



TRIFACTA

Application User Guide

Version: 9.2

Doc Build Date: 07/29/2022

Copyright © Trifacta Inc. 2022 - All Rights Reserved. CONFIDENTIAL

These materials (the “Documentation”) are the confidential and proprietary information of Trifacta Inc. and may not be reproduced, modified, or distributed without the prior written permission of Trifacta Inc.

EXCEPT AS OTHERWISE PROVIDED IN AN EXPRESS WRITTEN AGREEMENT, TRIFACTA INC. PROVIDES THIS DOCUMENTATION AS-IS AND WITHOUT WARRANTY AND TRIFACTA INC. DISCLAIMS ALL EXPRESS AND IMPLIED WARRANTIES TO THE EXTENT PERMITTED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE AND UNDER NO CIRCUMSTANCES WILL TRIFACTA INC. BE LIABLE FOR ANY AMOUNT GREATER THAN ONE HUNDRED DOLLARS (\$100) BASED ON ANY USE OF THE DOCUMENTATION.

For third-party license information, please select **About Trifacta** from the Help menu.

1. Workflow Basics	6
1.1 Object Overview	8
1.2 Import Basics	13
1.3 Profiling Basics	14
1.4 Transform Basics	17
1.5 Sampling Basics	27
1.6 Running Job Basics	31
1.7 Export Basics	33
2. Common Tasks	34
2.1 Import Tasks	37
2.1.1 Connect to Data	38
2.1.1.1 Share a Connection	40
2.1.2 Import a File	41
2.1.2.1 Change File Encoding	42
2.1.2.2 Remove Initial Structure	43
2.1.3 Import a Table	44
2.1.3.1 Disable Type Inference	45
2.1.4 Import from Another Flow	46
2.1.5 Import Excel Data	48
2.1.6 Import Google Sheets Data	51
2.1.7 Import PDF Data	54
2.1.8 Create Dataset with Parameters	58
2.1.8.1 Parameterize Files for Import	61
2.1.8.2 Parameterize Tables for Import	68
2.1.9 Create Dataset with SQL	73
2.2 Discovery Tasks	80
2.2.1 Explore Suggestions	81
2.2.2 Add or Edit Recipe Steps	85
2.2.3 Filter Data	86
2.2.4 Locate Outliers	88
2.2.5 Compute Counts	94
2.2.6 Calculate Metrics across Columns	98
2.2.7 Compare Strings	101
2.2.8 Analyze across Multiple Columns	105
2.2.9 Parse Fixed-Width File and Infer Columns	108
2.2.10 Generate a Sample	110
2.2.10.1 Change Recipe Sample Size	116
2.3 Validation Tasks	118
2.3.1 Profile Your Source Data	119
2.3.2 Validate Your Data	121
2.3.2.1 Validate Column Values against a Dataset	128
2.3.3 Find Bad Data	135
2.3.4 Find Missing Data	140
2.3.5 Manage Null Values	146
2.4 Structuring Tasks	148
2.4.1 Initial Parsing Steps	149
2.4.2 Reshaping Steps	153
2.4.3 Split Column	154
2.4.4 Move Columns	160
2.4.5 Delete Data	164
2.4.6 Select	167
2.4.7 Create Aggregations	169
2.4.8 Nest Your Data	172
2.4.9 Unnest Your Data	175
2.4.10 Pivot Data	184
2.4.11 Unpivot Columns	192
2.4.12 Window Transformations	195
2.4.13 Working with Arrays	205
2.4.14 Working with Objects	214
2.4.15 Working with JSON v2	224

2.4.16	Working with JSON v1	233
2.5	Cleanse Tasks	243
2.5.1	Rename Columns	244
2.5.2	Sanitize Column Names	253
2.5.3	Change Column Data Type	254
2.5.4	Copy and Paste Columns	258
2.5.5	Create Column by Example	259
2.5.6	Remove Data	261
2.5.7	Deduplicate Data	266
2.5.8	Compare Values	269
2.5.9	Replace Cell Values	271
2.5.10	Replace Values Using Patterns	273
2.5.11	Replace Groups of Values	280
2.5.12	Normalize Numeric Values	285
2.5.13	Standardize Using Patterns	291
2.5.14	Modify String Values	295
2.5.15	Manage String Lengths	306
2.5.16	Extract Values	309
2.5.17	Format Dates	318
2.5.18	Apply Conditional Transformations	325
2.5.19	Prepare Data for Machine Processing	328
2.6	Enrichment Tasks	333
2.6.1	Create New Column	337
2.6.2	Add Two Columns	339
2.6.3	Generate Primary Keys	342
2.6.4	Add Lookup Data	346
2.6.5	Append Datasets	349
2.6.6	Join Data	351
2.6.6.1	Configure Range Join	356
2.6.7	Insert Metadata	358
2.6.8	Invoke External Function	361
2.7	Publishing Tasks	363
2.7.1	Create Outputs	364
2.7.1.1	Create Output SQL Scripts	369
2.7.2	Publish Results on Demand	375
2.7.3	Reuse Recipe	376
2.8	Project Management Tasks	378
2.8.1	Take a Snapshot	379
2.8.2	Track Data Changes	381
2.8.3	Add Comments to Your Recipe	385
2.8.4	Create Target	386
2.8.5	Optimize Job Processing	388
2.8.6	Diagnose Failed Jobs	390
2.8.7	Schedule a Job	396
2.8.8	Create Branching Outputs	398
2.8.9	Build Sequence of Datasets	400
2.8.10	Fix Dependency Issues	403
2.8.11	Share a Flow	405
2.8.12	Export Flow	406
2.8.13	Import Flow	408
2.8.13.1	Reconnect Flow to Source Data	411
2.8.13.2	Reconnect Flow to Outputs	412
2.8.13.3	Define Import Mapping Rules	413
2.8.14	Create or Replace Macro	427
2.8.15	Apply a Macro	433
2.8.16	Export Macro	435
2.8.17	Import Macro	436
2.8.18	Create Flow Parameter	438
2.8.19	Flag for Review	445

2.8.20	<i>Manage Environment Parameters</i>	448
2.9	<i>Operationalization Tasks</i>	451
2.9.1	<i>Create Flow Webhook Task</i>	452
2.9.2	<i>Create a Plan</i>	459
2.9.2.1	<i>Create Delete Task</i>	467
2.9.2.2	<i>Create HTTP Task</i>	469
2.9.2.3	<i>Create Slack Task</i>	478
2.9.3	<i>Share a Plan</i>	480
2.9.4	<i>Export Plan</i>	481
2.9.5	<i>Import Plan</i>	482
2.10	<i>Account Management Tasks</i>	483
2.10.1	<i>Change Password</i>	484
2.10.2	<i>Configure Your Access to S3</i>	485
3.	<i>Concepts</i>	488
3.1	<i>Feature Overviews</i>	489
3.1.1	<i>Overview of Data Export</i>	490
3.1.2	<i>Overview of Data Import</i>	494
3.1.3	<i>Overview of Storage</i>	498
3.1.4	<i>Overview of Predictive Transformation</i>	503
3.1.5	<i>Overview of the Type System</i>	511
3.1.6	<i>Overview of Schema Management</i>	521
3.1.7	<i>Overview of Standardization</i>	525
3.1.8	<i>Overview of Cluster Clean</i>	530
3.1.9	<i>Overview of Visual Profiling</i>	534
3.1.10	<i>Overview of Sampling</i>	539
3.1.11	<i>Overview of Job Execution</i>	544
3.1.11.1	<i>Trifacta Photon Running Environment</i>	551
3.1.11.2	<i>EMR Running Environment</i>	552
3.1.11.3	<i>Snowflake Running Environment</i>	553
3.1.11.4	<i>AWS Databricks Running Environment</i>	559
3.1.11.5	<i>Azure Databricks Running Environment</i>	560
3.1.11.6	<i>Hadoop Spark Running Environment</i>	561
3.1.12	<i>Overview of TBE</i>	562
3.1.13	<i>Overview of Data Quality</i>	565
3.1.14	<i>Overview of Sharing</i>	570
3.1.15	<i>Overview of Job Monitoring</i>	576
3.1.16	<i>Overview of Automator</i>	582
3.1.17	<i>Overview of Parameterization</i>	584
3.1.18	<i>Overview of Authorization</i>	597
3.1.19	<i>Overview of Operationalization</i>	600
3.1.20	<i>Overview of Macros</i>	606
3.1.21	<i>Overview of Deployment Manager</i>	610
3.1.22	<i>Overview of Pattern Matching</i>	618
3.1.23	<i>Overview of RapidTarget</i>	622
3.2	<i>Using Connections</i>	625
3.2.1	<i>Using Databases</i>	626
3.2.2	<i>Using HDFS</i>	628
3.2.3	<i>Using S3</i>	632
3.2.4	<i>Using SQL DW</i>	636

Workflow Basics

Contents:

- *Overview*
 - *Prerequisites*
 - *Basic Workflow*
-

Learn the basics of how to import, wrangle, execute jobs, profile, and export your data from Trifacta®.

Overview

Trifacta® enables analysts, data specialists, and other domain experts to quickly cleanse and transform datasets of varying sizes for use throughout the enterprise. Using an innovative set of web-based tools, you can import complex datasets and wrangle them for use in virtually any target system. Key capabilities include:

- **Import** from flat file, databases, or distributed storage systems
- Locate and remove or modify **missing or mismatched** data
- **Unnest complex** data structures
- Identify statistical **outliers** in your data for review and management
- Perform **lookups** from one dataset into another reference dataset
- Aggregate columnar data using a variety of **aggregation functions**
- **Normalize** column values for more consistent usage and statistical modeling
- Merge datasets with **joins**
- Append one dataset to another through **union** operations

Most of these operations can be executed with a few mouse clicks. This section provides a basic overview of common workflows through Trifacta.

Prerequisites

Before you begin, please verify the following:

- **Trifacta account:** You have a Trifacta account and can login.
- **Example data:** You should use a sample set of data during this workflow.

Basic Workflow

1. **Import data:** Integrate data from a variety of sources of data.

Tip: When you login for the first time, you can immediately upload a dataset to begin transforming it.

See *Import Basics*.

2. **Profile your data:** Before, during, and after you transform your data, you can use the visual profiling tools to quickly analyze and make decisions about your data. See *Profiling Basics*.
3. **Build transform recipes:** Use the various views in the Transformer Page to build your transform recipes and preview the results on sampled data. See *Transform Basics*.
4. **Sample your data:** In Trifacta, you create your recipes while working with a sample of your overall dataset. As needed, you can take new samples, which can provide new perspectives and enhance performance in complex flows. See *Sampling Basics*.

5. **Run job:** Launch a job to run your recipe on the full dataset. Review results and iterate as needed. See *Running Job Basics*.
6. **Export results:** Export the generated results data for use outside of Trifacta. See *Export Basics*.

Object overview: You should review the overview of the objects that are created and maintained in Trifacta. See *Object Overview*.

Object Overview

Contents:

- *Flow Structure and Objects*
 - *Flow*
 - *Imported Dataset*
 - *Recipe*
 - *Flow Example*
 - *Working with recipes*
 - *Connections*
 - *Flow Schedules*
 - *Plans*
-

Explore the objects that you create and their relationships. Flows, imported datasets, and recipes are created to transform your sampled data. After you build your output objects, you can run a job to transform the entire dataset based on your recipe and deliver the results according to your output definitions.

Flow Structure and Objects

Within Trifacta®, the basic unit for organizing your work is the flow. The following diagram illustrates the component objects of a flow and how they are related:

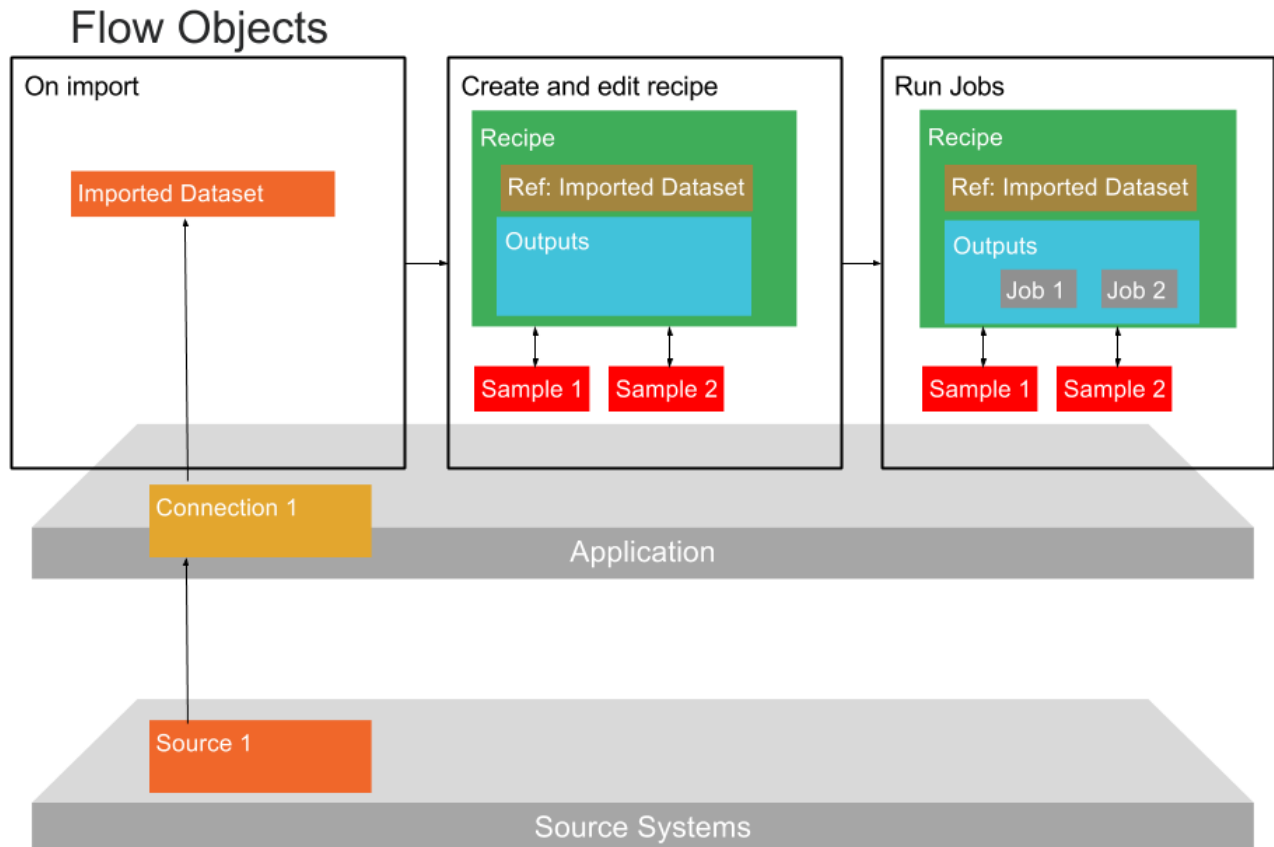


Figure: Objects in a Flow

Flow

A **flow** is a container for holding one or more datasets, associated recipes and other objects. This container is a means for packaging Trifacta objects for the following types of actions:

- Creating relationships between datasets, their recipes, and other datasets.
- Copying
- Execution of pre-configured jobs
- Creating references between recipes and external flows

Imported Dataset

Data that is imported to the platform is referenced as an imported dataset. An **imported dataset** is simply a reference to the original data; the data does not exist within the platform. An imported dataset can be a reference to a file, multiple files, database table, or other type of data.

NOTE: An imported dataset is a pointer to a source of data. It cannot be modified or stored within Trifacta.

- An imported dataset can be referenced in recipes.
- Imported datasets are created through the Import Data page.
- For more information on the process, see *Import Basics*.

After you have created an imported dataset, it becomes usable after it has been added to a flow. You can do this as part of the import process or later.

Recipe

A **recipe** is a user-defined sequence of steps that can be applied to transform a dataset.

- A recipe object is created from an imported dataset or another recipe. You can create a recipe from a recipe to chain together recipes.
- Recipes are interpreted by Trifacta and turned into commands that can be executed against data.
- When initially created, a recipe contains no steps. Recipes are augmented and modified using the various visual tools in the Transformer page.
- For more information on the process, see *Transform Basics*.

In a flow, the following objects are associated with each recipe, which are described below:

- Outputs
- References

Outputs and Publishing Destinations

Outputs contain one or more publishing destinations, which define the output format, location, and other publishing options that are applied to the results generated from a job run on the recipe.

When you select a recipe's output object in a flow, you can:

- Define the publishing destinations for outputs that are generated when the recipe is executed. **Publishing destinations** specify output format, location, and other publishing actions. A single recipe can have multiple publishing destinations.
- Run an on-demand job using the specified destinations. The job is immediately queued for execution.

Reference Datasets

When you select a recipe's reference object, you can add it to another flow. This object is then added as a reference dataset in the target flow. A **reference dataset** is a read-only version of the output data generated from the execution of a recipe's steps.

Flow Example

The following diagram illustrates the flexibility of object relationships within a flow.

Flow Example

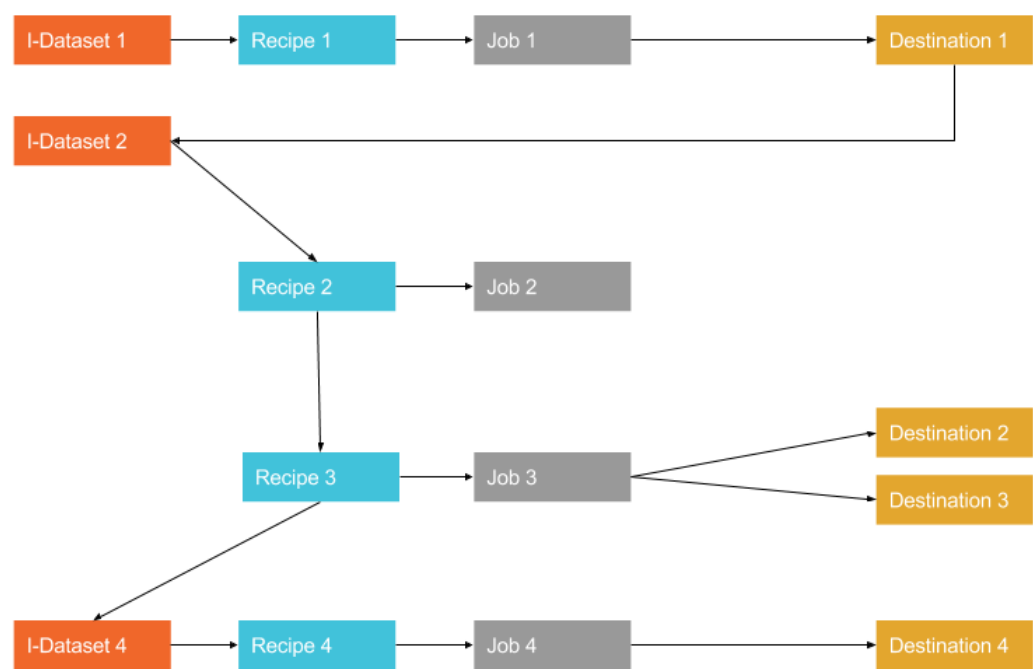


Figure: Flow Example

Type	Datasets	Description
Standard job execution	Recipe 1 /Job 1	Results of the job are used to create a new imported dataset (I-Dataset 2) from the Job Details page.
Create dataset from generated results	Recipe 2 /Job 2	Recipe 2 is created off of I-Dataset 2 and then modified. A job has been specified for it, but the results of the job are unused.
Chaining datasets	Recipe 3 /Job 3	Recipe 3 is chained off of Recipe 2. The results of running jobs off of Recipe 2 include all of the upstream changes as specified in I-Dataset 1/Recipe1 and I-Dataset 2/Recipe 2.
Reference dataset	Recipe 4 /Job 4	I-Dataset 4 is created as a reference off of Recipe 3. It can have its own recipe, job, destinations, and results.

Flows are created in the Flows page.

Working with recipes

Recipes are edited in the Transformer page, which provides multiple methods for quickly selecting and building recipe steps.

Samples: Within the Transformer page, you build the steps of your recipe against a **sample** of the dataset.

- A sample is typically a subset of the entire dataset. For smaller datasets, the sample may be the entire dataset.
- As you build or modify your recipe, the results of each modification are immediately reflected in the sampled data. So, you can rapidly iterate on the steps of your recipe within the same interface.
- As needed, you can generate additional samples, which may offer different perspectives on the data.
- See *Sampling Basics*.

Macros: As needed, you can create reusable sequences of steps that can be parameterized for use in other recipes.

Run Jobs: When you are satisfied with the recipe that you have created in the Transformer page, you can execute a **job**. A job may be composed of one or more of the following job types:

- **Transform job:** Executes the set of recipe steps that you have defined against your sample(s), generating the transformed set of results across the entire dataset.
- **Profile job:** Optionally, you can choose to generate a visual profile of the results of your transform job. This visual profile can provide important feedback on data quality and can be a key for further refinement of your recipe.
- When a job completes, you can review the resulting data and identify data that still needs fixing in the Job Details page.
- For more information on the process, see *Running Job Basics*.

Connections

A **connection** is a configuration object that provides a personal or global integration to an external datastore. Reading data from remote sources and writing results are managed through connections.

- Connections are not associated with individual datasets or flows.
 - Connections are not reflected in the above diagram.
- Most connections can be created by individual users and shared as-needed.
- Depending on the datastore, connections can be read-only, write-only, or both.
- Connections are created in the Connections page.

Flow Schedules

You can associate a schedule with a flow. A **schedule** is a combination of one or more triggers and the outputs that are generated from them.

NOTE: A flow can have only one schedule associated with it.

- A **trigger** is a scheduled time of execution. When a trigger's time occurs, all of the scheduled output destinations are queued for generation.
 - A schedule can have multiple triggers associated with it. Therefore, a flow can be scheduled for execution at multiple intervals.
- A **scheduled destination** is an output associated with a recipe. This output is generated only when the schedule for the flow is triggered.
 - A scheduled destination is not tied to a specific trigger. When a trigger occurs, all scheduled destinations in the flow are generated.
 - A scheduled destination generates one or more publishing actions (outputs) from the recipe when triggered.

- A recipe can have only one scheduled destination.
- Each recipe in a flow can have a scheduled destination.
- If a flow has a trigger but no scheduled destination, nothing is generated at trigger time.

Below, you can see the object hierarchy within a schedule.

```
+ schedule for Flow 1
+ trigger 1
+ trigger 2
+ scheduled destination a
+ scheduled destination b
+ schedule for Flow 2
+ trigger 3
+ scheduled destination c
+ scheduled destination d
```

Schedules are created for a flow through Flow View page.

Plans

A **plan** is a sequence of triggers and tasks that can be executed across multiple flows. A plan is executed on a snapshot of all objects at the time that the plan is triggered.

- A **task** is an executable action that is taken as part of a plan's sequence. For example, task #1 could be to execute a flow that imports all of your source data. Task #2 executes the flow that cleans and combines that data. Example task types:
 - A **flow task** is the execution of the recipes in a specified flow, which result in the generation of one or more selected outputs.
 - A **trigger** for a plan is the schedule for its execution.
 - A **snapshot** is a frozen image of the plan. This snapshot of the plan defines the objects that are executed as part of a plan run.
 - An **HTTP task** is a request submitted by the product to a third-party server as part of the sequence of tasks in a plan. For example, an HTTP task could be the submission of a message to a channel in your enterprise's messaging system.

Plans are created through the Plans page.

Import Basics

Trifacta® can import data from a variety of flat file formats and other distributed sources.

NOTE: Trifacta does not modify a source. Instead, a set of metadata is associated with the source data, which enables transformation of the source. On export, a new version of the data is written to one or more specified output destinations.

Steps:

When data is imported, a reference to it is stored by the platform as an imported dataset. The source data is not modified. In the application, you modify the recipe associated with a dataset to transform a sample of the imported data.

NOTE: Any user with a valid user account can import data from a local file.

1. Login to the application.
2. In the menubar, click **Library**. Click **Import Data**.
3. To add a dataset:
 - a. Select the connection where your source is located.
 - b. Upload:
 - i. Select **Upload** to upload a file from your local desktop. You can select multiple files to upload. For this example, select only one file.
 - ii. Navigate and select the file or files for your source. Click **Open**.
 - c. Backend storage, such as S3:
 - i. Navigate and select the file or files for your source.
 - ii. To queue the dataset for uploading, click the Plus icon next to its name.
 - iii. You can select multiple files.
 - d. Select the Add to new flow checkbox. This option creates a new flow, which is a container object for your Trifacta assets. Your imported dataset is added to it.
4. To begin working with your dataset, click **Continue**.
5. The imported dataset and its containing flow are created.
6. You can begin working with the dataset in the Transformer page. For more information, see *Transform Basics*.

Tip: If you are interested, you can create a visual profile of your source data before you begin transforming. For more information, see *Profiling Basics*.

Profiling Basics

Contents:

- *Profiling Source Data*
 - *Profiling in the Application*
 - *Status Bar*
 - *Column Header*
 - *Column Histogram*
 - *Column Details - statistics and outliers*
 - *Column Browser - profiles across columns*
 - *Profiling in Job Results*
 - *Download visual profile*
-

Trifacta® surfaces visual representations of your data for individual columns and the entire dataset and provides mechanisms for taking immediate action on issues in the data.

Profiling Source Data

When you first load your dataset into the application, you might want to run a job to profile your dataset before you build your recipe. The generated results and profile are accessible through the Job Details page in the Trifacta application. This profile of your source can be useful later in seeing how your dataset has changed during development.

Profiling in the Application

When you identify something of interest in the Trifacta application, you can select the visual representation of it, and the platform prompts you with a set of suggested transforms to add to your recipe. These visual profiles enable you to make quick assessments of problems, unusual patterns, and required changes to your data.

NOTE: Before your job is run, profiling information such as column statistics are exact counts of the sample that is currently loaded. After the job is run, profiled results in the Job Results page might include estimates for some metrics and counts, depending on the scale of the dataset.

Status Bar

The number of rows, columns, and data types in the current sample are displayed at the bottom of the page in the status bar.

Column Header

The top of each column contains a data quality bar, which identifies the valid, mismatched, and missing values in the column when compared against the specified data type, and column histogram, which identifies the range of values in the column.

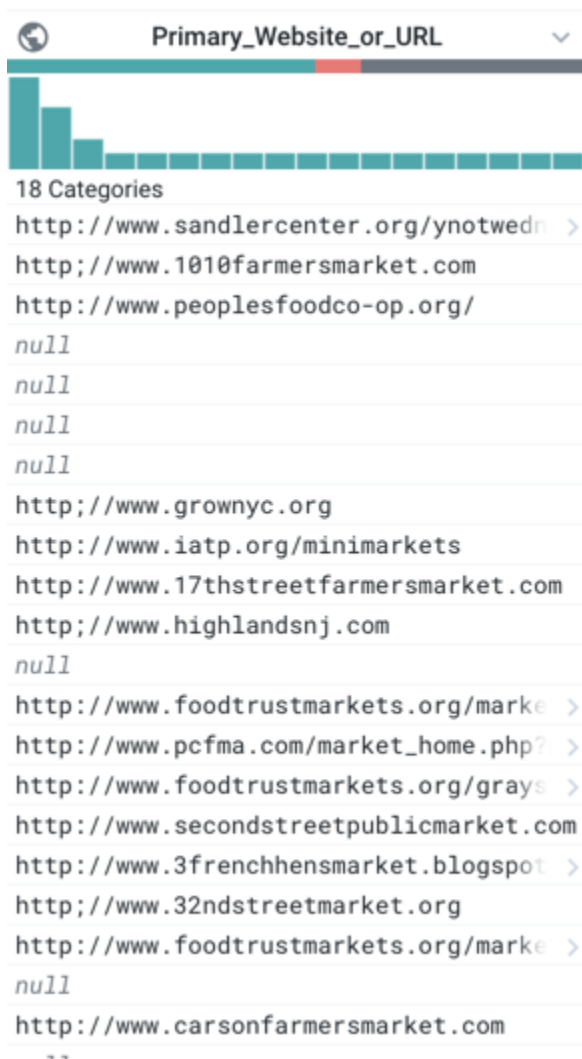


Figure: Example Column

Data Quality Bar - missing and mismatches values

Below the name of the column, the multi-colored band indicates the valid (green), mismatched (red), and missing (gray) values in the column, when matched against the column's data type. Click the missing or mismatched values in a column's data quality bar. You are prompted with suggestions of transformations to fix or remove these values.

Column Histogram

The bar chart at the top of each column in the Transformer page is called a histogram. Each column histogram displays the count of each detected value in the column (for string data) or the count of values within a numeric range (for number data). You can use this histogram to identify unusual values or outlier values, which can be corrected or removed.

Column Details - statistics and outliers

In the Column Details window, you can review key statistical information on the values in a column. Displayed statistics are based on the column's data type. Select Column Details from the drop-down for the specific column in the data grid.

Column Browser - profiles across columns

In the column browser, you can view visual histograms for each column in the dataset and make selections to identify correlations between values in multiple columns. To open the column browser, click the Columns icon in the Transformer bar.

Profiling in Job Results

When you run a job, you can choose to generate a visual profile based on the job results from the Run Job page.

Download visual profile

From the **Profiles** tab, you can download your job's visual profile to your desktop.

Transform Basics

Contents:

- *Goal*
 - *Recommended Methods for Building Recipes*
 - *Sample*
 - *Cleanse*
 - *Modify*
 - *Enrichment*
 - *Sampling*
 - *Profile*
-

When you edit your dataset's recipe, the Transformer page is opened, where you begin your wrangling tasks on a sample of the dataset. Through this interface, you build your transformation recipe and see the results in real-time as applied to the sample. When you are satisfied with what you see, you can execute a job against the entire dataset.

Goal

Your data transformation is complete when you have done the following:

- Cleansed your data of invalid, missing, or inaccurate values
- Enhanced your dataset as needed with data from other datasets
- Modified your dataset to constrain its values to meet the target schema
- Executed job against the entire dataset
- Exported the results from your dataset and recipe for use in downstream systems

Tip: Before you begin transforming, you should know the target schema that your transformed data must match. A **schema** is the set of columns and their data types, which define the constraints of your dataset. You can import this target schema as a dataset and use it during recipe development to serve as a mapping for your transformations.

Recommended Methods for Building Recipes

Trifacta® supports the following methods for building recipes in the Transformer page. These methods are listed in order of ease of use:

1. **Select something.** When you select elements of data in the Transformer page, you are prompted with a set of suggestions for steps that you can take on the selection or patterns matching the selection. You can select columns or one or more values within columns.

Tip: The easiest method for building recipes is to select items in the application. Over time, the application learns from your selections and prompts you with suggestions based on your previous use.

2. **Toolbar and column menus:** In the Transformer page, you can access pre-configured transformations through the Transformer toolbar or through the column context menus.

Tip: Use the toolbar for global transformations across your dataset and the column menu for transformations on one or more selected columns.

- a. When a Transformer toolbar item is selected, the Transform Builder is pre-populated with settings and values to get you started. As needed you can modify the step to meet your needs.
 - b. The column menus contain the most common transformations for individual or multiple columns. Often, no additional configuration is required.
 - c. Select multiple columns. Continue selecting columns to be prompted with a different set of suggestions applicable to all of them.
3. **Search and browse for transformations.** Using the Search panel and the Transform Builder, you can rapidly assemble recipe steps through a simple, menu-driven interface. When you choose to add a step, you search for your preferred transformation in the Search panel. When one is selected, the Transform Builder is pre-populated from your selection in the Search panel.

Tip: Use the Transform Builder for performing modifications to the transformation you selected from the Search panel or a suggestion card.

Sample

Loading very large datasets in Trifacta can overload your browser or otherwise impact performance, so the application is designed to work on a sample of data. After you have finished your recipe working on a sample, you execute the recipe across the entire dataset.

The default sample is the first set of rows of source data in the dataset, the number of which is determined by the platform. For smaller datasets, the entire dataset can be used as your sample. In the Transformer page, it's listed as **Initial Data** in the upper-left corner.

In some cases, the default sample might be inadequate or of the wrong type. To generate a new sample, click the name of the sample in the upper-left corner.

NOTE: Collecting new samples requires system resources and storage. In some environments, collecting samples incurs monetary cost.

Tip: You should consider collecting a new sample if you have included a step to change the number of rows in your dataset or have otherwise permanently modified data (keep, delete, lookup, join, or pivot operations). If you subsequently remove the step that made the modification, the generated sample is no longer valid and is removed. This process limits unnecessary growth in data samples.

On the right side of the Transformer page, you can launch a new sampling job on your dataset from the Samples panel. You may have to open it first.

Cleanse

Data cleansing tasks address issues in data quality, which can be broadly categorized as follows:

- **Consistency.** Values that describe the same thing should agree with each other. For example, numeric values should have the same precision. String values should be consistently structured to mean the same thing.
- **Validity.** Values should be constrained to the requirements of each field's data type. For example, a DateOfSale field should be a valid date.
- **Reliability.** Values in the same field in different records should mean the same thing. For example, the value 15 in the Temperature field of two different records should not mean Centigrade in one record and Fahrenheit in the other record.

When data is initially imported, it can contain multiple columns, rows, or specific values that you don't need for your final output. Specifically, this phase can involve the following basic activities:

- Remove unused columns
- Address missing and mismatched data
- Change data types
- Improve consistency, validity, and reliability of the data

NOTE: An imported dataset requires about 15 rows to properly infer column data types and the row, if any, to use for column headers.

First recipe steps:

When a dataset sample is first loaded into the Transformer page, Trifacta attempts to split out the unstructured data to form regular, tabular data. If your data appears to contain a header row, it can be used for the titles of the columns.

The screenshot displays the Trifacta Transformer interface. The main area shows a data grid with 4 columns and 91,561 rows. The columns are labeled 'column2', 'column3', 'column4', and 'column5'. The data grid shows various data types including categories, timestamps, and strings. On the right, the Recipe panel is visible, showing a message 'The recipe is empty' and a button 'Add New Step'.

Figure: Transformer page

In the above image, some initial parsing steps have been applied to structure the data into tabular format, but these steps are not added as formal parts of the recipe. They are hidden from view in the recipe. By default, these steps are automatically added to the recipe when you permit the application to detect the structure of the imported data.

The data resulting from these initial transforms is displayed in the **data grid**.

- Your recipe is displayed in the Recipe panel on the right side. You might have to open this panel to see it.
- When you select items in the data grid, suggestion cards are displayed for you to begin building transform steps.
- These suggestions can be modified to build more complex or subtle commands in the Transform Builder.
- Don't forget to use the Transformer toolbar, which pre-configures the Transform Builder with the configuration required for a useful transformation.
- You can use the column menu to apply changes to an individual column.

Use a row to create headers:

In most cases, the names of your columns are inferred from the first row of the data in the dataset. If you need to specify a different row, please complete the following:

1. Click the Search icon in the menu bar.

2. In the Search panel textbox, type: header
3. The transformation is displayed in the Transform Builder. Specify the following properties:

Transformation Name	Rename columns
Parameter: Option	Use row as header
Parameter: Row	1

4. If you need to specify a different row to use, you can specify a specific row number to use in the Row textbox.
5. To add this or any transform in development to your recipe, click **Add**. This button is disabled if the step is invalid.

Generate metadata:

On the left side of the data grid, you might notice a set of black dots. If you hover over one of these, the original row number from the source data is listed. Since the data transformation process can change the number of rows or their order, you might want to retain the original order of the rows.

Tip: Some operations, such as unions and joins, can invalidate source row number information. To capture this data into your dataset, it's best to add this transformation early in your recipe.

To retain the original row numbers in a column called, `rowId`, please complete the following:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>\$sourcerownumber</code>
Parameter: New column name	<code>rowId</code>

You can use a similar transformation to generate the full path and filename to file-based sources:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>\$filepath</code>
Parameter: New column name	<code>filepath</code>

Delete unused columns:

Your data might contain columns that are not of use to you, so it's in your interest to remove them to simplify the dataset. To delete a column, click the caret next to the column's title and select **Delete**.

Tip: If you are unsure of whether to delete the column, you can use the same caret menu to hide the column for now. Hidden columns do appear in the output.

Tip: You can also delete multiple columns, including ranges of columns.

Check column data types:

When a dataset is imported, Trifacta attempts to identify the data type of the column from the first set of rows in the column. At times, however, type inference can be incorrect.

Tip: Before you start performing transformations on your data based on mismatched values, you should check the data type for these columns to ensure that they are correct.

Display only columns of interest:

You can choose which columns you want to display in the data grid, which can be useful to narrow your focus to problematic areas.

In the Status bar at the bottom of the screen, click the Eye icon.

Review data quality:

After you have removed unused data, you can examine the quality of data within each column just below the column title.

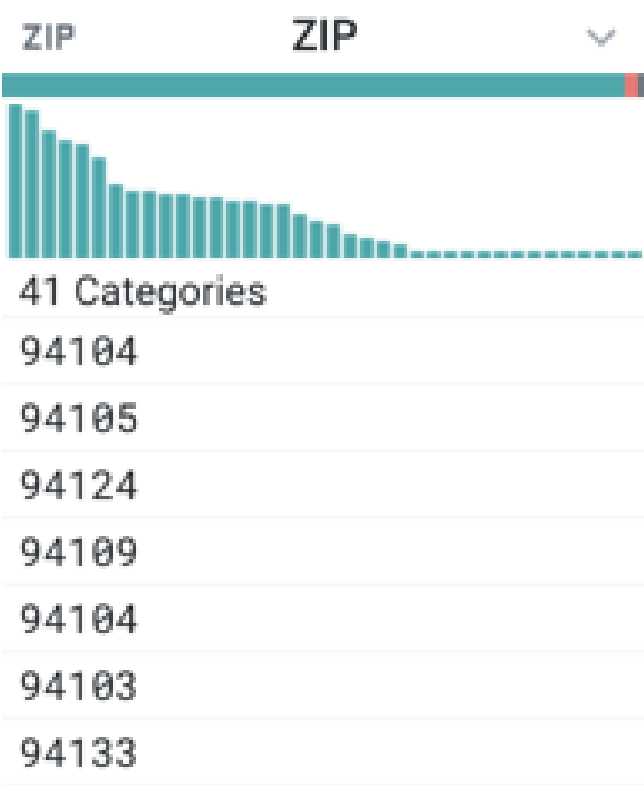


Figure: Column header with data quality bar

The horizontal bar, known, as the **data quality bar**, identifies the quality of the data in the column by the following colors:

Color	Description
green	These values are valid for the column data type.
red	These values do not match those of the column type.

gray There are no values for the column in these rows.

Tip: When you select values in the data quality bar, those values are highlighted in the sample rows, and suggestions are displayed at the bottom of the screen in the suggestion cards to address the selected rows.

Suggestion Cards:

Based on your selections and its knowledge of common data patterns, Trifacta prompts you with suggested transformations. You can then select pre-configured transformations in the right panel of the Transformer page to quickly add steps.

Tip: Where possible, you should try to create your transforms by selecting data and then selecting the appropriate suggestion card. In some cases, you might need to modify the details of the recipe.

In the following example, the missing values in the SUBSCRIBER_AGE column have been selected, and a set of suggestion cards is displayed.

The screenshot displays the Trifacta Transformer interface. On the left, a 'Preview' panel shows a data table with columns: RACT_START, #, SUBSCRIBER_AGE, STATUS, WEB_CHAT_ID, and COUNTRY. The SUBSCRIBER_AGE column is highlighted with a blue bar, indicating selected data. Below the table, a status bar shows '19 Columns', '20,000 Rows', and '8 Data Types'. On the right, a 'Suggestions' panel lists three cards: 'Delete rows', 'Keep rows', and 'Create a new column'. The 'Delete rows' card is selected and shows the suggestion 'with missing values in SUBSCRIBER_AGE'. The 'Keep rows' card also shows the same suggestion. The 'Create a new column' card shows the suggestion 'flag missing values in SUBSCRIBER_AGE'. Below these cards, a 'Set' section shows 'missing values to NULL()' and 'missing values to 0'.

Figure: Selecting missing values

Tip: When previewing a recipe step, you can use the checkboxes in the status bar to display only affected rows, columns, or both, which helps you to assess the effects of your step.

Depending on the nature of the data, you might want to keep, delete, or modify the values. Since the data is missing, the Delete card has been selected.

- To accept this suggest, click **Add**.
- You can modify the step if needed. An example is provided later.

For more background information, see *Overview of Predictive Transformation*.

Change data types:

If a column contains a high concentration of mismatched data (red), the column might have been identified as the wrong data type. For example, your dataset includes internal identifiers that are primarily numeric data (e.g. 10000022) but have occasional alphabetical characters in some values (e.g. 1000002A). The column for this data might be typed for integer values, when it should be treated as string values.

Tip: Trifacta maintains statistical information and enable some transformation steps based upon data type.

- 1. To change a column's data type, click the icon to the left of the column title.
- 2. Select the new data type.
- 3. Review the mismatched values for the column to verify that their count has dropped.

Explore column details:

As needed, you can explore details about the column's data, including statistical information such as outliers. From the caret drop-down next to a column name, select **Column Details**.

Review histograms:

Just below a column's data quality bar, you can review a histogram of the values found in the column. In the following example, the data histogram on the left applies to the `ZIP` column, while the one on the right applies the `WEB_CHAT_ID` column.

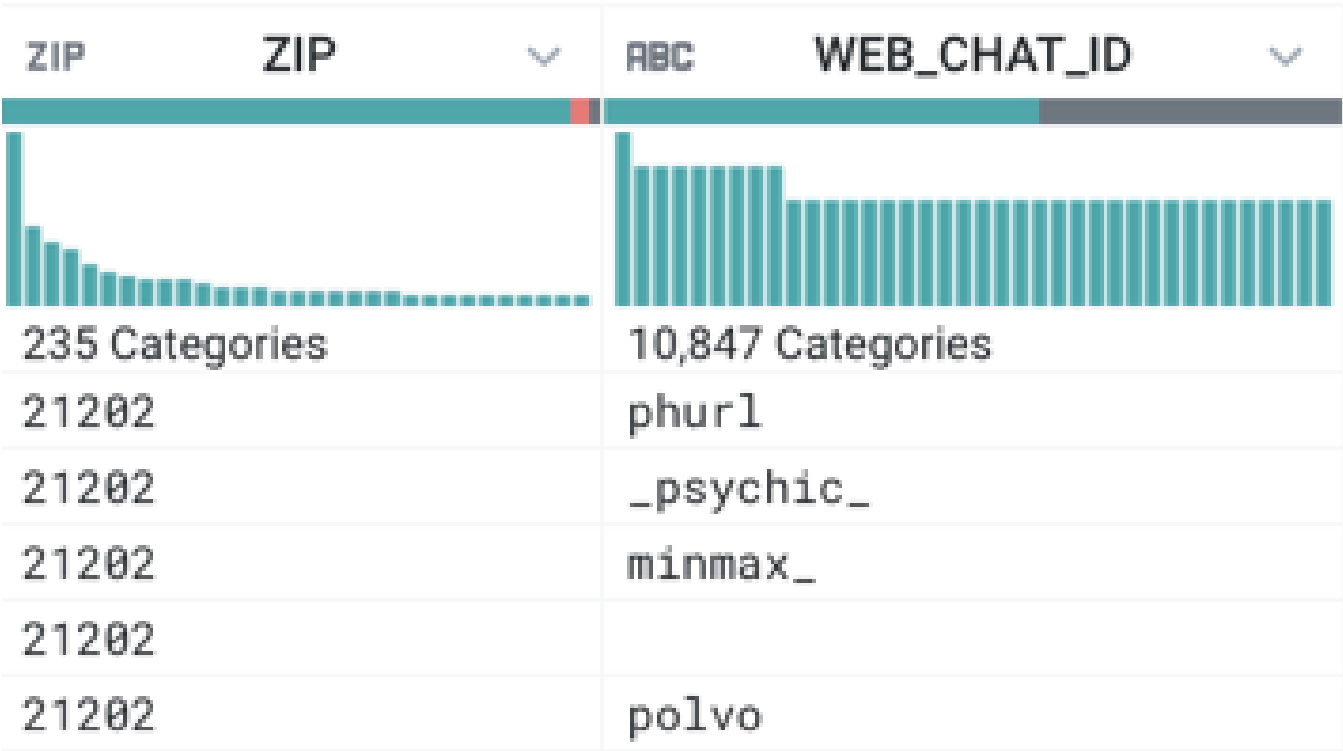


Figure: Column data histogram

When you mouse over the categories in the histogram, you can see the corresponding value, the count of instances in the sample's column, and the percentage of affected rows. In the left one, the bar with the greatest number of instances has been selected; the value 21202 occurs 506 times (21.28%) in the dataset. On the right, the darker shading indicates how rows with `ZIP=21202` map to values in the `WEB_CHAT_ID` column.

Tip: Similar to the data quality bar, you can click values in a data histogram to highlight the affected rows and to trigger a set of suggestions. In this manner, you can use the same data quality tools to apply even more fine-grained changes to individual values in a column.

Modify

After you have performed initial cleansing of your data, you might need to perform modifications to the data to properly format it for the target system, specify the appropriate level of aggregation, or perform some other modification.

In the following example, the improperly capitalized word `BALTIMORE` has been selected, so that you can change it to its propercase spelling (`Baltimore`). Those rows are highlighted in the row data, and a set of suggestions for how to fix has been provided in the Selection Details panel.

The screenshot shows a data table with columns: LAST_NAME, SSN, ADDRESS, CITY, and STATE. The 'CITY' column has a histogram showing a distribution of values. The 'Suggestions' panel on the right shows a 'Replace' suggestion: 'first occurrence of `'(start){upper}+(end)'` with ' in CITY'. Below this, it shows 'first occurrence of `'(start){upper}(9){end}'` with ' in CITY'. The suggestion is 'BALTIMORE' with ' in CITY'. There are 'Edit' and 'Add' buttons. Below the suggestion, there are sections for 'Extract values matching', 'Split on values matching', and 'Count values matching', each with a 'See all' link.

Figure: Selecting values to modify

Depending on the nature of your data, you might want to keep or change the values, or you can remove the problematic rows altogether.

Tip: When you select one of the suggestion cards, the implied changes are previewed in the Transformer page, so you can see the effects of the change. This previewing capability enables you to review and tweak your changes before they are formally applied. You can always remove a transform step if it is incorrect or even re-run the recipe to generate a corrected set of results, since source data is unchanged.

In this case, select the Replace transformation. However, there are a couple of minor issues with the provided suggestion.

- Since the platform has no idea about the meaning of the selection, it might initially suggest removing the text altogether. In this case, you want to change the spelling.
- In the transformation, the Find parameter value contains the pattern used to identify the selection. In this case, it is selecting all values that are capitalized. For now, you only want to fix `BALTIMORE`.

So, some aspects of this transform must be changed. Click **Edit**.

Transform Builder:

When you modify a transform step, you can make changes in the Transform Builder, which is a simple, menu-driven interface for modifying your transformations:

The screenshot displays the Transform Builder interface. On the left, a data table is shown with columns: LAST_NAME, SSN, ADDRESS, RBC, CITY, and STATE. The table contains 20,000 rows of data. On the right, a configuration panel titled 'Replace text or patterns' is visible. It includes a 'Column' dropdown set to 'CITY', a 'Find' input field containing 'BALTIMORE', and a 'Replace with' input field. The 'Replace with' field is currently blank. Below these fields are 'Advanced options' and 'Cancel' and 'Add' buttons. The bottom status bar indicates '20 Columns', '20,000 Rows', and '8 Data Types'.

Figure: Modifying steps in the Transform Builder

In the Transform Builder, you can replace the pattern with the specific string to locate: BALTIMORE. The new value, which is currently blank, can be populated with the replacement value: Baltimore. Click **Add**.

The step is added to the recipe and automatically applied to the data sample displayed in the Transformer page. You can continue to add new steps through the Transform Builder.

Enrichment

Before you deliver your data to the target system, you might need to enhance or augment the dataset with new columns or values from other datasets.

Union datasets:

You can append a dataset of identical structure to your currently loaded one to expand the data volume. For example, you can string together daily log data to build weeks of log information using the Union page.

Join datasets:

You can also join together two or more datasets based on a common set of values. For example, you are using raw sales data to build a sales commission dataset:

- Your sales transaction dataset contains a column for the salesman's identifier, which indicates the employee who should receive the commission.
- You might want to join your sales transaction dataset to the employee dataset, which provides information on the employee's name and commission rate by the internal identifier.
- If there is no corresponding record in the employee dataset, a commission is not rewarded, and the sales transaction record should not be present in the commission dataset.

This commission dataset is created by performing an inner join between the sales transaction dataset and the employee dataset. In the Search panel, enter `join` to join data.

Lookup values:

In some cases, you might need to include or replace values in your dataset with other columns from another dataset. For example, transactional data can reference product and customer by internal identifiers. You can create lookups into your master data set to retrieve user-friendly versions of customer and product IDs.

NOTE: The reference data that you are using for lookups must be loaded as a dataset into Trifacta first.

To perform a lookup for a column of values, click the caret drop-down next to the column title and select **Lookup...**.

Sampling

The data that you see in the Transformer page is a sample of your entire dataset.

- If your dataset is small enough, the sample is the entire dataset.
- For larger datasets, Trifacta auto-generates an initial data sample from the first rows of your dataset.

For larger datasets, you must learn how to generate new samples, which can provide different perspectives on your data and, in complex flows, enhance performance.

Tip: Sampling is an important concept in Trifacta.

Profile

As part of the transformation process, you can generate and review visual profiles of individual columns and your entire dataset. These interactive profiles can be very helpful in identifying anomalies, outliers, and other issues with your data.

Sampling Basics

Contents:

- *Initial Data*
 - *Take a Sample*
 - *Sampling and Memory*
 - *Sampling Considerations*
 - *Invalid samples*
 - *Best Practices*
-

A **sample** is a selection of rows from your dataset, which can be used as the basis for building the transformation steps in your recipe. The Trifacta® application automatically creates initial data samples of your data whenever you create a new recipe for a dataset and enables you to create additional samples at any time using a variety of sampling techniques.

Initial Data

When you create a new recipe and load it in the Transformer page, the Trifacta application displays the initial data sample of the dataset. The **initial data** consists of the first X rows of the datasets, where X is determined by the following factors:

- The number of columns in the dataset
- The amount of data in each cell
- The maximum permitted size of each sample

Take a Sample

These first rows are displayed for you to begin your work in the Transformer page. However, you may begin to run into limitations with this sample. For example, suppose your dataset is organized by date, with earliest dates listed first. There may be significant changes in the data later in the time period that do not appear in the initial sample. You may decide that you need to take a different sample that captures some of these changes.

Steps:

1. In the Transformer page, click the Eyedropper icon at the top of the page.

2. The Samples panel is displayed.

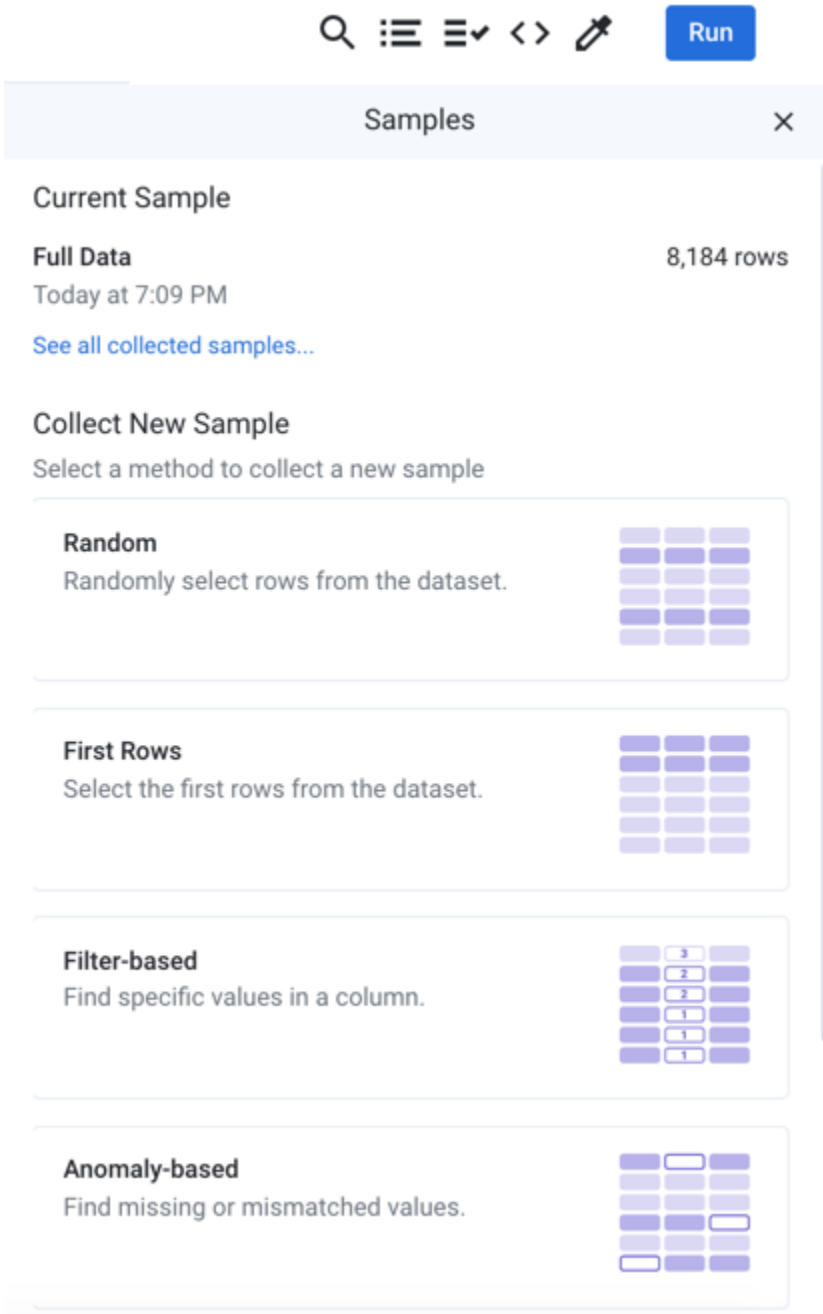


Figure: Samples panel

3. At the top of the panel, you can review the Current Sample.

Tip: In some cases, then the entire dataset is displayed in the data grid. Unless you wish to use a specific sampling technique to filter down your data, sampling may not be useful across the entire dataset.

4. Below the current sample, you can see the available sample types. To take a new random sample:
 - a. Click the Random card.
 - b. Depending on your product edition, you may be able to select Quick Scan or Full Scan.
 - i. Quick Scan creates your sample by making some assumptions about the data when it scans.

- ii. Full Scan creates your sample by scanning across all rows of the dataset. This option can take awhile across a large dataset.
- c. Click **Collect**.
5. The sampling job is queued for execution. When it completes, click **Load Sample**.
6. The data grid is refreshed to display the rows gathered in the new random sample.

Sampling and Memory

NOTE: After you generate a sample, all steps in a recipe that occur after the step selected when you generated the sample are executed in browser memory on the sample data and then displayed in the data grid.

The above statement is best explained by example:

Action	Sampling
1. Create a new recipe and open it in Transformer page.	The initial sample is generated and displayed.
2. Add 3 steps to your recipe.	The 3 new steps are applied to the initial sample in the browser's memory.
3. Generate a new random sample.	The random sample is generated. When you load the sample, it is displayed in the data grid.
4. Add 25 steps to your recipe.	The 25 new steps are applied to the random sample in the browser's memory.
5. Select one of the first 3 steps of your recipe.	The initial sample is loaded and displayed.
6. Insert a new step below the current one.	Now, the first 4 steps are displayed using the initial sample.

Implications:

- As you add steps to your recipe without resampling, your recipe and sample consume more memory in your browser.
- When you perform complex multi-dataset operations, such as joins or unions, your recipe/sample combination consumes a lot more memory.
- If you continue adding steps:
 - Performance in the browser can be impacted. Basic operations such as selection of data or new recipe steps can become slow to respond.
 - The browser can crash.

Sampling Considerations

Tip: When resources permit, it's a good habit to take a new sample after a few multi-dataset operations or operations that otherwise change the number of rows in your dataset have been added to your recipe.

Other considerations:

- **Generating samples takes time.** This is particularly true for Full Scan samples.
- **Sampling can cost money.** In some cloud-based environments, generating a sample costs compute resources, which can add to your computing bill.
- **You may need multiple samples.** For long or complex recipes, you may need to take multiple samples.
- **Reference datasets should begin with a sample.** When you create a recipe for a reference dataset, you should start by generating a new sample for it.

Invalid samples

Samples can become invalid. If your recipe steps change the number of rows or otherwise reshape your dataset using transformations such as pivot or join in the steps leading up to where you took the current sample, your existing sample may no longer be valid.

When the application determines that a sample is invalid:

- The sample can no longer be used. It is now listed under the Unavailable tab in the Samples panel.
- The application automatically reverts to the last known good sample.

NOTE: Depending on when the last known good sample was generated, this reversion could suddenly force a large number of steps to be processed in the browser's memory.

- You should consider generating a new sample immediately.

Best Practices

For more information on best practices, see <https://community.trifacta.com/s/article/Best-Practices-Managing-Samples-in-Complex-Flows>.

Running Job Basics

Contents:

- *Configure Job*
 - *Run Job*
 - *Iterate*
-

Configure Job

When you are ready to test your recipe against the entire dataset, click **Run** in the Transformer page. In the Run Job page, you specify the output formats and any compression to apply. Unless you are working with a large dataset, compression is unneeded for this basic walkthrough.

Tip: Optionally, you can disable generating a visual profile of your results. While the visual profile is very useful for examining issues in your recipe and iterating, it is a resource-intensive process. If you are working with large datasets that do not require additional debugging, you can consider disabling the profiling of your results. For more information, see *Overview of Visual Profiling*.

Tip: Depending on your product configuration, you may have multiple running environments available to you. In most cases, you should choose to use the default running environment, which is selected for you based on the size of the dataset.

For more information on the job execution options, see *Run Job Page*.

Run Job

To queue the specified job for execution, click **Run**.

The job is queued up for processing.

You can track progress in the Job Details page.

- If visual profiling was enabled for the job, click the Profile tab.
- When the job is completed, you can access results in the Output Destinations tab.
- For more information, see *Job Details Page*.

Iterate

In the Profile tab of the Job Details page, you can review the effects of the transformation recipe across the entire dataset. Statistics and data histograms provide overall visibility into the quality of your transformation recipe.

All data



Results profile by column

#	FMID	RBC	MarketName	Primary_Website_or_URL	RBC	Address	RBC	Location_Description	Seasons
Valid	38	Valid	38	Valid	22	Valid	38	Valid	38
Mismatched	0	Mismatched	0	Mismatched	4	Mismatched	0	Mismatched	0
Empty	0	Empty	0	Empty	12	Empty	0	Empty	0
Top 20 values		Top 14 values		Top 20 values		Top 7 values		Top 20 values	
58th and Chester Farmer...		http://www.growncyc.org		587 Harrison Street,Kal...		Other		{ "Date": "", "Time": "" }	
57th Street Greenmarket		http://www.foodtrustmar...		3rd St N (Hwy 11) besid...		Private business parki...		{ "Date": "", "Time": "Tu...	
52nd and Haverford Farm...		http://www.sandlercente...		3rd & Curry Street,Cars...		Local government build...		{ "Date": "06/17/2013 t...	
3rd Street N (hwy 11) b...		http://www.peoplesfoodc...		3808 North Meridian Str...		Faith-based instituti...		{ "Date": "06/06/2013 t...	
3rd & Curry St. Farmers...		http://www.pcfma.com/ma...		362 Plymouth Street,Abi...		Closed-off public stre...		{ "Date": "06/06/2013 t...	
38th & Meridian Farmers...		http://www.iatp.org/min...		29th and Wharton Street...		On a farm from: a bar...		{ "Date": "06/06/2013 t...	
33rd and Diamond Farmer...		http://www.foodtrustmar...		27 W Potomac Street,Bru...		Co-located with wholes...		{ "Date": "06/06/2013 t...	
32nd Street/Waverly Far...		http://www.experimental...		2349 S Hwy 127,Russell...				{ "Date": "05/11/2013 t...	
3 French Hens French Co...		http://www.carsonfarmer...		22nd and Tasker Streets...				{ "Date": "05/11/2013 t...	
29th and Wharton Farmer...		http://www.abquptowngro...		201 Market Street,Virgi...				{ "Date": "05/04/2013 t...	
25th Avenue Farmers' Ma...		http://www.abingtonsage...		1st Ave - E 92nd & 93 S...				{ "Date": "05/01/2013 t...	
22nd and Tasker Farmers...		http://www.abingdonfarm...		194 W 25th Avenue,San M...				{ "Date": "04/20/2013 t...	
2012 Wood County Farmer...		http://www.6701burnetro...		1622 6th St NE,Minneapo...				{ "Date": "04/20/2013 t...	
17th Ave Market		http://www.3frenchhensm...		1400 U Street NW,Washin...				{ "Date": "01/01/2013 t...	
175th Street Greenmarket				12th & Brandywine Stree...				{ "Date": "01/01/2013 t...	

Figure: Visual Profile

See Job Details Page.

Use the links in the Job Details page to resume working on your dataset sample and the related recipe, generating jobs when you think you are done, until you have generated the appropriate dataset.

Export Basics

After you have iterated on your recipe and generated a result that is to your satisfaction, you can export the transformed data.

Steps:

1. In the left nav bar, click the Jobs icon.
2. In the Jobs page, click the job identifier to open the job in the Job Details page.
3. Click the Output Destinations tab.

Export by:

- **Direct file download:** Click the file to download. From its righthand context menu, select **Download result**.

NOTE: Some file types cannot be downloaded.

- **Create new dataset:** You can create a new dataset from a generated output. Click the file. From its righthand context menu, select **Create imported dataset**.
- **Publish:** If Trifacta® has been integrated with an external datastore, you can publish your results to a designated target. Click **Publish**.

Common Tasks

Contents:

- *Import Tasks*
 - *Discovery Tasks*
 - *Validation Tasks*
 - *Structuring Tasks*
 - *Cleanse Tasks*
 - *Enrichment Tasks*
 - *Publishing Tasks*
 - *Project Management Tasks*
 - *Operationalization Tasks*
 - *Account Management Tasks*
-

This section contains documentation on common methods for performing your data wrangling tasks in Trifacta®.

Import Tasks

- *Connect to Data*
 - *Share a Connection*
- *Import a File*
 - *Change File Encoding*
 - *Remove Initial Structure*
- *Import a Table*
 - *Disable Type Inference*
- *Import from Another Flow*
- *Import Excel Data*
- *Import Google Sheets Data*
- *Import PDF Data*
- *Create Dataset with Parameters*
 - *Parameterize Files for Import*
 - *Parameterize Tables for Import*
- *Create Dataset with SQL*

Discovery Tasks

- *Explore Suggestions*
- *Add or Edit Recipe Steps*
- *Filter Data*
- *Locate Outliers*
- *Compute Counts*
- *Calculate Metrics across Columns*
- *Compare Strings*
- *Analyze across Multiple Columns*
- *Parse Fixed-Width File and Infer Columns*
- *Generate a Sample*
 - *Change Recipe Sample Size*

Validation Tasks

- *Profile Your Source Data*
- *Validate Your Data*
 - *Validate Column Values against a Dataset*

- *Find Bad Data*
- *Find Missing Data*
- *Manage Null Values*

Structuring Tasks

- *Initial Parsing Steps*
- *Reshaping Steps*
- *Split Column*
- *Move Columns*
- *Delete Data*
- *Select*
- *Create Aggregations*
- *Nest Your Data*
- *Unnest Your Data*
- *Pivot Data*
- *Unpivot Columns*
- *Window Transformations*
- *Working with Arrays*
- *Working with Objects*
- *Working with JSON v2*
- *Working with JSON v1*

Cleanse Tasks

- *Rename Columns*
- *Sanitize Column Names*
- *Change Column Data Type*
- *Copy and Paste Columns*
- *Create Column by Example*
- *Remove Data*
- *Deduplicate Data*
- *Compare Values*
- *Replace Cell Values*
- *Replace Values Using Patterns*
- *Replace Groups of Values*
- *Normalize Numeric Values*
- *Standardize Using Patterns*
- *Modify String Values*
- *Manage String Lengths*
- *Extract Values*
- *Format Dates*
- *Apply Conditional Transformations*
- *Prepare Data for Machine Processing*

Enrichment Tasks

- *Create New Column*
- *Add Two Columns*
- *Generate Primary Keys*
- *Add Lookup Data*
- *Append Datasets*
- *Join Data*
 - *Configure Range Join*
- *Insert Metadata*
- *Invoke External Function*

Publishing Tasks

- *Create Outputs*
 - *Create Output SQL Scripts*
- *Publish Results on Demand*
- *Reuse Recipe*

Project Management Tasks

- *Take a Snapshot*
- *Track Data Changes*
- *Add Comments to Your Recipe*
- *Create Target*
- *Optimize Job Processing*
- *Diagnose Failed Jobs*
- *Schedule a Job*
- *Create Branching Outputs*
- *Build Sequence of Datasets*
- *Fix Dependency Issues*
- *Share a Flow*
- *Export Flow*
- *Import Flow*
 - *Reconnect Flow to Source Data*
 - *Reconnect Flow to Outputs*
 - *Define Import Mapping Rules*
- *Create or Replace Macro*
- *Apply a Macro*
- *Export Macro*
- *Import Macro*
- *Create Flow Parameter*
- *Flag for Review*
- *Manage Environment Parameters*

Operationalization Tasks

- *Create Flow Webhook Task*
- *Create a Plan*
 - *Create Delete Task*
 - *Create HTTP Task*
 - *Create Slack Task*
- *Share a Plan*
- *Export Plan*
- *Import Plan*

Account Management Tasks

- *Change Password*
- *Configure Your Access to S3*

Import Tasks

These workflows pertain to creating imported datasets for use in the product.

An **imported dataset** is a reference to a source of data. It is not a copy of the data.

NOTE: Trifacta® never modifies source data.

Connect to Data

Contents:

- *Locate Connections*
 - *Use Connections*
 - *Read-Only*
 - *Write*
 - *Create Connection*
 - *Delete Connection*
-

When you import data into Trifacta®, you are creating a reference to a source of data; the source is never touched. When the data is required for use, Trifacta reads a sample of the source data into the application for your use. Data is read into the application through an object called a **connection**.

The following are the supported types of connection for the product:

- **Upload/Download:** You can upload data directly from your local desktop. You can also save it locally on export.
- **Base storage layer:** Your deployed instance of the product is connected to a base storage layer, where you can read sources and write your results.
- **Relational sources:** You can read from database tables into the product.

Locate Connections

You already have a set of connections that you can use. Connections can be either read-only or read-write.

1. In the Home page, click the Connections icon in the left nav bar.
2. The currently available Connections is displayed.

In the Import Data page, your list of available connections is displayed in the left nav bar.

Use Connections

Read-Only

1. In the Import Data page, select one of the available connections.
2. Navigate through the connection to select the asset to import.
3. Select the object and click **Open**.
4. In the Import Data page, review the settings of the asset in the card in the right panel. Make updates as needed.

Write

You write results through a connection by specifying a set of settings.

1. In the Run Job page, click **Add Publishing Action**.
2. In the left nav bar, select the connection.
3. Specify the settings for the publishing action.
4. Run the job.
5. When it successfully completes, the specified results are published through the selected connection.

Create Connection

NOTE: Some connections require additional configuration outside of the application.

When a new connection is created, it is initially available only to you.

Prerequisites:

Before you create a new connection, please verify the following:

- On the datastore, you have read and (optionally) write locations.
- You have credentials to use to connect to this datastore. These credentials have permissions on your read /write locations.
- Some datastores require a special connection string, which must be inserted as part of the connection object.

Read-only:

1. In the Import Data page, click the New icon in the left nav bar.
2. In the Create Connection window, specify the parameters of the connection.

Read-write:

1. In the Connections page, click Create Connection.
2. Click the connection category or search for a specific connection to create.
3. If a connection is grayed out:
 - a. It may already exist. Some connections types permit only one globally available connection.
 - b. It may not be supported in your product.
 - c. It may be read-only.
4. Click the name of the connection.
5. In the Create Connection window, specify the parameters of the connection.

Delete Connection

NOTE: You can delete a connection only if you are an admin or the connection owner, and the connection is not used to import any current datasets.

Steps:

1. In the Connections page, locate the connection to remove.
2. In the context menu, select **Delete....**
3. The connection is deleted.

Share a Connection

Contents:

- *Share a Connection*
- *Make a Connection Public*
- *Remove Sharing From a Connection*

This section provides an overview of sharing connections with other users for collaboration.

You can share connections with other users to use the same connection through the Connections page.

NOTE: Access to the Connections page in the application and privileges on connections is governed by roles in your workspace. For more information, please contact your workspace administrator.

Share a Connection

Steps:

1. From the Connections page, locate the connection to share.
2. From the context menu, select **Share**.
3. In the Share dialog, enter the name or email address of the user with whom you would like to share the connection.
4. Specify the privilege level of the user to whom you are sharing. For more information on sharing privileges, see *Overview of Sharing*.
5. As the owner of a connection, you can specify whether to share your credentials with other users who have access to the connection:

NOTE: Connections that use OAuth 2.0 authentication cannot be shared with credentials.

- a. **Share credentials:** (default) The credentials specified in the connection definition are shared to each user of the connection.
 - b. **Do not share credentials:** The connection credentials are not shared. Each user who is shared the connection must specify their own credentials.
6. Click **Share**.
 7. The selected users can now see the connections in the Shared with Me tab of the Connections page and can use the connection.

Make a Connection Public

Only an administrator can make a connection public.

Remove Sharing From a Connection

You can remove the sharing from a connection by performing the following steps:

1. From the Share dialog for connections, select the user to remove sharing.
2. From the drop-down next to the user, Select **Remove**.
3. The sharing for the connection is removed.

Import a File

You can import one or more files into the Library or immediately add them to a flow.

NOTE: When you import a file, the data is not stored in Trifacta®. What you create is an **imported dataset**, which is simply a reference to the source of the data. Trifacta never stores or modifies source data.

Steps:

- From the menubar, click **Library**.
- In the Library page, click **Import Data**.
- From the left sidebar in the Import Data page, select the connection where your data is located.
 - You must have read permissions on any directory and file that you wish to import.
 - **Upload:** Navigate your local desktop to select the file or files that you wish to upload.

Tip: You can select multiple files in the same directory for uploading at the same time.

- **File-based datastore:** If you are uploading from a file-based backend datastore, navigate the available directories to locate your file.
 - **Microsoft Excel:** If you are importing an Excel file that contains multiple worksheets, you must select the worksheets to include as part of your import.
 - **Dataset with Parameters:** If you are importing multiple files with similar filenames, you can import them as part of the same dataset using parameters or variables. In this manner, you create a single imported dataset, which automatically includes any new files that appear in the directory and that follow the same file naming pattern.
- Some aspects of the import process can be modified. In the right panel, click **Edit Settings** for a file that you have imported.
 - By default, the application applies a few steps to file-based imported datasets to attempt to organize them into tabular format and hides these steps from your recipe. As needed, you can disable these automated steps, so that the steps themselves appear in the Recipe panel.
 - If your file uses a different file encoding than the default encoding, you can change it for the file during the import process.
- When you are ready to complete the import process:

Tip: If present, you can click the **Add to new flow** checkbox, which adds the imported datasets to an Untitled flow.

- Your files are available as imported datasets.

For more information, see *Import Data Page*.

Change File Encoding

Files are imported based on the default file encoding for Trifacta®.

The default file encoding can be configured. For more information, see *Configure Global File Encoding Type*.

As needed, you can override the default file encoding during the importing of individual datasets.

NOTE: All output files are written in UTF-8 encoding.

Tip: If you have already imported the dataset and need to change this setting, you can re-import the source and change the settings. In any flows that use the previously imported version of this dataset, you can change the input for any recipe that uses the old version to use this newly imported version in Flow View.

Steps:

1. After you have selected or specified the file to import in the Import Data page, click **Edit Settings** for the dataset card in the right panel.
2. From the drop-down, select the preferred encoding to apply to this specific file.
3. Continue the import process.

Remove Initial Structure

When you import a dataset from a file, Trifacta® attempts to detect the structure of the file and to apply an initial set of parsing steps to the data to render it in tabular form for display in the Transformer page. For example, JSON files may be turned into a table of data as long as the structure of the data supports this structuring.

NOTE: Initial parsing steps are applied only to file-based sources of data.

These steps vary based on the file format of data that is being imported. Depending on the dataset, you may need to modify these steps or rebuild them altogether. You can use the following steps to prevent Trifacta® from detecting the structure and automatically hiding these steps.

Tip: You should allow the product to detect the structure first. If it does not detect the structure well, you can experiment with disabling it and rebuilding the steps to meet your dataset requirements.

Tip: If you have already imported the dataset and need to change this setting, you can re-import the source and change the settings. In any flows that use the previously imported version of this dataset, you can change the input for any recipe that uses the old version to use this newly imported version in Flow View.

NOTE: When the steps are completed, the initial parsing steps are listed in any recipe that you create from the imported dataset. If you wish to remove them altogether, you can delete them from the recipe.

Steps:

1. After you have selected or specified the file to import in the Import Data page, click **Edit Settings** for the dataset card in the right panel.
2. Deselect the Detect Structure checkbox. For more information, see *File Import Settings*.
3. Continue the import process by adding the dataset to a new flow.
4. When the imported dataset is added to a flow, it is listed as an **unstructured dataset**.
5. Select the dataset and click **Add new recipe**.
6. When you select the recipe, the initial parsing steps are listed in the right panel.
7. When the dataset is loaded into the Transformer page, you can modify these steps to improve the parsing or delete them altogether.

NOTE: Any step that breaks up the data into individual rows into individual rows must be the first step in the recipe. To create, enter **Break into rows** in the Search panel.

Import a Table

You can import one or more tables into the Library or immediately add them to a new or existing flow.

NOTE: When you import a file, the data is not stored in Trifacta®. What you create is an **imported dataset**, which is simply a reference to the source of the data. Trifacta never stores or modifies source data.

Steps:

- From the menubar, click **Library**.
- In the Library page, click **Import Data**.
- From the left sidebar in the Import Data page, select the connection to the relational datastore where your data is located.
- Browse the relational datastore to locate the table that you wish to import.
- You must have read permissions on any database and table that you wish to import.
- **Create Dataset with SQL:** You can apply a custom SQL statement to import from a database. For more information, see *Create Dataset with SQL*.
- Some aspects of the import process can be modified. In the right panel, click **Edit Settings** for a table that you have imported.
 - The application's data types are applied to the table's columns during the import process. If needed, you can disable type inference, so that the data types of the original source are preserved, if possible, during import. For more information, see *Disable Type Inference*.
- When you are ready to complete the import process:

Tip: If present, you can click the **Add to new flow** checkbox, which adds the imported datasets to an Untitled flow in Flow View.

- Your tables are available as imported datasets.

For more information, see *Import Data Page*.

Disable Type Inference

When Trifacta® creates an imported dataset from a schematized source, the product applies its own type inferencing to the columns of the imported data. Type inferencing may be reapplied during some operations, such as the creation of samples or when data reshaping transformations are applied in the Transformer page.

If preferred, you can disable this type inferencing on the columns of your imported dataset. When the data is imported, the original types from the source system remain. Any types that do not have a corresponding match with the Trifacta data types must be manually typed in the application.

Methods of disabling:

Column data typing is applied to schematized sources in one of three ways:

1. Globally
2. Per-connection type inference settings override the global setting.
3. Per-file type inference settings override both global and per-connection settings.

For more information on applying global or per-connection type inference settings, see *Configure Type Inference*.

You can use the following steps to disable type inference applied to a specific file during the import process.

Tip: For imported datasets from relational sources, you can identify in Flow View whether type inferencing has been applied to the dataset. When the dataset is selected in Flow View, locate the Type Inference entry in the right panel.

Tip: If you have already imported the dataset and need to change this setting, you can re-import the source and change the settings. In any flows that use the previously imported version of this dataset, you can change the input for any recipe that uses the old version to use this newly imported version through Flow View.

Steps:

1. After you have selected or specified the relational table to import in the Import Data page, click **Edit Settings** for the dataset card in the right panel.
2. Deselect the Infer column data types checkbox.
3. Continue the import process.
4. When the dataset is loaded into the Transformer page, no new data typing is applied at all, unless you manually specify the Trifacta data types for the column.

Import from Another Flow

Contents:

- *Import Imported Dataset*
 - *Import Reference Dataset*
 - *Import Snapshot of Flow Output*
-

You can use one of the following methods to import data from another flow into your current flow.

NOTE: When you import a file or a reference, the data is not stored in Trifacta®.

Import Imported Dataset

If another flow contains an imported dataset that you want to use, you can import it into your current flow.

NOTE: To use an imported dataset from another flow, you must have access to the dataset itself. If you are not the owner of the flow, it must be shared with you. If the connection used to import the dataset is not shared with you, you may have to build your own connection to the source.

Steps:

1. Open the target flow.
2. In Flow View, select **Add Datasets**.
3. In the Add Datasets to Flow dialog, click the Imported tab.
4. Browse the available datasets:
 - a. Select the one to import.
 - b. If you do not see it, click **Import datasets**. Navigate and select the dataset to import.
5. The dataset is imported into the flow.

Import Reference Dataset

For any flow, you can create a reference to a recipe in it. This **reference** enables the output of the recipe, after execution, to be used elsewhere. When you import this reference into another flow, you create a **reference dataset**.

NOTE: A reference dataset is a dynamic object. If the recipe that is the source of the reference changes, then the reference dataset may change without warning. In the flow that uses the reference dataset, you may see unexpected errors in your recipe. For more information, see *Fix Dependency Issues*.

Steps:

1. In the source flow in Flow View, locate the recipe whose output you wish to use in another recipe.
2. Right-click and select **Add > Reference**.

3. The reference is created:



Figure: Reference object

4. In the right panel, click **Add to Flow....**
5. Select the flow to which to add the reference, or create a new one.
6. The reference is used to create the reference dataset in the target flow.



Figure: Reference dataset in a new flow

For more information, see [View for Reference Datasets](#).

Import Snapshot of Flow Output

If you need a snapshot of data at a point in time from another flow, you can do either of the following.

NOTE: Since you are generating an output file in both of the following cases, the imported dataset that you create from these outputs does not receive updated data.

1. **Snapshot of recipe in development:**
 - a. In the source flow, select a specific step in your recipe in the Recipe panel.
 - b. From the panel context menu, select **Download Sample as CSV**.
 - c. The recipe steps up to the selected step are performed on the current sample, and the current state of the sample is download in CSV format to your local desktop.
 - d. Through the Import Data page, you can import this generated file.
 - e. For more information, see *Take a Snapshot*.
2. **Snapshot of job results:**
 - a. In the source flow, select your recipe in the Recipe panel.
 - b. Select the output object icon above the recipe.
 - c. In the side panel, click **Run**. Specify the job outputs. For best results, select a CSV or JSON output in the Run Job page.
 - d. When the job completes, click the job identifier. The Job Details page opens.
 - e. In the Job Details page, click the Output Destinations tab. For the generated output, select **Create imported dataset** from its context menu.
 - f. A new imported dataset is created in your Library.
 - g. In the target flow, add this dataset to your flow.
 - h. For more information, see *Build Sequence of Datasets*.

Import Excel Data

In addition to CSV and other formats, Trifacta® can directly import Microsoft® Excel® workbooks and folders containing workbooks.

The worksheets of a workbook can be imported as:

- Individual datasets
- A single dataset
- A dataset with parameters

NOTE: When importing one or more Excel files as a parameterized dataset, you select worksheets to include from the first file. If there are worksheets in other Excel files that match the names of the worksheets that you selected, those worksheets are also imported. All worksheets are unioned together into a single imported dataset with parameters. Pattern-based parameters are not supported for import of Excel worksheets.

Limitations

- XLSX and XLS format are supported. Other Excel-related formats, such as XLSM format, are not supported.
- Some characters, such as hashtags (#) and curly braces ({}) cannot be used in filenames. For more information, see *Supported File Formats*.
- Filepath and source row number information is not available from original Excel files. These references return values from the CSV files that have been converted on the backend. For more information, see *Source Metadata References*.
- Source Excel files with cells bracketed by single double quotes may not be properly ingested if any terminating quotes are missing.

Tip: You can check the data quality bars for mismatched values or, for strings, the data histogram bars for anomalous values to see if the above issue is present. If so, deselect Detect Structure on import. Then, use a Split rows transformation applied to the affected column to break up the column as needed.

- Macros in your Excel files are not imported.
 - During import, cell formulas are applied, and the output values are used in the imported dataset.
- You cannot import password-protected Excel files.
- Import of Excel files with protected columns or cells is not supported.
- Compressed Excel files are not supported.
- Conversion of large Excel files require non-linear increases in memory requirements on the Trifacta node.
- If loading your Excel-based dataset in the Transformer page results in a blank screen, please take a new sample. The file requires conversion again with each generated sampling.

NOTE: When you share a flow that contains a dataset sourced from Microsoft Excel, the user with whom the flow is shared may receive a `Could not parse` error. In this case, the user does not have access to the original sample. The workaround is to take a new sample or to run a job on the full dataset.

- Latest state of the Excel file may not be reflected in the Transformer page due to caching. When you run a job, the platform always collects the latest version of the data and converts it to CSV for execution.

Use

When Excel data is imported into Trifacta, each sheet in an imported file must be converted to a CSV and then ingested for use.

Steps:

1. In the menu bar, click **Library**.
2. In the Library page, click **Import Data**. Select the connection to use.

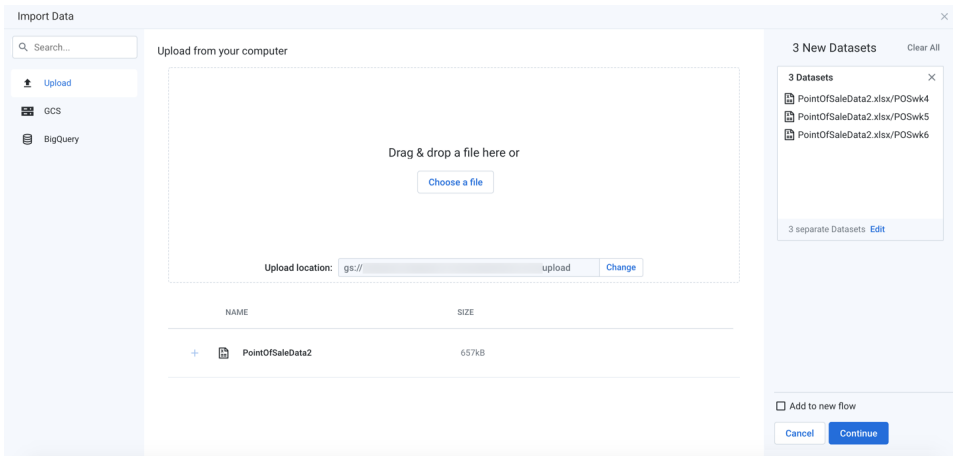


Figure: Import Excel workbook

Tip: If you experience issues uploading large XLS/XLSX files, you can convert the files to CSV files and then upload them.

3. After you select the workbook, it is uploaded and converted to CSV format and stored by the platform. Depending on the size of the workbook, this process may take a while.
4. By default, all worksheets in the workbook are imported as individual datasets. To change how the data is imported, click **Edit** in the right panel.

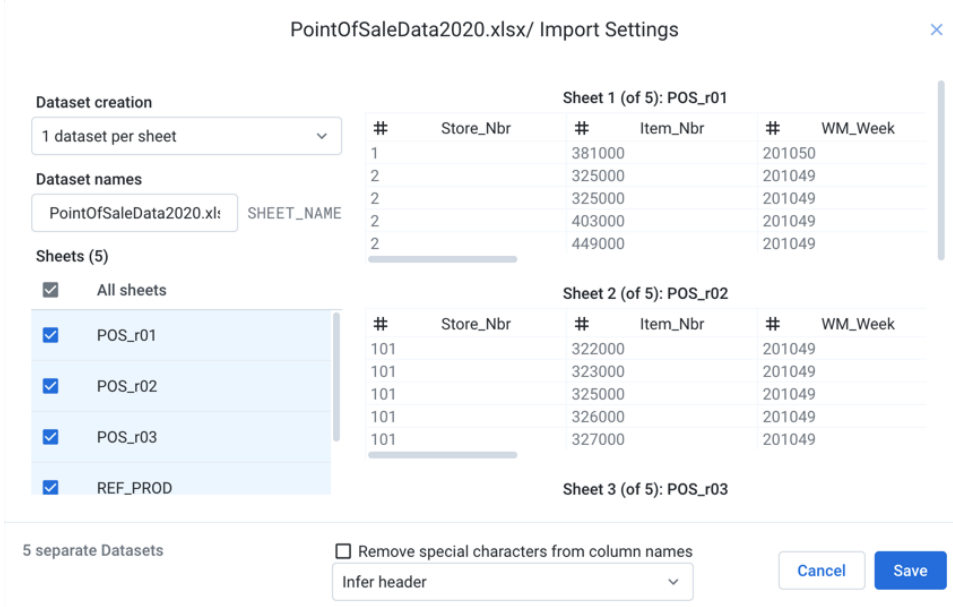


Figure: Import settings for Excel datasets

5. Dataset creation:

- a. **1 dataset per sheet:** (Default) Each selected sheet in the workbook is imported as a separate dataset.
Specify the base name of the datasets that you are creating. If you are creating a single dataset, the name of the workbook is used.
- b. **Selected sheets into 1 dataset:** All selected sheets in the workbook are combined and imported as a single dataset.

NOTE: The schemas of each dataset must match. Columns must be listed in the same order in each dataset. The column headers are taken from the first selected dataset.

- c. **All and future sheets into 1 dataset:** If the workbook is updated periodically with new sheets that you would like to add in the future, select this option. After initial selection of sheets, all sheets that are added to the workbook in the future are automatically added as part of the imported dataset.

Tip: Use this option to capture future additional sheets or changes to the names of the current sheets.

NOTE: When an imported dataset based on this option is first loaded into the Transformer page, the data grid displays an initial sample taken from rows in the first sheet only. When you take another sample from the Samples panel, data is collected from other sheets.

NOTE: This option is available only if you are connected to a backend file storage system.

6. Selected sheets:

- a. You can select the sheets to import.

NOTE: If you are importing a folder of Excel files, data preview and initial sampling are executed against the first file found in the folder.

- b. To preview the data of an individual sheet, mouse over a dataset and click **Jump to**.

7. Remove special characters from column names: Select this option to remove any special characters from the inferred column headers during import.
8. From the drop-down, you can specify how you want the application to parse the data for column headers.
9. To save changes, click **Save**.
10. After your datasets have been added, you can edit the name and description information for each in the right navigation panel.
11. Optionally, you can assign the new dataset(s) to an existing flow or create a new one to contain them.

For more information, see *Import Data Page*.

Import Google Sheets Data

Trifacta® can import Google® Sheets® spreadsheets.

The sheets of a spreadsheets can be imported as:

- Individual datasets
- A single dataset

Limitations:

NOTE: This integration provides access to all Google Sheets in the connecting user's account. Access includes spreadsheets with disabled options for download, print, or copy as well as hidden sheets within spreadsheets.

- Import-only support

NOTE: After you import a Google Sheet into Trifacta, renaming the source Google Sheet or a tab in it can break your datasets and flows. Details are below.

- Creation of a dataset with parameters from Google Sheets is not supported.
- Connected sheets or embedded external datasources in your Google Sheets are not supported.

Tip: If your connected sheet is linked to a table-based source, you may import that source directly into the product.

- If you have enabled Google Advanced Protection, this connection type does not work.
- A Google Sheet can contain up to 5,000,000 cells. Each cell can contain up to 50,000 characters.
 - Trifacta supports a maximum of 25,000 characters in a cell.
- Filepath and source row number information is not available from original Sheets. These references return values from the CSV files that have been converted on the backend. For more information, see *Source Metadata References*.
- Source Sheets files with cells bracketed by single double quotes may not be properly ingested if any terminating quotes are missing.

Tip: You can check the data quality bars for mismatched values or, for strings, the data histogram bars for anomalous values to see if the above issue is present. If so, deselect Detect Structure on import. Then, use a Split rows transformation applied to the affected column to break up the column as needed.

- If loading your Sheets-based dataset in the Transformer page results in a blank screen, please take a new sample. The file requires conversion again with each generated sampling.
- Latest state of the spreadsheet may not be reflected in the Transformer page due to caching. When you run a job, the platform collects the latest version of the data and converts it to CSV for execution.
- IMPORTRANGE function in Google Sheets is not supported for importing data from another sheet.

Process:

1. A spreadsheet can be read directly from your Google Drive.

NOTE: When you first use the Google Sheets connector, you must enable Trifacta to read all of your Google Drive data. When the connector is used, it locates only the Google Sheets data, including any Sheets that have been shared with you. All other data in Google Drive, including any Microsoft® Workbooks®, is ignored. You can then select the Sheet or Sheets you wish to import.

2. Sheets in a worksheet are ingested and written to Base Storage in CSV format.
3. CSV files are available for selection.
4. These CSV files are the source from which the imported datasets are created.

Steps:

1. In the menu bar, click **Library**.
2. In the Library page, click **Import Data**. Select the Google Sheets connection.

Tip: You can paste links that you gather from Google to select spreadsheets. To access a Google Sheet, edit the path and paste the link. Use this method for publicly available Google Sheets, too.

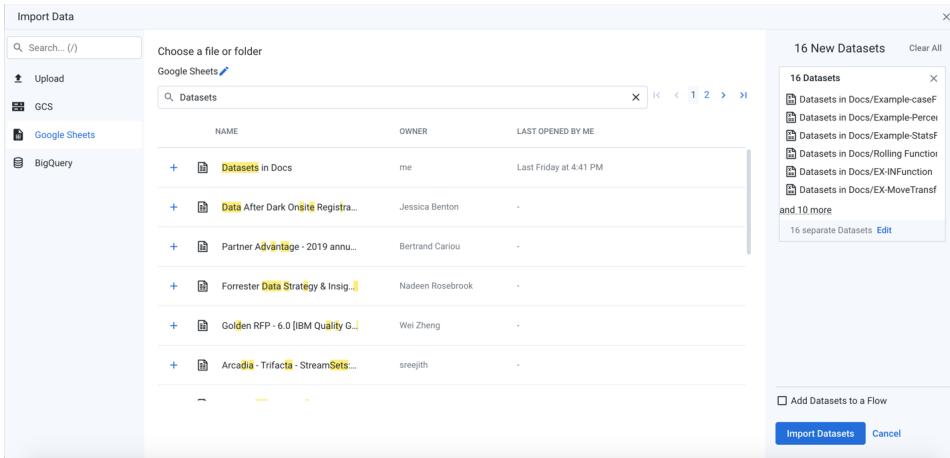


Figure: Import Google Sheets spreadsheet

3. After you select the spreadsheet, it is uploaded and converted to CSV format and stored. Depending on the size of the spreadsheet, this process may take a while.
4. By default, all sheets in the spreadsheet are imported as individual datasets. To change how the data is imported, click **Edit** in the right panel.

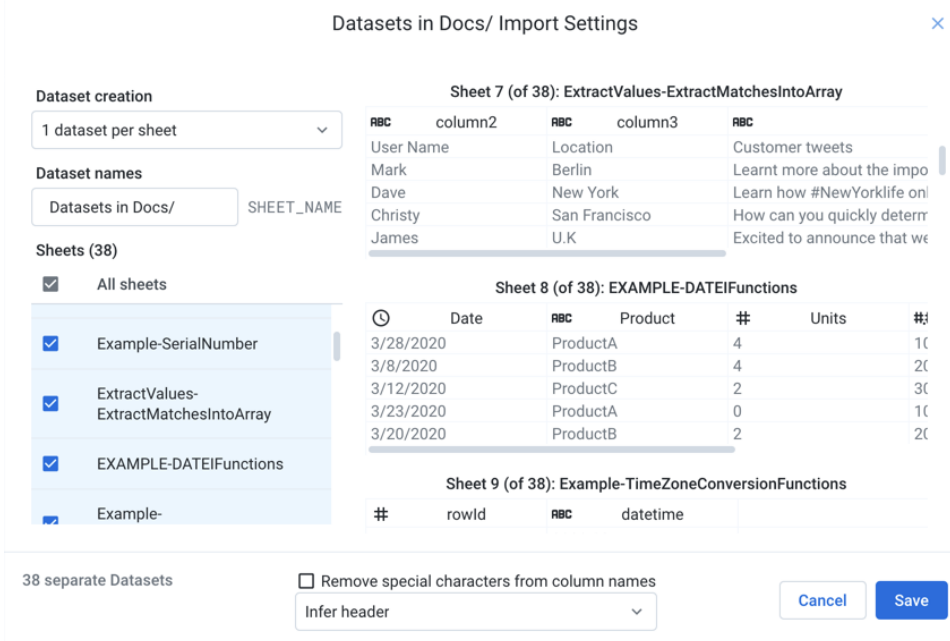


Figure: Import settings for Google Sheets datasets

5. Dataset creation:

- a. **1 dataset per sheet:** (Default) Each selected sheet in the spreadsheet is imported as a separate dataset.
Specify the base name of the datasets that you are creating. If you are creating a single dataset, the name of the spreadsheet is used.
- b. **Selected sheets into 1 dataset:** All selected sheets in the spreadsheet are combined and imported as a single dataset.

NOTE: The schemas of each dataset must match. Columns must be listed in the same order in each dataset. The column headers are taken from the first selected dataset.

- c. **All and future sheets into 1 dataset:** If the spreadsheet is updated periodically with new sheets that you would like to add in the future, select this option. After initial selection of sheets, all sheets that are added to the spreadsheet in the future are automatically added as part of the imported dataset.

NOTE: When an imported dataset based on this option is first loaded into the Transformer page, the data grid displays an initial sample taken from rows in the first sheet only. When you take another sample from the Samples panel, data is collected from other sheets.

6. Selected sheets:

- a. You can select the sheets to import.

NOTE: Special characters in sheet names are filtered out.

- b. To preview the data of an individual sheet, mouse over a dataset and click **Jump to**.
7. Remove special characters from column names: Select this option to remove any special characters from the inferred column headers during import.
8. You can apply the column headers to your datasets during import. Select the required option from the drop-down list:
 - **Infer header:** (default) When selected, the Trifacta application infers the header based on the data in the import.
 - **Use first row as header:** When selected, the first row is used as the column headers.
 - **No header:** When selected, the inference is ignored and column headers are defined using generic names with no headers.
9. To save changes, click **Save**.
10. After your datasets have been added, you can edit the name and description information for each in the right navigation panel.
11. Optionally, you can assign the new dataset(s) to an existing flow or create a new one to contain them.

For more information, see *Import Data Page*.

After import:

After you have imported the Google Sheet, you should avoid renaming the Google Sheet or any tab in it that is part of the imported datasets. If you rename a datasource, you can see one or more of the following issues in Trifacta:

- When you open a recipe using the dataset in the Transformer page, you may receive an error that the Base Storage path cannot be loaded.
- Collecting samples in the Transformer page returns a generic error message.

Import PDF Data

Contents:

- *Limitations*
- *Enable*
- *Table Import*
- *Import Steps*

NOTE: This feature is in Beta release.

Trifacta® can directly import Adobe® Acrobat® PDF files containing one or more tables.

The tables of a PDF can be imported as:

- Individual datasets
- A single dataset
- A dataset with parameters

NOTE: When importing as a parameterized dataset, all selected tables are imported into a single dataset.

PDF files can be uploaded from your local system.

Limitations

- PDF ingest is limited to 100 MB per file.
- Filepath and source row number information is not available from original PDF files. These references return values from the CSV files that have been converted on the backend. For more information, see *Source Metadata References*.
- You cannot import password-protected PDF files.
- Compressed PDF files are not supported.
- Conversion of large PDF files require non-linear increases in memory requirements on the Trifacta node.
- If loading your PDF-based dataset in the Transformer page results in a blank screen, please take a new sample. The file requires conversion again with each generated sampling.
- Latest state of the PDF file may not be reflected in the Transformer page due to caching. When you run a job, the platform always collects the latest version of the data and converts it to CSV for execution.

Enable

This feature is disabled by default. To enable, please complete the following:

Steps:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
2. Locate the following parameter and set it to `true`:

```
"feature.enablePDFSupport": false,
```

3. Add references to the PDF format to the following parameters:

```
"webapp.convertableExtensions": "xls,XLS,xlsx,XLSX,pdf,PDF",  
"webapp.client.allowedFileExtensions": "<other_options>,pdf,PDF",
```

4. Save your changes and restart the platform.

Table Import

The PDF file format is a publishing format designed around visual layout of information, some of which may include tabular data. Table data in PDF files must be detected and converted into CSV data for proper ingestion in the platform. This ingest process occurs on the backend datastore.

To facilitate ingestion, the following requirements must be met for tables in your source PDF files:

- Non-tabular data in the file is ignored.
- Tables must be enclosed in a border. Each cell in the table must be bordered.
- Tabular data in the PDF cannot be scanned data, which is stored as an image. Data must be written into the file.
- When a table spans multiple pages, it is ingested as two separate CSV files, which can be combined later.
- If a file contains multiple tables, each table is converted as a separate dataset.

Tip: After import, separate datasets can be unioned together or integrated using as a dataset with parameters.

Import Steps

1. In the menu bar, click **Library**.
2. In the Library page, click **Import Data**. Select the connection to use.

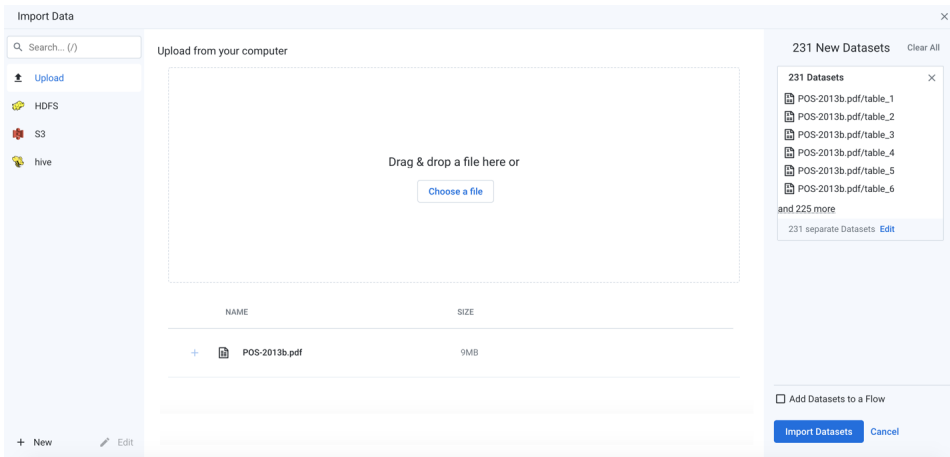


Figure: Import PDF file containing multiple pages

3. After you select the file, it is uploaded and converted to into individual CSV files for each page in the PDF file and then stored by the platform. Depending on the size of the file, this process may take a while.

4. By default, all pages in the PDF are imported as individual datasets. To change how the data is imported, click **Edit** in the right panel.

POS-2013b.pdf/ Import Settings ✕

Dataset creation

1 dataset per table ▼

Dataset names

POS-2013b.pdf/ TABLE_NAME

Tables (231)

☒ All tables

- ☒ table_1
- ☒ table_2
- ☒ table_3
- ☒ table_4

Table 1 (of 231): table_1

#	Store_Nbr	#	Item_Nbr	#	WM_Week
1		381000			201050
2		325000			201049
2		325000			201049
2		403000			201049
2		449000			201049 201049

Table 2 (of 231): table_2

#	column2	#	column3	#	column4	⌂
4		530000			201049	2/
4		543000			201049	2/
4		548000			201049	2/
4		551000			201049	2/
4		555000			201049	2/

Table 3 (of 231): table_3

231 separate Datasets

☐ Remove special characters from column names

Infer header ▼

Cancel Save

Figure: Import settings for PDF datasets

5. Dataset creation:

- a. **1 dataset per table:** (Default) Each selected table in the PDF is imported as a separate dataset. Specify the base name of the datasets that you are creating. If you are creating a single dataset, the name of the PDF file is used.
- b. **Selected tables into 1 dataset:** All selected tables in the PDF are combined and imported as a single dataset.

NOTE: The schemas of each dataset must match. Columns must be listed in the same order in each dataset. The column headers are taken from the first selected dataset.

- c. **All and future tables into 1 dataset:** If the PDF is updated periodically with new tables that you would like to add in the future, select this option. After initial selection of the tables to include, all PDF pages that are added to the PDF file in the future are automatically added as part of the imported dataset.

NOTE: This option is available only if you are connected to a backend file storage system.

NOTE: When an imported dataset based on this option is first loaded into the Transformer page, the data grid displays an initial sample taken from rows in the first table only. When you take another sample from the Samples panel, data is collected from other tables.

6. Selected tables:

- a. You can select the tables to import. A table can be a single page, or a single table among multiple on a page.

NOTE: If you are importing a folder of PDF files, data preview and initial sampling are executed against the first file found in the folder.

- b. To preview the data of an individual table, mouse over a dataset and click **Jump to**.

7. Remove special characters from column names: Select this option to remove any special characters from the inferred column headers during import.
8. You can also choose how to detect column headers from each imported table.
9. To save changes, click **Save**.
10. After your datasets have been added, you can edit the name and description information for each in the right navigation panel.
11. Optionally, you can assign the new dataset(s) to an existing flow or create a new one to contain them.

For more information, see *Import Data Page*.

Create Dataset with Parameters

Contents:

- *From File System*
 - *Parameterize bucket names*
- *From Relational Sources*
- *Edit Parameter*
- *Apply Parameter Overrides*
 - *Apply parameter overrides for your flow*
 - *Apply parameter overrides for your job*
- *Delete Parameter*

This section provides an overview on how to parameterize relational sources and files while importing data to your flow.

For more information on parameterization of datasets and other types of parameters, see *Overview of Parameterization*.

From File System

When browsing for data on your default storage layer, you can choose to parameterize elements of the path. Through the Import Data page, you can select elements of the path, apply one of the supported parameter types and then create the dataset with parameters.

NOTE: When you import a file, the data is not stored in Trifacta®. What you create is an imported dataset that is simply a reference to the source of the data. Trifacta never stores or modifies source data.

When you create a dataset with parameters in Trifacta, you can replace segments of the input path with parameters. Suppose you have the following files that you'd like to capture through a parameterized dataset:

```
//source/user/me/datasets/month01/2017-01-31-file.csv
//source/user/me/datasets/month02/2017-02-28-file.csv
//source/user/me/datasets/month03/2017-03-31-file.csv
//source/user/me/datasets/month04/2017-04-30-file.csv
//source/user/me/datasets/month05/2017-05-31-file.csv
//source/user/me/datasets/month06/2017-06-30-file.csv
//source/user/me/datasets/month07/2017-07-31-file.csv
//source/user/me/datasets/month08/2017-08-31-file.csv
//source/user/me/datasets/month09/2017-09-30-file.csv
//source/user/me/datasets/month10/2017-10-31-file.csv
//source/user/me/datasets/month11/2017-11-30-file.csv
//source/user/me/datasets/month12/2017-12-31-file.csv
```

A parameterized reference to all of these files would look something like:

```
//source/user/me/datasets/month##/YYYY-MM-DD-file.csv
```

Through the application, you can specify the parameters to match all values for:

- ## - You can use a wildcard or (better) a pattern to replace these values.
- YYYY-MM-DD - A formatted Datetime parameter can replace these values.

For more information, see *Parameterize Files for Import*.

Parameterize bucket names

You can create environment parameters for your bucket names.

From Relational Sources

You can create datasets from a relational source by applying parameters to the custom SQL that pulls the data from the source. During import of database tables through relational connections, you can apply parameters to the SQL query that you use to define the imported dataset. In some scenarios, you may need to define the table to import using a variable parameter or to parameterize the time value associated with a table name. Using parameters, you can define the tables, columns, and conditions of the query that you use to bring in data from a relational database.

For more information, see *Parameterize Tables for Import*.

Edit Parameter

After you have created your dataset with parameters, you can edit the parameter as needed.

Steps:

1. In the left nav bar, select **Library**.
2. In the Library page, locate the dataset. From its context menu, select either of the following:
 - a. **Files:** Select **Edit parameters**. In the Edit Dataset with Parameters, click the parameter to modify its definition.
 - b. **Tables:** Click **Edit Custom SQL**. In the Custom SQL window, you can modify the SQL statement, including any parameters in it. For more information, see *Create Dataset with SQL*.

Apply Parameter Overrides

After you have created a parameterized dataset, you can apply overrides to the default value. These override values can be applied in the following cases.

Case	Precedence	Scenario
Job	1	When you choose to execute a job, you can set a new value for the parameter, which is applied for the specified job only.
Flow	2	<p>If your imported dataset containing a parameter is added to a flow, you can define an override value for the dataset's parameter through Flow View.</p> <p>Whenever a job is executed on the imported dataset within the flow, the override value is applied to the dataset.</p> <div>NOTE: If a job-level override is applied on top of a flow-level override, the job override value is applied to the job.</div>
Default	3	The default value for the parameter is used if no override is applied.

Apply parameter overrides for your flow

Steps:

1. Open the flow.
2. In Flow View, select the icon for your dataset with parameters.
3. From the context menu, select **Parameter**.
4. In the Manager Parameters dialog, click the Overrides tab.
5. Edit the required values, click **Save**.

For more information, see *Manage Parameters Dialog*.

Apply parameter overrides for your job

You can apply parameter overrides to your job.

Steps:

1. In Flow View, select the output that you wish to generate.
2. In the right context panel, click **Run Job**.
3. In the Run Job page, you can specify job-level overrides at the bottom of the screen.

For more information, see *Run Job Page*.

Delete Parameter

Steps:

1. In the Edit Dataset with Parameters screen, select the parameter that you wish to remove.

NOTE: Before you remove parameter, you may want to take note of the default value, which may need to be applied to the path or query after you remove the parameter.

2. In the popup, click **Delete**.
3. Save your changes.
4. The parameter is removed from the imported dataset definition.

Parameterize Files for Import

Contents:

- *Structuring Your Data*
 - *Steps*
 - *Add Datetime Parameter*
 - *Extend Datetime parameter*
 - *Add Variable*
 - *Parameterize bucket names*
 - *Add Pattern Parameter*
-

This section describes how to create datasets and replace segments by parameterizing the input paths to your data in Trifacta .

Structuring Your Data

Each file that is included as part of the dataset with parameters should have identical structures:

- Matching file formats
- Matching column order, naming, and data type
- Matching column headers. Each column in any row that is part of a column header in a dataset with parameters should have a valid value that is consistent with corresponding values across all files in the dataset.

NOTE: If your files have missing or empty values in rows that are used as headers in your recipe, these rows may be treated as data rows during the import process, which may result in unexpected or missing column values.

- Within each column, the data format should be consistent.
 - For example, if the date formats change between files in the source system, you and your recipe may not be able to manage the differences, and it is possible that data in the output may be missing.

NOTE: Avoid creating datasets with parameters where individual files or tables have differing schemas. Either import these sources separately and then correct in the application before performing a union on the datasets, or make corrections in the source application to standardize the schemas.

When working with datasets with parameters, it may be useful to do the following if you expect the underlying datasets to be less than 100% consistent with each other.

- Recreate the dataset with parameters, except deselect the Detect Structure option during the import step.
- In the Transformer page, collect a Random Sample using a full scan. This step attempts to gather data from multiple individual files, which may illuminate problems across the data.

Tip: If you suspect that there is a problem with a specific file or rows of data (e.g. from a specific date), you can create a static dataset from the file in question.

Steps

NOTE: Matching file path patterns in a large directory can be slow. Where possible, avoid using multiple patterns to match a file pattern or scanning directories with a large number of files. To increase matching speed, avoid wildcards in top-level directories and be as specific as possible with your wildcards and patterns.

1. In the Import Data page, navigate your environment to locate one of the files or tables that you wish to parameterize.
2. Click **Create Dataset with Parameters**.

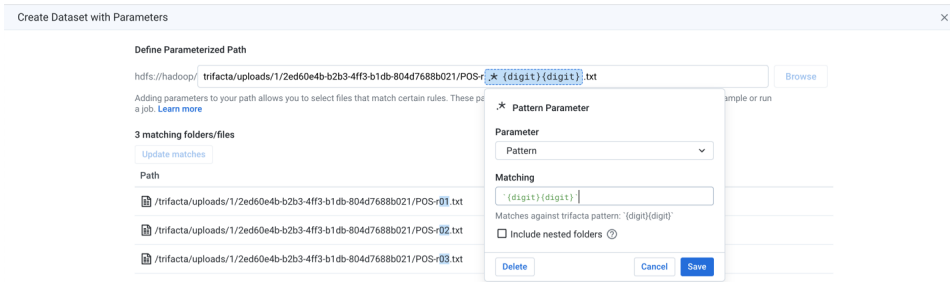


Figure: Create Dataset with Parameters

3. Within the Define Parameterized Path, select a segment of text. Then select one of the following options:

Tip: For best results when parameterizing directories in your file path, include the trailing slash (/) as part of your parameterized value.

- a. Add Datetime Parameter
 - b. Add Variable
 - c. Add Pattern Parameter - wildcards and patterns
 - d. If you need to navigate elsewhere, select **Browse**.
4. Specify the parameter. Click **Save**.
 5. Click **Update matches**. Verify that all of your preferred datasets are matching.

NOTE: If you are matching with more datasets than you wish, you should review your parameters.

6. Click **Create**.
7. The parameterized dataset is loaded.

Add Datetime Parameter

Datetime parameters require the following elements:

Format: You must specify the format of the matching date and/or time values using alphanumeric patterns. To review a list of example formats, click **Browse Date/Timestamp Patterns**.

You can also create custom formats using patterns. For example, the following regex pattern matches patterns like MM.DD.YYYY:

```
/[0-9][0-9]\.[0-9][0-9]\.[0-9][0-9][0-9][0-9]/
```

Date range: Use these controls to specify the range that matching dates must fall within.

NOTE: Date range parameters are case-insensitive.

Tip: Datetime parameters that you configure here are evaluated at the time of job execution. So, `now` refers to the time when the job is executed.

Time zone: The default time zone is the location of the host of the application. To change the current time zone, click **Change**.

For a list of supported time zone values, see *Supported Time Zone Values*.

Extend Datetime parameter

A parameterized dataset can support only one Datetime parameter. If you have multiple parts of the path that contain date information, you can create a Datetime element for each part.

Steps:

1. Within the Define Parameterized Path, select a segment of text for which to create the first part.
2. Create the Datetime parameter for this element. Remember to use the appropriate format for the part. For example, if you have highlighted a four-digit year for the part, the date format value should be: `YYYY`.
3. Then, select the second element and click the Extend Datetime Parameter icon.

The screenshot shows the 'Define Parameterized Path' interface. The path is `hdfs://trifacta/uploads/1/aa112099-4f6e-4925-93fa-d98677233965/`. Two segments are highlighted: `YYYY` and `04`. A 'Datetime Parameter' dialog is open for the `04` segment. The dialog shows the format `MM`, an example `01`, and a date range of `1 years` starting from `01/2018` to `12/2018` in the `America/Los_Angeles` time zone. The 'Save' button is highlighted.

Figure: Click the Extend Datetime Parameter icon to create additional parts to your Datetime parameter.

4. In the dialog, you can specify the date format of the second element of your Datetime parameter. Matches are made on the two elements, as well as any static text in between them.

Add Variable

A **variable** parameter is a key-value pair that can be inserted into the path.

- At execution time, the default value is applied, or you can choose to override the value.
- A variable can have an empty default value.

Name: The name of the variable is used to identify its purpose.

NOTE: If multiple datasets within the same flow share the same variable name, they are treated as the same variable.

Tip: Type `env.` to see the environment parameters that can be applied. These parameters are available for use by each user in the environment.

Default Value: If the variable value is not overridden at execution time, this value is inserted in the variable location in the path.

NOTE: When you edit an imported dataset, if a variable is renamed, a new variable is created using the new name. Any override values assigned under the old variable name for the dataset must be re-applied. Instances of the variable and override values used in other imported datasets remain unchanged.

Parameterize bucket names

You can create environment parameters to specify your bucket names. An **environment parameter** is a variable name and String value that can be referenced by all users of the environment.

NOTE: A workspace administrator or project owner can create environment parameters.

Uses:

- Parameterized bucket names are very useful when you are moving flows between workspaces or projects. When the flow is imported into a new workspace, the environment parameter references the appropriate bucket name in the new workspace.
- If you change source buckets or move data to a new storage bucket, updating the paths to your objects can be as simple as changing the value of the environment parameter where your data is stored.

For example, suppose you have two environments: Dev and Prod. You can create an environment parameter called `env.sourceBucketName` to store the name of the bucket from which all data in the workspace or project is imported.

Environment Name	Source Bucket Name	Environment Parameter Value
Dev	MyCo_Dev	<code>\$env.sourceBucketName = 'MyCo_Dev'</code>
Prod	MyCo_Prod	<code>\$env.sourceBucketName = 'MyCo_Prod'</code>

For more information, see *Environment Parameters Page*.

Add Pattern Parameter

In the screen above, you can see an example of pattern-based parameterization. In this case, you are trying to parameterize the two digits after the value: `POS-r`.

Include nested folders

When you create a wildcard or pattern-based parameter, you have the option to scan any nested folders for matching sources.

- If disabled, the scan stops when the next slash (/) in the path is encountered. Folders are not matched.
- If enabled, the scan continues to any depth of folders.

NOTE: A high number of files and folders to scan can significantly increase the time required to load your dataset with parameters.

Example 1: all text files

Suppose your file and folder structure look like the following:

```
//source/user/me/datasets/thisfile.txt
//source/user/me/datasets/thatfile.txt
//source/user/me/datasets/anotherfile.csv
//source/user/me/datasets/detail/anestedfile.txt
//source/user/me/datasets/detail/anestedfile2.txt
//source/user/me/datasets/detail/anestedfile4.txt
//source/user/me/ahigherfile.txt
```

Since the filenames vary significantly, it may be easiest to create your pattern based on a wildcard. You create a wildcard parameter on the first file in the `//source/user/me/datasets` directory:

```
//source/user/me/datasets/*.txt
```

For the specified directory, the above pattern matches on any text file (.txt). In the example, it matches on the first two files but does not match on the CSV file.

When the Include nested folders checkbox is selected:

- The first two files are matched.
- The next three files inside a nested folder are matched.
- The last file (`ahigherfile.txt`) is not matched, since it is not inside a nested folder.

Example 2: pattern-based files

Suppose your file and folder structure look like the following:

```
//source/user/me/datasets/file01.csv
//source/user/me/datasets/file02.csv
//source/user/me/datasets/file03.csv
//source/user/me/datasets/detail/file04.csv
//source/user/me/datasets/detail/file05.csv
//source/user/me/datasets/detail/file06.csv
//source/user/me/file07.csv
```

You create a pattern parameter on the first file in the `//source/user/me/datasets` directory with the following pattern-based parameter:

```
`file{digit}+`
```

The above pattern matches on the word `file` and a sequence of one or more digits. For example, suppose `file100.csv` lands in the directory at some point in the future. This pattern would capture it.

In the above example, this pattern matches on the first three files, which are all in the same directory.

When the Include nested folders checkbox is selected:

- The first three files are matched.
- The next three files inside a nested folder are matched.
- The last file (`file07.csv`) is not matched, since it is not inside a nested folder.

Wildcard

The easiest way to is to add a wildcard: `*`

A **wildcard** can be any value of any length, including an empty string.

Tip: Wildcard matching is very broad. If you are using wildcards, you should constrain them to a very small part of the overall path. Some running environment may place limits on the number of files with which you can match.

Pattern - Regular expression

Instead of a wildcard match, you could specify a regular expression match. **Regular expressions** are a standardized means of expressing patterns.

Regular expressions are specified between forward slashes, as in the following:

```
/my_regular_expression/
```

NOTE: If regular expressions are poorly specified, they can create unexpected matches and results. Use them with care.

The following regular expression matches the same two sources in the previous screen:

```
/\[0-9]*\[0-9]*/
```

The above expression matches an underscore (`_`) followed by any number of digits, another underscore, and any number of digits.

Tip: In regular expressions, some characters have special meaning. To ensure that you are referencing the literal character, you can insert a backslash (`\`) before the character in question.

While the above matches the two sources, it also matches any of the following:

```
_2_1  
__1  
_1231231231231231231235245234343_
```

These may not be proper matches. Instead, you can add some specificity to the expression to generate a better match:


```
/\[0-9]{13}\_[0-9]{4}/
```

The above pattern matches an underscore, followed by exactly 13 digits, another underscore, and then another 4 digits. This pattern matches the above two sources exactly, without introducing the possibility of matching other numeric patterns.

Pattern - Trifacta pattern match

A Trifacta pattern is a platform-specific mechanism for specifying patterns, which is much simpler to use than regular expressions. These simple patterns can cover much of the same range of pattern expression as regular expressions without the same risks of expression and sometimes ugly syntax.

Trifacta patterns are specified between back-ticks, as in the following:

```
`my_pattern`
```

In the previous example, the following regular expression was used to match the proper set of files:

```
/\[0-9]{13}\_[0-9]{4}/
```

In a Trifacta pattern, the above can be expressed in a simpler format:

```
`\_ {digit}{13}\_ {digit}{4}`
```

This simpler syntax is easier to parse and performs the same match as the regular expression version.

For more information on Trifacta patterns, see *Text Matching*.

Parameterize Tables for Import

Contents:

- *Import Parameterized Tables*
 - *Create a custom SQL dataset*
 - *Parameterize dataset with a variable*
 - *Parameterize dataset with a timestamp*
 - *Examples*
 - *Pre-filter rows from a table*
 - *Run a weekly job on daily tables*
 - *Parameterize entire query*
-

This section provides an overview on how to apply parameters to the tables that you import as datasets.

During import of database tables through relational connections, you can apply parameters to the SQL query that you use to define the imported dataset. In some scenarios, you may need to define the table to import using a variable parameter or to parameterize the time value associated with a table name. Using parameters, you can define the specific tables that you use to bring in data from a relational database.

Following are the type of parameters you can apply for relational sources:

- **Timestamps:** Inserts a formatted timestamp when creating a custom SQL query.
- **Variables:** Inserts a value for the variable. This variable has a default value that you assign.

NOTE: Pattern-based parameters are not supported for relational imports.

Import Parameterized Tables

While importing data, you parameterize relational tables by creating custom SQL statements to specify the dataset. By default, when you import a table from a relational source, Trifacta generates a `SELECT *` statement to import the entire table. The Custom SQL enables you to customize the query to pull the data from the source system.

The following are the prerequisites and procedures for parameterizing the relational sources table:

Prerequisites:

- A connection must be created for your target database.
 - Verify that you have access to a read-only or read-write set of connections.
 - For more information, see *Connect to Data*.

Create a custom SQL dataset

You can create a custom SQL dataset through the Import Data page.

Steps:

1. In the Trifacta application, click **Library** in the left nav bar.
2. In the Library page, click **Import Data**.
3. From the left side of the Import Data page, select the relational connection from which to import.
4. Depending on the type of relational connection, you may need to select the database or schema to browse.

5. Locate the tables to import. Take note of the table name or names.
6. Click **Create Dataset with SQL**. The Create Dataset with SQL window is displayed.

In this window, you specify the `SELECT` statement to retrieve the data from a table or tables that you specify.

NOTE: When specifying a SQL statement for your database, you are constructing a direct query of the database. You must use the syntax required by the database vendor.

For more information on creating datasets with SQL, see *Create Dataset with SQL*.

Parameterize dataset with a variable

A variable parameter enables you to insert variable into the query statement used to define your dataset. You can replace or highlight elements of the query to add parameters.

How to use variables:

- When a job is executed, the currently specified variable value is passed to the running environment. By default, the value that you specify as part of the dataset creation process is provided.
- You can override this value:
 - You can specify an override for a variable parameter through Flow View.
 - For any specific job run, you can specify an override value through the Run Job page.
- In this manner, you can specify the exact data that you wish to retrieve at the flow- or job-level.

Steps:

1. Create a custom dataset using SQL. For more information, see Create a Custom SQL Dataset above.
2. In the Create Dataset with SQL window, enter a `SELECT*` statement to retrieve data from the specified table. Click **Validate SQL** to verify that the query is properly specified.
3. Now, highlight the part of the query that you wish to parameterize. Click the Variable icon.

Figure: Define Variable Parameter

4. In the Variable dialog, enter the following details:

Tip: Type `env.` to see the environment parameters that can be applied. These parameters are available for use by each user in the environment.

- a. **Name:** Enter a display name for the variable.
- b. **Default value :** Enter a default value for the parameter.
5. Click **Save** to save the parameter.
6. To verify that your SQL is still valid, click **Validate SQL**.

7. If the SQL is valid, click **Create Dataset**.

Parameterize dataset with a timestamp

Timestamp parameters can be helpful when you want to filter datasets based on date and time format, time zone, or exact and relative start time. You can apply timestamp parameters based on the specific region or time zone for which the data is generated.

Steps:

1. Create a custom dataset using SQL. For more information, see [Create a Custom SQL Dataset](#) above.
2. In the Create Dataset with SQL window, enter a `SELECT*` statement to retrieve data from the specified table. Click **Validate SQL** to verify that the query is properly specified.
3. Now, highlight the part of the query that you wish to parameterize. Click the Timestamp icon.

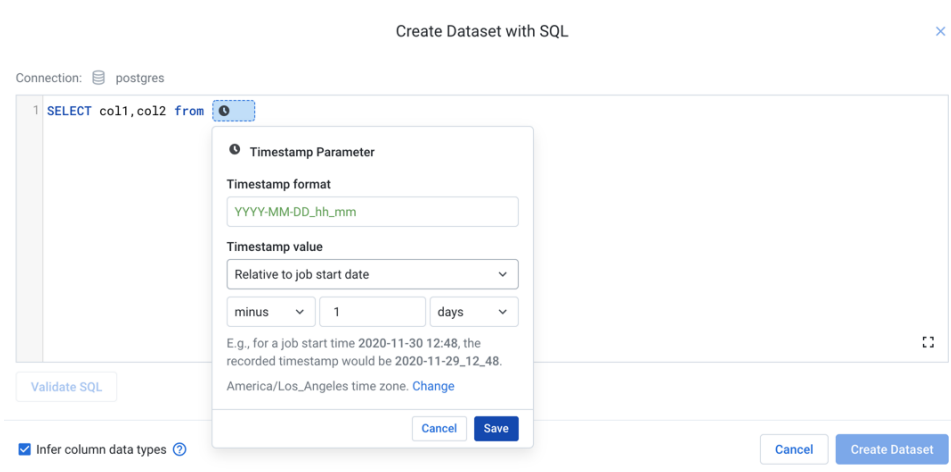


Figure: Define Timestamp Parameter

4. In the Timestamp Parameter dialog, enter the following details:
 - i. **Timestamp format:** Specify the format for timestamp values.
 1. Example: `YYYY-MM-DD_hh_mm`.
 2. Datetime values can be expressed as either date or time elements.
 - ii. **Timestamp value:** Select the value to record in the path:
 1. **Exact job start date:** recorded timestamp in path is the start time of the job.
 2. **Relative to the job start date:** recorded timestamp in path is relative to the start time of the job according to the settings that you specify here.
 - iii. **Time zone:** Click **Change** to change the time zone recorded in the timestamp.
 1. Example: `America/Los Angeles` or `Asia/Calcutta`.
 2. For more information on the available time zones, see [Supported Time Zone Values](#).
5. Click **Save** to save the parameter.
6. To verify that your SQL is still valid, click **Validate SQL**.
7. If the SQL is valid, click **Create Dataset**.

Examples

In the following examples, you can see how dataset parameters can be used to pre-filter rows or parameterize the tables to include in your dataset.

NOTE: The syntax in these examples uses PostgreSQL syntax. The syntax that you use must match the requirements of the target database system.

Pre-filter rows from a table

Suppose you have a set of orders in a single table: `myOrders`. From this table, you want to be able to import a dataset that is pre-filtered for values in the customer identifier (`custId`) column. The following might be a query that you use for the `customerOrders` dataset to retrieve the orders for `custId=0001` from the `myOrders` table in the `transactions` database:

```
SELECT "custId","ordDate","prodId","ordQty","unitPrice" FROM "transactions"."myOrders" WHERE "custId" = "0001"
```

Tip: This example uses a variable parameter.

Steps:

In this case, you can do the following:

1. In the Create Dataset with SQL window, specify the above query. Click **Validate SQL** to verify that it works.
2. Now, highlight the value `0001`.
3. Select the Variable icon.
4. Specify your variable:
 - a. **Name:** `myCustId`
 - b. **Default Value:** `0001`
5. Click **Save**.
6. Before you create the dataset, validate the SQL.

Using the parameter:

- When the job is executed, the `customerOrders` dataset is pre-filtered to retrieve the data for `custId=0001` by default.
- You can override this variable value as needed:
 - At the flow level, you can define an override for the `myCustId` variable. For example, you can set the variable value to: `0002`. Whenever a job is run on this imported dataset, the value `0002` is passed to the running environment, which retrieves only the rows in the table where `custId=0002`.
 - When you run the job through the application, you can specify an override to the variable. This override takes precedence over the flow that was set at the flow level. So, for a specific run, you can set the value to `0003`, generating results for `custId=0003` only.
 - Then, the next time that a job is run from the flow using the dataset, the flow override value (`0002`) is used.

Run a weekly job on daily tables

Suppose you have a database that captures log data into separate tables for each date. Each table is named according to the following pattern:

```
20201101-ServerLogs
20201102-ServerLogs
20201103-ServerLogs
20201104-ServerLogs
20201105-ServerLogs
```

Once per week, you want to run a job to ingest and process the log entries from the preceding week.

The following could be a query that you use to retrieve all columns from a single file:

```
SELECT * FROM "logs"."20201101-ServerLogs"
```

Tip: This example uses a timestamp parameter.

Steps:

In this case, you can do the following:

1. In the Create Dataset with SQL window, specify the above query. Click **Validate SQL** to verify that it works.
2. Now, highlight the value `20201101`.
3. Select the Timestamp icon.
4. Specify your variable:
 - a. **Timestamp Format:** `YYYYMMDD`
 - b. **Timestamp Value:** Relative to job start date
 - i. Select minus, 7, days.
5. Click **Save**.
6. Before you create the dataset, validate the SQL.

Using the parameter:

When the job is executed, the imported dataset includes all of the tables whose timestamp format is within 7 days of the time when the job was started.

- You may need to modify the time zone setting on the Timestamp parameter if the log files were recorded using a different time zone.
- Typically, jobs created on a dataset like this one are executed according to a schedule.
 - For scheduled jobs, the value that is used for the job start date is the timestamp for when the job was scheduled to execute. It's possible that delays in starting the job could create a difference in the timestamps.
 - For more information, see *Schedule a Job*.

Parameterize entire query

You can turn the entire query of your custom SQL statement into a parameter. When you create your dataset with SQL, instead of entering any SQL in the window, create a variable parameter. For example, your parameter could be like the following:

- **Name:** `selectCustomersTable`
- **Value:**

```
SELECT * from "MDM"."customers"
```

If the SQL validates, then you can create the imported dataset using only this parameter.

How to use this parameter:

- By default, when the dataset is imported, all of the columns from the `customers` table are imported.
- As needed, you can configure overrides at the flow- or job-level to, for example, import only select columns. In the override, you specify the list of columns to gather only the data required for your needs.

Tip: This example uses a variable parameter.

Create Dataset with SQL

Contents:

- *Limitations*
 - *General*
 - *Single Statement*
 - *Multi-Statement*
 - *Enable*
 - *Use*
 - *Create with Variables*
 - *Create with timestamp parameter*
 - *SQL Validation*
 - *SQL Syntax*
 - *Troubleshooting*
 - *Snowflake*
-

As needed, you can insert custom SQL statements as part of the data import process. These custom SQL statements allow you to pre-filter the rows and columns of relational source data within the database, where performance is faster. This query method can also be used for wider operations on relational sources from within Trifacta®.

Limitations

General

All queries are blindly executed. It is your responsibility to ensure that they are appropriate. Queries like `DELETE` and `DROP` can destroy data in the database. Please use caution.

NOTE: Column names in custom SQL statements are case-sensitive. Case mismatches between SQL statement and your datasource can cause jobs to fail.

- SQL statements are stored as part of the query instance for the object. If the same query is being made across multiple users using private connections, the SQL must be shared and entered by individual users.

NOTE: If a dataset created from custom SQL is shared, collaborators are not permitted to edit the custom SQL.

- Each statement must be terminated with a semi-colon (;) and a newline:

```
SELECT * FROM myDB.myTable;
```

- SQL statements must be valid for the syntax of the target relational system.
- If you modify the custom SQL statement when reading from a source, all samples generated based on the previous SQL are invalidated.
- Declared variables are not supported.
- Common Table Expressions (CTEs) are not supported.
- For each SQL statement, all columns must have an explicit name. Example:

- Function references such as:

```
UPPER(col)
```

- Must be specified as:

```
UPPER(col) as col_name
```

- When using custom SQL to read from a Hive view, the results of a nested function are saved to a temporary name, unless explicitly aliased.
 - If aliases are not used, the temporary column names can cause jobs to fail, on Spark in particular.
 - For more information, see *Hive Connections*.

Single Statement

The following limitations apply to creating datasets from a single statement.

1. All single-statement SQL queries must begin with a `SELECT` statement.
2. Selecting columns with the same name, even with " * ", is not supported and generates an ambiguous column name error.

Tip: You should use fully qualified column names or proper aliasing. See *Column Aliasing* below.

3. Users are encouraged to provide fully qualified path to table being used. Example:

```
SELECT "id", "value" FROM "public"."my_table";
```

4. You should use proper escaping in SQL.

Multi-Statement

These limitations apply to creating datasets using a sequence of multiple SQL statements.

NOTE: Use of multiple SQL statements must be enabled. See *Enable Custom SQL Query*.

1. **Repeatable:** When using multi-statements, you must verify that the statements are repeatable without failure. These statements are run multiple times during validation, datasets creation, data preview, and opening the dataset in the Transformer page.

NOTE: To ensure repeatability, any creation or deletion of data in the database must occur before the final required `SELECT` statement.

2. **Line Termination:** Each statement must terminate with a semi-colon and a new line. Example:

```
SELECT * FROM transactions.orders;
SELECT custId,custName FROM master.customers;
```

3. **Validation:** All statements are run immediately when validating or creating dataset.

NOTE: No DROP or DELETE checking is done prior to statement execution. Statements are the responsibility of the user.

4. **SELECT requirement:** In a multi-statement execution, the last statement must be a SELECT statement.
5. **Database transactions:** All statements are run in a transaction. DDL statements in most dialects (vendors) can't be run within a transaction and might be automatically committed by the driver.

Enable

Steps:

1. You apply this change through the *Workspace Settings Page*. For more information, see *Platform Configuration Methods*.
2. Locate the following setting:

Enable custom SQL Query

Setting	Description
enabled	Set to true to enable the ability to create datasets using customized SQL statements. By default, this feature is enabled.

Use

To use, please complete the following steps.

Steps:

1. In the Library page, click **Import Data**.
2. In the Import Data page, select a connection.
3. Within your source, locate the table from which you wish to import. Do not select the table.
4. Click the Preview icon to review the columns in the dataset.

Tip: You may wish to copy the database, table name, and column names to a text editor to facilitate generating your SQL statement.

5. Click **Create Dataset with SQL**. Enter or paste your SQL statement.

Through the custom SQL interface, it is possible to enter SQL statements that can delete data, change table schemas, or otherwise corrupt the targeted database. Please use this feature with caution.

NOTE: If this button is disabled and you have enabled the custom SQL feature, the connection that you are using may lack credentials. Please review the connection definition.

Connection: hive

```
1 SELECT INST#, BUCKET# FROM 'AUDSYS'.'CLI_SWP$7395268a$1$1'
```

[Validate SQL](#)

☒ Infer column data types

[Cancel](#) [Create Dataset](#)

Figure: Create Dataset with SQL dialog

- a. To test the SQL, click **Validate SQL**. For details, see below.
- b. To apply the SQL to the import process, click **Create Dataset**.
6. The customized source is added to the right panel. To re-edit, click **Custom SQL**.
7. Complete the other steps to define your imported dataset.
8. When the data is imported, it is altered or filtered based on your SQL statement.

Create with Variables

If parameterization has been enabled, you can specify variables as part of your SQL statement. Suppose you had table names like the following:

```
publish_create_all_types_97912510
publish_create_all_types_97944183
publish_create_all_types_14202824
```

You can insert an inline variable as part of your custom SQL to capture all of these variations.

Connection: hive

```
1 SELECT * from 'default'.'publish_create_all_types_<> iid;
```

<> Variable

Name

Default value

[Cancel](#) [Save](#)

[Validate SQL](#)

☒ Infer column data types

[Cancel](#) [Create Dataset](#)

Figure: Insert variables in your custom SQL

In the above, custom SQL has been added to match the first example table. When the value is highlighted and the icon is clicked, the highlighted value is specified as the default value.

Tip: Type `env.` to see the environment parameters that can be applied. These parameters are available for use by each user in the environment.

Provide a name for the variable, and click **Save**.

Through the Run Job page, you can specify overrides for the default value, so the same job definition can be used across all matching tables without much modification.

Create with timestamp parameter

You can insert a timestamp parameter into your custom SQL. These parameters are used to describe timestamp formats for matching timestamps relative to the start of the job at the time of execution.

NOTE: A SQL timestamp parameter only describes the formatting of a timestamp value. It cannot be used to describe actual values. For example, you cannot insert fixed values for the month to parameterize your input using this method. Instead, parameterize the input using multiple input variables, as described in the previous section.

NOTE: Values for seconds in a SQL timestamp parameter are not supported. The finest supported granularity is at the minutes level.

NOTE: When the dataset is created, the current date is used for comparison, instead of the job execution date.

In the following example, the timestamp parameter has been specified as `YYYY-MM-DD`:

```
SELECT * FROM <YYYY-MM-DD> ;
```

If the job executes on May 28th, 2019, then this parameter resolves as `2019-05-28` and gathers data from that table.

Create Dataset with SQL ✕

Connection: hive

1 `SELECT 1 as`

Timestamp Parameter

Timestamp format

Timestamp value

Exact job start date ▼

E.g., for a job start time 2019-10-18 02:39, the recorded timestamp would be 2019-10-18.

America/Los_Angeles time zone. [Change](#)

Cancel Save

Validate SQL

☒ Infer column data types

Cancel Create Dataset

Figure: Insert timestamp parameter

Steps:

1. Click the Clock icon in the custom SQL dialog.
2. **Timestamp format:** You can specify the format of the timestamp using supported characters.

Tip: The list and definition of available tokens is available in the help popover.

3. **Timestamp value:** Choose whether the timestamp parameter is to match the exact start time or a time relative to the start of the job.

Tip: You can use relative timestamp parameters to collect data from the preceding week, for example. This relative timestamp allows you to execute weekly jobs for the preceding week's data.

4. To indicate that the timestamps are from a timezone different from the system timezone, click **Change**.
5. To save the specified timestamp parameter, click **Save**.

SQL Validation

You cannot create a SQL-based dataset if any of your SQL statements do not pass validation. Errors must be corrected in the SQL or in the underlying database.

- All `SELECT` statements are planned, which includes syntactical validation. However, these statements are not executed. Validation should be a matter of a few seconds.
- For multi-line statements, all non-`SELECT` statements are planned and executed. The final `SELECT` statement is only planned.

NOTE: For multi-line SQL statements, validation may take longer to complete if the non-`SELECT` statements require significant time to execute.

SQL Syntax

For more information on SQL syntax and supported variations, see *Supported SQL Syntax*.

Troubleshooting

Snowflake

Selecting time zone data returns null values in profiling and fails in publishing

When you import a column from Snowflake that contains time zone information, you may see the following behavior:

- Sampled data appears to import correctly into the Transformer page for the TIMESTAMP-based column.
- When a job is run, the visual profile for the output column based on this data indicates null values.
- When the data is published back to Snowflake, the publishing job fails.

The above issue is caused by the following:

- When data is imported into the Transformer page, it is automatically converted to UTC timezone during the JDBC ingestion step for displaying the sample in the application. This ingestion process is called by the application and outside of the application's control.
 - During this ingestion process, some auto-recognition and conversion to UTC of Datetime values is applied to the sample for display.
 - Example: You design a recipe step to parse the following Datetime format: 2020-10-11 12:13:14., which has been auto-converted to UTC.
- When a job is run:
 - The application instructs Snowflake to unload the entire dataset from Snowflake and write it the target location, bypassing this automatic conversion process.
 - The recipe that was created to handle the data in the sample does not properly handle the data that is directly unloaded from Snowflake.
 - In the previous example: The Datetime parsing in your recipe may receive an input that looks very different from what you parsed in the displayed sample: 2020-10-11 14:13:14 CEST.

Solution:

For a time stamp with a time zone, you must wrap your reference to it like the following:

```
TO_TIMESTAMP(CONVERT_TIMEZONE('UTC', <timestamp_column_or_function>))
```

Suppose your query was the following:

```
SELECT *, CURRENT_TIMESTAMP() AS current_time FROM MY_TABLE;
```

To address this issue, the query needs to be rewritten as follows:

```
SELECT *, TO_TIMESTAMP(CONVERT_TIMEZONE('UTC', CURRENT_TIMESTAMP())) AS current_time FROM MY_TABLE;
```

When the above wrapper function is applied, the data is imported normally and validated and published as expected.

Discovery Tasks

Use various tools and techniques to identify patterns, anomalies, inconsistencies, and other issues in your datasets.

Explore Suggestions

Contents:


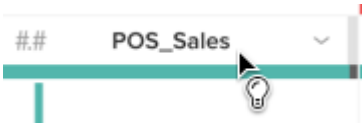
- *Select Something*
- *Suggestion Cards*
- *Decide on the Suggestion*
- *Modify Suggestion*
- *Previews*
- *Iterate*

When you make selections in the Transformer page, Trifacta® responds by posting a set of suggestions for transformations to apply to the selected data in the sample. You can experiment with these suggestions to see what properly transformations your data.

Select Something

Selection Hints:

As you move the cursor around the Transformer page, the cursor changes when it is over a selectable data element.

Icon	Description
	Value or values can be selected.
	Column or columns can be selected.

In the data grid:

- You may select categories of values in a column's data quality bar: Valid, Mismatched, and Missing.
- You may select one or more values in a column's histogram. Use `SHIFT` or `CTRL` to select multiple values.
- Click a column for column-based operations. Click additional columns to add to your selection. Click a selected column to deselect.
- Select a whole or partial cell value to prompt suggestions for managing that specific string of data.

Tip: If you `CTRL`-select multiple partial values in a column of numeric data, the suggestion cards apply to the pattern that matches your selected strings. This does not apply to string data.

In the Column Browser or Column Details:

- Select categories of values in the data quality bar.
- Select one or more values in a column's histogram.

NOTE: Some complex transformations, such as joins and unions, cannot be suggested based on selection of values in the data grid.

Suggestion Cards

Based on your selections, relevant suggestions appear in suggestion cards:

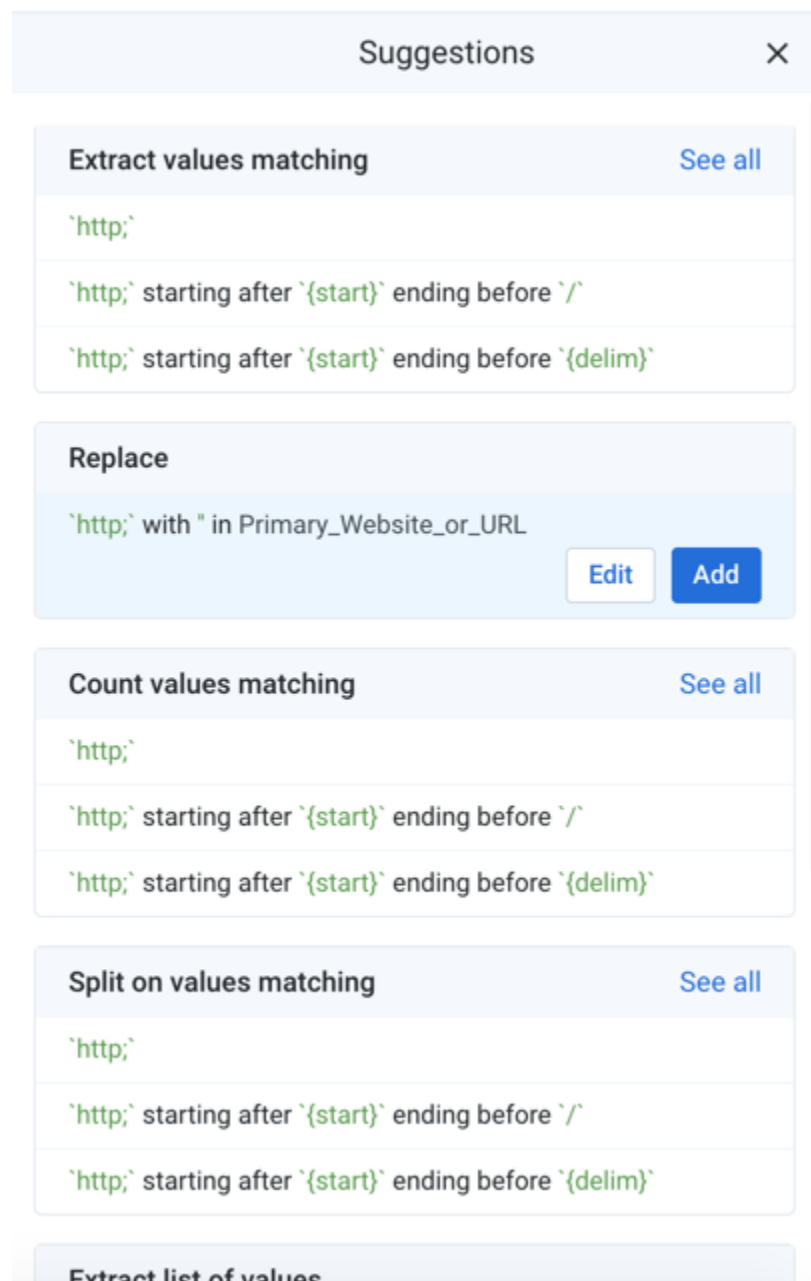


Figure: Suggestion Cards

In the suggestion cards, the label at the top identifies the transformation type that is being recommended, followed by a brief preview of how the selection might transform the data.

Tip: A suggestion card may contain multiple variants for each suggestion. For example, in the previous image the `extract` suggestion has many variants, which can be selected and reviewed by selecting the dots at the bottom of the card.

Additional suggestions may be available. Try horizontal scrolling the set of cards to reveal new suggestions.

Decide on the Suggestion

Before you decide on the suggestion to follow, you can do one of the following:

- **Select the suggestion to use.** After a suggestion is selected, the changes to the data are previewed in the Transformer page immediately. If there are multiple variants for the suggestion, verify that you are selecting the most appropriate one.
- **Select additional columns or values in the Transformer page.** A different pattern-based set of suggestions is presented to you. Make your transformation selection.
- **Modify the suggestion.** You may need to customize the suggestion to meet more specific requirements.
- **Start over.** If you discover that you have selected the wrong example data, click **Cancel**. Start again.

Modify Suggestion

To make the suggestion work for your specific use, you might need to modify the step. For example, for the selected text, you might need to define a replacement value, which Trifacta may not be able to guess. Click **Edit**. The Transform Builder is displayed, where you can edit the details of the transformation.

Previews

As soon as you select a suggestion card, the changes are previewed in the Data Grid:

The screenshot displays the Trifacta Transformer interface. On the left, a data grid shows a list of URLs under the column 'Primary_Website_or_URL'. The grid is divided into 'Source' and 'Preview' sections. The 'Source' section shows the original data, and the 'Preview' section shows the data after applying a suggestion. The suggestions on the right include 'Extract values matching', 'Replace', 'Split on values matching', and 'Count values matching'. The 'Replace' suggestion is selected, and its details are shown in the 'Replace' section. The 'Replace' section shows the suggestion: 'http;' with ' ' in Primary_Website_or_URL. The 'Edit' button is visible next to the suggestion. The 'Split on values matching' section shows the suggestion: 'http;' starting after '{start}' ending before '/' and 'http;' starting after '{start}' ending before '{delim}'. The 'Count values matching' section shows the suggestion: 'http;' starting after '{start}' ending before '/' and 'http;' starting after '{start}' ending before '{delim}'. The 'Extract list of values' section is also visible.

Figure: Previewed suggestion

In this manner, you can review the change before it is applied to the sample.

Tip: You can use the checkboxes in the status bar to display only the rows, columns, or both that are affected by the previewed transformation.

Iterate

Experiment away! Things to keep in mind:

- If you select the wrong thing, you can always cancel the recipe step. Start again.
- To delete a step that has already been added, select the step in the Recipe panel and click the Trash icon to delete it.
- To step back a number of steps in the recipe, select the recipe to which you want to revert and start adding steps. Note that any added steps may invalidate the subsequent steps in your recipe.
- You can always undo and redo your most recent actions. Use the buttons on the top of the Recipe panel.
- An executed recipe does not change the source, so you can always step back to your recipe in the Transformer page and revert or modify recipe steps.

Add or Edit Recipe Steps

You can add or edit steps in your recipe through the Recipe panel, which is available on the right side of the Transformer page.

Steps:

To add or edit steps in your recipe, do the following:

- 1. If it's not already opened, open the recipe panel:

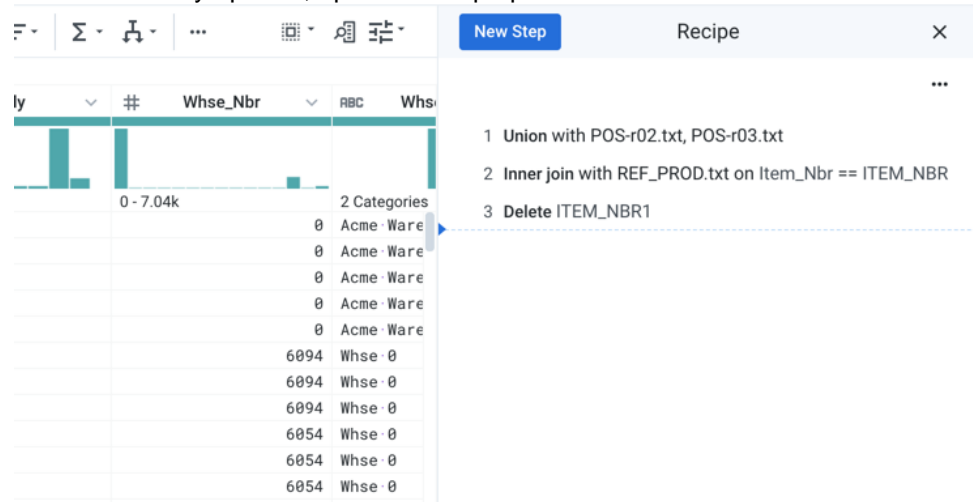


Figure: Recipe Panel

- 2. Edit a step:
 - a. Select the step in the recipe.
 - b. Click the Pencil icon.
 - c. Skip the next step.
- 3. Add a step:
 - a. In the recipe, select the step next to where you would like to add the step.
 - b. Select **Insert step before** or **Insert step after** from the drop-down menu.
- 4. To specify a step, you can:
 - a. Select something in the data grid. A set of suggestions is provided to you in the Selection Details panel.
 - b. Enter some text in the Search panel. For the selected transformation, specify required and optional parameters in the Transform Builder to see a preview of the transform.
- 5. After you have specified your step:
 - a. To add it to the recipe as it is currently specified, click **Add**. The step is inserted in the proper location.
 - b. To modify it, click **Edit**. You can edit the step in the Transform Builder.

Filter Data

Contents:

- *Filter Dataset*
 - *Filter Data Grid*
 - *Toggle display of columns*
 - *Filter the data grid*
 - *Filter during previews*
-

In the Transformer page, you can filter data from display in the data grid or from the dataset permanently.

Filter Dataset

You can apply various tools to remove columns of data and rows based on conditions you define. For more information on how to permanently remove rows and columns of data from the sample and the dataset, see *Remove Data*.

Filter Data Grid

You can make selections in the data grid interface to filter the sampled data that is displayed in the data grid.

Depending on your current tasks, you may want to hide columns or rows of the sample, so that you can focus on the task at hand.

- The displayed sample for smaller datasets may be the full dataset.
- Columns or filtered rows that are hidden from view are not removed from the dataset. They are included in any output. Please note that hidden columns can be affected by recipe steps. You should get in the habit of reviewing the Visible Columns panel and the Filters panel before running a job.

NOTE: Data grid filters do not remove any data. They can be used to hide data that is not important for the task at hand. The hidden data is still part of the sample and the full dataset.

Toggle display of columns

- To toggle display of a single column in the data grid, select the drop-down next to the column name. Then, select **Edit column > Hide**.
- To show a hidden column, click the Eye icon in the status bar at the bottom of the page. In the Visible Columns panel, click the Eye icon next to the column name. The column is displayed again in the data grid or column browser.

Tip: You can use the Visible Columns panel to toggle the display of single columns or multiple columns at the same time.

- You can also hide one or more columns through the Column Browser:
 - In the Transformer page, click the Columns icon in the toolbar.
 - In the Column Browser, select the column or columns to hide. From the Actions drop-down, select **Edit > Hide**.
 - Hidden columns must be resurfaced through the Visible Columns panel.

Filter the data grid

In the data grid panel, you can apply row- or column-based filters. At the top of the data grid, click **Filters**. In the Filter panel:

- **Columns:** Search for individual columns or filter columns of a specific type. Filtered columns are displayed, and the rest are hidden.
- **Rows:** Highlight search term matches found in any column for a row.

Filter during previews

When you are constructing transforms, the expected results are previewed in the data grid. As needed, you can narrow the display to only the affected rows, columns, or both. Select the appropriate checkbox or checkboxes in the status bar at the bottom of the Transformer page.

Locate Outliers

Contents:

- *Single-column outliers*
 - *Data Histogram*
 - *Column Details*
 - *Tune standard deviation calculations*
 - *Custom functions*
 - *Methods for fixing single-column outliers*
-

Before you begin performing analytics on a dataset, it is important to identify and recognize outlier data patterns and values.

Unusual values or patterns in the data can be sources for the following:

- Missing data.
- Bad data.
- Poorly formatted data
- Mismeasured data
- Data that skews statistics

This section provides guidance in how to locate these patterns of data in individual columns.

Single-column outliers

For assessing anomalies in individual columns, Trifacta® provides visual features and statistical information to quickly locate them.

Data Histogram

You can use the data quality bar and histogram to locate unusual values in your column data. The following example illustrates a dataset that contains two columns with outlier data. The first two rows are outliers with the subsequent rows to be consistently patterned data:Click to download the *Dataset-Outliers.csv* example data.

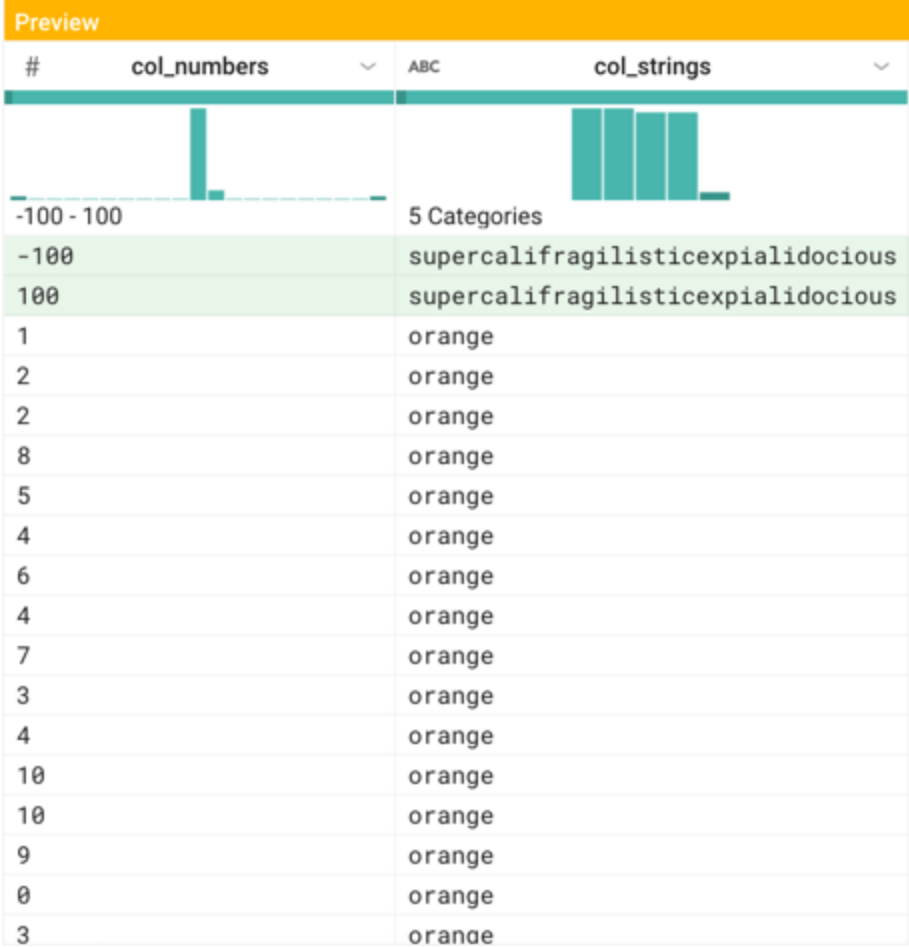


Figure: Numeric and string anomalies

Numeric data

The `col-numbers` column contains 100 random values 0-10, and singleton values -100 and 100.

In the histogram, you can see the outliers at the extremes of the graph. Note the slight visual distinction between the two extreme values and the values next to them, which are not represented in the column data.

Tip: In a histogram for numeric data, the spread between the extreme values and the more frequent values is a visual cue for outliers.

For numeric data, the range of values is displayed as part of the histogram. In this dataset, the extreme values are singletons. If a dataset contains more instances of outlier values, you should investigate further.

NOTE: In numeric datasets, a high count of outlier values may be statistically significant. You should review those values and related data in other columns before you perform operations to change or remove those rows.

Significant counts of unusual values

When your data contains a significant number of specific values, you should review them to see if the values have meaning. They may be placeholders for missing values.

For numeric data, you should be skeptical of occurrences of the following values:

Suspicious value	Reason
-1	In system generated data, -1 is often an indicator of a failed result of some kind.
0	Some systems will fill missing numeric values with the number 0. You should verify the meaning of the value of 0 in your dataset.
555-####	In the United States, the phone number prefix 555 never corresponds to a person's phone number. These informational phone numbers and should not be considered as valid values for individuals' data.
65535	<div>In older versions of Microsoft Excel, 65,535 was the maximum number of rows permitted in a single sheet. NOTE: 65,536 is 2^{16}, which is the maximum number of data bits in a 16-bit system.</div>
2147483647	This value is the largest positive integer that can be stored in an <code>int</code> datatype by 32-bit systems, which are still sources of data. If you see these values, the source system may have been unable to represent the true value and wrote this value instead.
4294967295	This value is the largest raw value that can be stored in 32-bit systems. If you see these values, the source system may have been unable to represent the true value and wrote this value instead.
January 1st, 1900	This value is the earliest date recognized by Microsoft Excel. The true date may not be accurately represented in your data.
January 1st, 1904	This value is the earliest date recognized by Microsoft Excel for Macintosh.
00:00:00 UTC on January 1, 1970	This value is the earliest recognized date in UTC timestamp values. UTC timestamps are recorded as the number of milliseconds since this moment in time, stored as a signed 32-bit integer. Since datetime values may be represented in many different formats, you should identify these values for the date formats in your dataset.
03:14:07 UTC on Tuesday, 19 January 2038	<div>This value is the latest recognized date in UTC timestamp values. Since datetime values may be represented in many different formats, you should identify these values for the date formats in your dataset.<ul style="list-style-type: none">This limit is generally known as the "Year 2038" problem.</div>

String data

The `col-strings` column contains approximately 25 values for orange, red, green, yellow, and two instances of supercalifragilisticexpialidocious.

NOTE: For string-based data, outliers can be identified as strings with a low count of instances. These are the shorter stacks in the histogram.

Column Details

In the Column Details panel, you can review detailed statistics on the values in the currently selected column, including data on outliers. In the Transformer page, select **Column Details** from a column's drop-down.

Tip: In the Column Details panel, you can select specific outlier values, prompting suggestions, which enables you to take action on values identified by the platform as outliers.

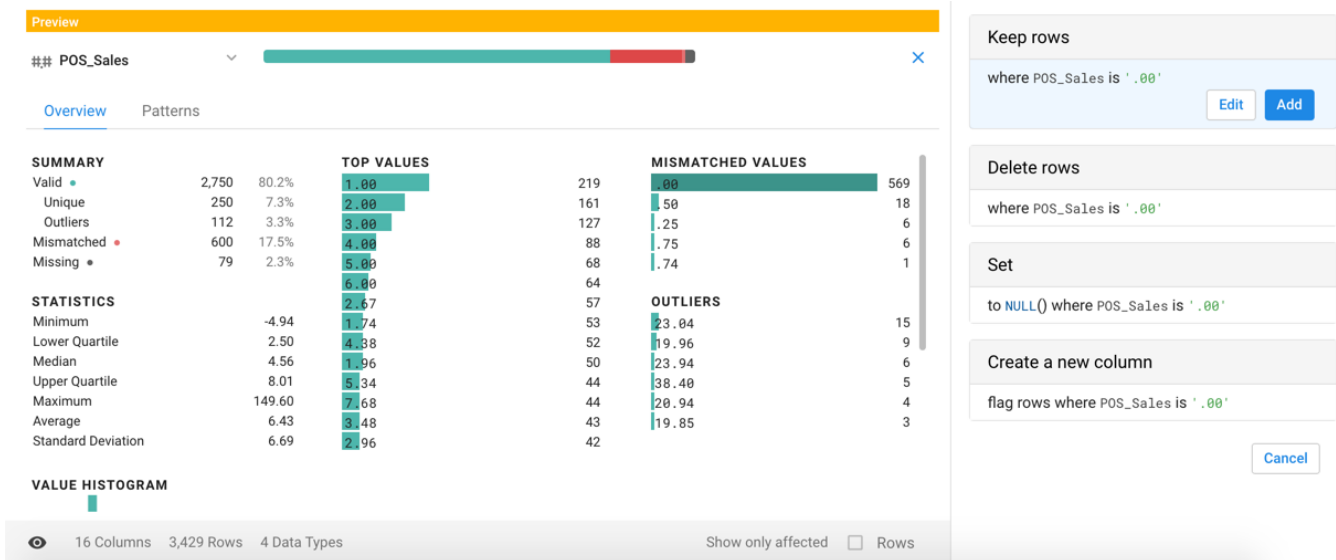


Figure: Outliers in the Column Details

Column Detail Statistics

The Column Details panel provides information on the following:

- Count of valid, mismatched, and missing values
- Count of value instances
- Min, max, and average
- Outlier values. See below.
- Lowest and highest quartiles
- Standard deviation

NOTE: For string-based data types, these statistics pertain to string length.

Tip: Any green bar in the Column Details panel can be selected to prompt for suggestions on actions, including values in Outliers, Value Histogram, and Frequent Values graphs. Multi-select values as needed.

Outliers

Trifacta uses a special set of computations to identify values that it designates as outliers.

For more information on these computations and other calculations in the Column Details panel, see *Column Statistics Reference*.

Tune standard deviation calculations

Although standard deviation information is available in the Column Details, you may want to generate your own standard deviation calculation. For example, the following transform generates a new column which computes the number of standard deviations that a column value is from the average value for the column:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	$\frac{(\text{col_numbers} - \text{AVERAGE}(\text{col_numbers}))}{\text{STDEV}(\text{col_numbers})}$

You can then compute your own outlier function, using something like the following, which assumes that the above derived column has been renamed `col_numbers_stdev` and identifies outliers greater than 4 standard deviations from the average:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>ABS(col_numbers_stdev) > 4</code>

The above function generates boolean values in a new column, setting the value to `true` if the absolute value of the standard deviation for the `col_numbers_stdev` is more than 4. You can then perform operations based on the values written to this column or leave the column in place for downstream analytics tools.

The variance function is also supported.

Custom functions

If necessary, developers can build their own custom functions in their preferred programming language and then import them into the platform. See *User-Defined Functions*.

Methods for fixing single-column outliers

After you have identified the values that are outliers in your column, you must determine if those values are valid or invalid for your dataset. For example, a value of 0 may be a valid measurement, or it may be a value that was inserted for lack of a valid value.

For invalid values:

- **Fix the values.** The fix may require converting the values to be valid for the column's data type. For example, on import, values for 0 and 1 may be written as `false` or `true`. The following steps convert them back to numeric values:

Transformation Name	Edit column with formula
Parameter: Columns	<code>col_numbers</code>
Parameter: Formula	<code>IF((col_numbers == 'false'),'0', col_numbers)</code>

Transformation Name	Edit column with formula
Parameter: Columns	<code>col_numbers</code>
Parameter: Formula	<code>IF((col_numbers == 'true'),'1',col_numbers)</code>

- **Delete the rows.** If the removal of these records does not skew your data, you can create a simple delete statement. For example, the following deletes rows where the value in the `col_numbers` column is less than 25:

Transformation Name	Filter rows
---------------------	-------------

Parameter: Condition	Custom formula
Parameter: Type of formula	Custom single
Parameter: Condition	<code>col_numbers < 25</code>
Parameter: Action	Delete matching rows

For valid values:

- **Let them be.** If the data is valid, do not remove it unless you have an explicit reason for doing so.
- **Convert to more meaningful values.** You can use the `set` transform to change outlier values to values that are valid for purposes of analysis.

NOTE: Please be aware that changing of values may impact the validity of your statistical analysis.

Example of overwriting values where values in the `col_numbers` column that are below 25 are set to the average value for the column. Otherwise, use the current value:

Transformation Name	Edit column with formula
Parameter: Columns	<code>col_numbers</code>
Parameter: Formula	<code>IF((col_numbers < 25), AVERAGE(col_numbers), col_numbers)</code>

Compute Counts

Contents:

- *Important Note on Counts*
 - *Visual Profiling*
- *Row and Column Counts*
 - *Computed row counts*
- *Count by Pattern*
 - *Count pattern or text*
 - *Count between patterns*
- *Count Functions*
 - *Aggregated counts*
 - *Conditional count functions*

Trifacta® supports computation of counts of rows, columns, and ad-hoc values within your data, so that you can make assessments of the quality, consistency, and statistical validity of your data.

Important Note on Counts

Any computed counts that you see in the Transformer page are computed from the displayed sample.

These computed counts reflect the entire dataset, only if the data grid is displaying the full dataset:



Figure: Data grid sample is the full dataset.

When the job is executed, however, any computations of counts are applied across the entire dataset.

Visual Profiling

When you run a job, you can enable the profiling of the job results, which renders a visual profile and some statistics on the dataset. This profile is available for review through the application. For more information, see *Overview of Visual Profiling*.

Row and Column Counts

In the status bar at the bottom of the data grid, you can review the current count of rows and columns in the displayed sample.

Tip: The row and column counts in the status bar may be useful for comparing the changes to these metrics between steps. For example, you can click step 2 in your recipe and then review these metrics. When you click step 3, these metrics may change.

Row counts: Depending on your method of sampling, the row counts may change. For more information, see *Overview of Sampling*.

Column counts: By default, all columns in the panel are displayed. Column counts should change only if you delete or hide them.

Computed row counts

You can use the following functions to identify and compute the row counts in your dataset.

Function Name	Description
<i>COUNT Function</i>	Generates the count of rows in the dataset. Generated value is of Integer type. Tip Typically, this function is used as part of an aggregation, in which rows are grouped according to shared values in other columns. This function can also be applied without grouping, which is called a flat aggregate. More information on how to apply aggregated counts is below.
<i>ROWNUMBER Function</i>	Generates a new column containing the row number as sorted by the <code>order</code> parameter and optionally grouped by the <code>group</code> parameter.
<i>SOURCE ROW NUMBER Function</i>	Returns the row number of the current row as it appeared in the original source dataset before any steps had been applied. NOTE: This function may fail to return results if the original source row information is not available. For example, if you have performed a join between multiple datasets, the source row number information cannot be computed. Similarly, if you compute this function and then perform a join, the results may not make sense. Tip: You can pair this function later with the <code>MIN</code> or <code>MAX</code> functions to compute the highest and lowest row number information.

Count by Pattern

These transformations allow you to compute counts of literals or patterns in a cell's values. Then, you can perform calculations on this new column of values to compute metrics across the dataset.

Count pattern or text

The following example computes the number of references in the `tweet` column for `My Company`:

Transformation Name	Count matches
Parameter: Option	Text or pattern
Parameter: Text or pattern to count	'My Company'
Parameter: New column name	tweetCompanyReferences

Suppose, however, that the company has multiple ways in which it is reference. It could be:

- My Company
- My Co
- My Company, Inc.

You can modify the above transformation to use a `Pattern` to capture these variations:

Transformation Name	Count matches
----------------------------	---------------

Parameter: Option	Text or pattern
Parameter: Text or pattern to count	`(My Company My Co My Company, Inc.)`
Parameter: New column name	tweetCompanyReferences

If needed, you can use the following to add up all of the counts in `tweetCompanyReferences` to determine the total number.

NOTE: Keep in mind that this sum reflects only the sum of values in the sample in the data grid. When you run a job containing this calculation, it is applied across all rows in the dataset.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>SUM(tweetCompanyReferences)</code>
Parameter: New column name	<code>sum_tweetCompanyReferences</code>

Count between patterns

You can also collect counts of values between two patterns within a cell's value. In this manner, you can analyze a more constrained substring of the cell value.

The following transformation calculates the URLs in each row of the `msgText` column, assuming that the URL begins with `http://` or `https://` and ends with `.com` or `.net`:

Transformation Name	Count matches
Parameter: Option	Between two delimiters
Parameter: Starting pattern	`(http\:\/\ https\:\/\)`
Parameter: Include as part of the match	Selected
Parameter: Ending pattern	`(\.com \.net)`
Parameter: Includes as part of the match	Selected
Parameter: Ignore case	Selected
Parameter: New column name	<code>countURLs</code>

Count Functions

Aggregated counts

You can perform calculations based on groups that you define as part of the calculation. These groupings, called **aggregations**, are powerful tools for delivering insightful analysis on your data.

In the following example, several aggregated computations, including the `COUNT` function are performed on transactional data, which is grouped by region (`regionId`) and product (`prodId`):

--	--

Transformation Name	Group by
Parameter: Group by 1	regionId
Parameter: Group by 2	prodId
Parameter: Values 1	SUM(sales)
Parameter: Values 2	COUNT()
Parameter: Type	Group by as new column(s)

NOTE: The above calculation inserts two new columns into the dataset. Alternatively, you can choose to do a full replacement of the dataset with these aggregated counts. For more information, see *Pivot Data*.

Conditional count functions

You can use a set of functions that count occurrences, based on conditions. In the following list of functions:

- Some of the conditions are implicit in the function itself. For example, `COUNTA` counts values that are non-null.
- Some conditions are specified as part of the function. For example, `COUNTIF` tabulates counts provided a specified condition is met.

Function Name	Description
<i>COUNTIF Function</i>	Generates the count of rows in each group that meet a specific condition. Generated value is of Integer type.
<i>COUNTA Function</i>	Generates the count of non-null rows in a specified column, optionally counted by group. Generated value is of Integer type.
<i>COUNTAIF Function</i>	Generates the count of non-null values for rows in each group that meet a specific condition.
<i>COUNTDISTINCT Function</i>	Generates the count of distinct values in a specified column, optionally counted by group. Generated value is of Integer type.
<i>COUNTDISTINCTIF Function</i>	Generates the count of distinct non-null values for rows in each group that meet a specific condition.

The following transformation counts the rows where the length of `msgText` is longer than 140 characters, grouped by `userId`:

Transformation Name	Group by
Parameter: Group by 1	userId
Parameter: Values 1	COUNTIF(LEN(msgText) > 140)
Parameter: Type	Group by as new column(s)

Calculate Metrics across Columns

You can use a variety of mathematical and statistical functions to calculate metrics within a column.

To calculate metrics across columns, you can use a generalized version of the following example.

Source:

Your dataset tracks swimmer performance across multiple heats in a race, and you would like to calculate best, worst, and average times in seconds across all three heats. Here's the data:

Racer	Heat1	Heat2	Heat3
Racer X	37.22	38.22	37.61
Racer Y	41.33	DQ	38.04
Racer Z	39.27	39.04	38.85

In the above data, Racer Y was disqualified (DQ) in Heat 2.

Transformation:

To compute the metrics, you must bundle the data into an array, break out the array into separate rows, and then calculate your metrics by grouping. Here are the steps:

1. When the data is imported, you may need to create a header for each row:

Transformation Name	Rename columns with a row
Parameter: Option	Use row as header
Parameter: Row	1

2. The columns containing heat time data may need to be retyped. From the drop-down next to each column name, select Decimal type.
3. The DQ value in the Heat2 column is invalid data for Decimal type. You can use the following transformation to turn it into a missing value. For purposes of calculating averages, you may or may not want to turn invalid data into zeroes or blanks. In this case, replacing the data as 0.00 causes improper calculations for the metrics.

Transformation Name	Replace text or patterns
Parameter: Column	Heat2
Parameter: Find	'DQ'
Parameter: Replace with	' '

4. Use the following to gather all of the heat data into two columns:

Transformation Name	Unpivot columns
Parameter: Columns	Heat1,Heat2,Heat3
Parameter: Group size	1

5. You can now rename the two columns. Rename `key` to `HeatNum` and `value` to `HeatTime`.
6. You may want to delete the rows that have a missing value for `HeatTime`:

Transformation Name	Delete rows
Parameter: Condition	ISMISSING([value])

7. You can now perform calculations on this column. The following transformations calculate minimum, average (mean), and maximum times for each racer:

Transformation Name	New formula
Parameter: Formula type	Multiple row formula
Parameter: Formula	MIN(HeatTime)
Parameter: Group rows by	Racer
Parameter: New column name	'BestTime'

Transformation Name	New formula
Parameter: Formula type	Multiple row formula
Parameter: Formula	AVERAGE(HeatTime)
Parameter: Group rows by	Racer
Parameter: New column name	'AvgTime'

Transformation Name	New formula
Parameter: Formula type	Multiple row formula
Parameter: Formula	MAX(HeatTime)
Parameter: Group rows by	Racer
Parameter: New column name	'WorstTime'

8. To make the data look better, you might want to reformat the values in the `AvgTime` column to two decimal points:

Transformation Name	Edit column with formula
Parameter: Columns	AvgTime
Parameter: Formula	NUMFORMAT(AvgTime, '##.00')

Results:

After you use the Move transformation to re-organize your columns, the dataset should look like the following:

Racer	HeatNum	HeatTime	BestTime	WorstTime	AvgTime
Racer X	Heat1	37.22	37.22	38.22	37.68
Racer X	Heat2	38.22	37.22	38.22	37.68
Racer X	Heat3	37.61	37.22	38.22	37.68
Racer Y	Heat1	41.33	38.04	41.33	39.69
Racer Y	Heat3	38.04	38.04	41.33	39.69
Racer Z	Heat1	39.27	38.85	39.27	39.05
Racer Z	Heat2	39.04	38.85	39.27	39.05
Racer Z	Heat3	38.85	38.85	39.27	39.05

Compare Strings

Contents:

- Find Substrings
- Compare String Ends by Pattern
- Match Strings
 - Exact matching
 - Doublemetaphone matching
- Compare Strings

Unlike other types of data, text data has very few restrictions on the kinds of values that appear in a cell. In the application, this data is typically inferred as String data type. As a result, finding string values that mean the same thing can be a challenge, as minor differences in their content or structure can invalidate a match.

This section provides some methods for comparing strings.

- Some target systems may impose limits on the lengths of imported values. For more information on managing the lengths of your strings, see *Manage String Lengths*.

Find Substrings

You can use the following functions to locate sub-strings that are part of a column's value.

Function Name	Description
LEFT Function	Matches the leftmost set of characters in a string, as specified by parameter. The string can be specified as a column reference or a string literal.
RIGHT Function	Matches the right set of characters in a string, as specified by parameter. The string can be specified as a column reference or a string literal.
FIND Function	Returns the index value in the input string where a specified matching string is located in provided column, string literal, or function returning a string. Search is conducted left-to-right.
RIGHTFIND Function	Matches the right set of characters in a string, as specified by parameter. The string can be specified as a column reference or a string literal.
SUBSTRING Function	Matches some or all of a string, based on the user-defined starting and ending index values within the string.

The following transformation checks the left five values of the lowercase version of the ProdId column to see if it matches xxxx-. If the value is detected, then the ProdName value is set to NO_NAME:

Transformation Name	Edit with formula
Parameter: Columns	ProdName
Parameter: Formula	IF(LEFT(LOWER(ProdId,5))='xxxx-', 'NO_NAME', ProdName)

Compare String Ends by Pattern

You can use the STARTSWITH and ENDSWITH functions to determine if a string begins or ends with a specified pattern.



Tip: These functions are most useful for performing pattern-based checks on strings. For string literals, you can use the `LEFT` and `RIGHT` functions. See below.

The following transformation inserts the value `error` in the `custCodeStatus` column if the `custCode` value begins with six digits in a row:

Transformation Name	Edit with formula
Parameter: Columns	<code>custCodeStatus</code>
Parameter: Formula	<code>IF(STARTSWITH(custCode,`{digit}{6}`), 'error',custCodeStatus)</code>

Function Name	Description
<i>STARTSWITH Function</i>	Returns <code>true</code> if the leftmost set of characters of a column of values matches a pattern. The source value can be any data type, and the pattern can be a Pattern , regular expression, or a string.
<i>ENDSWITH Function</i>	Returns <code>true</code> if the rightmost set of characters of a column of values matches a pattern. The source value can be any data type, and the pattern can be a Pattern , regular expression, or a string.

Match Strings

Exact matching

You can use the `EXACT` function to compare if two strings are exact matches. String inputs can be literals, column references, or expressions that evaluate to strings.

NOTE: The `EXACT` function evaluates for exact matches. Whitespace or capitalization differences return `false`.

You can nest function expressions inside of the `EXACT` reference to eliminate common and perhaps not useful differences between strings. In the following transformation, a value of `true` is inserted into the `matches` column, if `colA` and `colB` are exact matches, after whitespace and case differences have been removed:

Transformation Name	New formula
Parameter: Formula	<code>IF(EXACT(LOWER(REMOVEWHITESPACE(colA)))==EXACT(LOWER(REMOVEWHITESPACE(colB))) , 'true' , 'false')</code>
Parameter: New column name	<code>matches</code>

Doublemetaphone matching

The platform also supports the doublemetaphone algorithm for fuzzy matching. This algorithm provides mechanism for proximity matching; the `DOUBLEMETAPHONEEQUALS` function supports an optional second parameter to define the strength of the algorithm.

This algorithm works by generating two separate encodings for each string: a primary encoding and a secondary encoding. You can experiment with these encodings using the `DOUBLEMETAPHONE` function. See *DOUBLEMETAPHONE Function*.

This algorithm can be applied to compare two strings, as in the following transformation.

Transformation Name	New formula
----------------------------	-------------

Parameter: Formula	<code>DOUBLEMETAPHONEEQUALS(colA,colB,'strong')</code>
Parameter: New column name	<code>matches</code>

- The first two parameters of the function are the string literals, column references, or functions returning strings to compare.
- The third parameter is optional. It determines the level of matching required to return `true`. Options:

Match threshold	Description
'strong'	Both primary encodings must match.
'normal'	At least one primary encoding must match either of the other string's encodings
'weak'	A primary or secondary encoding from each can match.

- For more information, see *DOUBLEMETAPHONEEQUALS Function*.

Compare Strings

For string values, you can use the string comparison functions to check how strings compare using Latin collation settings.

Tip: Any column can be converted to String data type to use these functions.

Collation refers to the organizing of written content into a standardized order. String comparison functions utilize collation rules for Latin. A summary of the rules:

- Comparisons are case-sensitive.
 - Uppercase letters are greater than lowercase versions of the same letter.
 - However, lowercase letters that are later in the alphabet are greater than the uppercase version of the previous letter.
- Two strings are equal if they match identically.
 - If two strings are identical except that the second string contains one additional character at the end, the second string is greater.
- A **normalized version** of a letter is the unaccented, lowercase version of the letter. In string comparison, it is the lowest value of all of its variants.
 - a is less than .
 - However, when compared to b, a = .
 - The set of Latin normalized characters contains more than 26 characters.

This table illustrates some generalized rules of Latin collation.

Order	Description	Lesser Example	Greater Example
1	whitespace	(space)	(return)
2	Punctuation	'	@
3	Digits	1	2
4	Letters	a	A
5		A	b

Resources:

NOTE: In the following set of charts (linked below), the values at the top of the page are lower than the values listed lower on the page. Similarly, the charts listed in the left nav bar are listed in ascending order.

For more information on the applicable collation rules, see <http://www.unicode.org/charts/collation/>.

Available functions:

- *STRINGLESSTHAN Function*
- *STRINGLESSTHANEQUAL Function*
- *STRINGGREATERTHAN Function*
- *STRINGGREATERTHANEQUAL Function*

Analyze across Multiple Columns

This section describes some techniques for performing analysis across data stored in multiple columns.

For example, you may want to analyze combinations of height and weight. Some options:

- **Consolidate dimensions to a single metric.** For example, height and weight can be combined using a BMI (body mass index) calculation. Then, use available outlier analysis capabilities in Trifacta®. Below, you can review a method for bringing together similar data from multiple columns into a single column for easier analysis.
- **Flag outlier values of individual columns**, perhaps giving each column a weighting factor (e.g. 0.5). Sum the outliers and their weights together.
- **Defer analysis** until the data has arrived in the target system.
- **Build a custom function** in another programming language. See *User-Defined Functions*.

If you have homogeneous data across multiple columns, such as multiple individual events recorded in a single row, you can use a different method to calculate metrics. See *Calculate Metrics across Columns*.

In some cases, you may need to identify outliers across multiple columns of data. For example, you have a dataset containing scores from three separate tests taken by a set of individuals. Your columns may look like the following:

- LastName
- FirstName
- TestScore1
- TestScore2
- TestScore3

You can download the *Dataset-TestScores.csv* dataset.

Most calculations, such as standard deviation, work for a single column of data. To perform analysis across all three columns, you must reshape the above dataset to look like the following:

- LastName
- FirstName
- TestNumber
- TestScore

This steps below outline the workflow for this example. The full recipe is provided at the bottom of this section.

Steps:

1. Load the TestScores dataset into the Transformer page. It should already be split out into five separate columns.
2. The three columns listed side by side are data that has been organized in a pivot table. To break down this data, you must unpivot the data, which breaks down the data into a *key* column (containing TestScore1, TestScore2, TestScore1) and a *value* column, which contains individual test scores.

Transformation Name	Unpivot columns
Parameter: Columns	TestScore1,TestScore2,TestScore3
Parameter: Group size	1

3. Rename the generated column of test scores to TestScore.
4. The numeric information in the *key* column values can be extracted using the following:

Transformation Name	Extract text or pattern
---------------------	-------------------------

Parameter: Column to extract from	key
Parameter: Option	Custom text or pattern
Parameter: Text to extract	`{digit}`

- The key2 column contains just the numeric data now. Rename this column to TestNumber. You can delete the key column now.
- The dataset does not contain a primary key, which field containing a unique identifier for each row. The combination of last name, first name, and test number is a unique identifier for each row in the dataset:

Transformation Name	Merge columns
Parameter: Columns	LastName, FirstName, TestNumber
Parameter: Separator	' - '

- Rename the new column to TestID. Typically, primary keys are listed as the first field in a dataset. You might want to move the column before the LastName column.
- You may have noticed that the data is still organized by name (first and last) and test number, so that an individual's tests are scattered throughout the dataset. To reorganize the information, you can re-aggregate the data using the following:

Transformation Name	Pivot table
Parameter: Row labels	LastName, FirstName, TestNumber, TestID
Parameter: Values	SUM(TestScore)
Parameter: Max number of columns to create	1

Tip: The above retains all instances of tests that have been taken. If you are only interested in the average test score, you can remove the TestNumber and TestID groupings and the change the SUM function to AVERAGE. In the results, you have one average for each test taker.

- You may want to rename the aggregation column. Your final dataset should look like the following:

ABC	LastName	ABC	FirstName	#	TestNumber	ABC	TestID	#	TestS
<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><</div>									

- Delete Randomizer
- Convert 3 columns into rows
- Extract `{digit}` from key
- Rename key1 to 'TestNumber'
- Rename value to 'TestScore'
- Concatenate 3 columns separated by ' - '
- Pivot and compute SUM(TestScore) grouped by 4 columns
- Rename sum_TestScore to 'TestScore'

Figure: Single column of test scores

Now that your columns of data have been consolidated to a single column, you can use the single-column transforms and functions to perform analysis, such as locating outliers.

Parse Fixed-Width File and Infer Columns

For datasets that have a fixed width for each row, determining the column breaks can be more challenging, due to the uncertain number of spaces and tabs between each data element. With enhanced pattern matching, the application can help you identify the appropriate locations to break columns and then trim down the data to eliminate the whitespace padding.

Steps:

1. Import your fixed-width dataset through the application and begin wrangling.
2. The data should now look similar to the following:

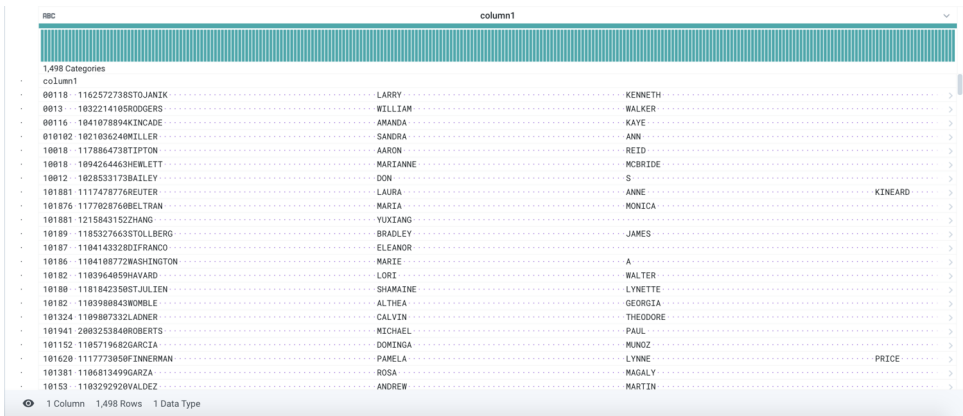


Figure: Fixed-width dataset after import

3. From the drop-down to the right of the column name, select **Column Details**.
4. In the Column Details panel, click the Patterns tab.
5. Click in the All Patterns area.

NOTE: Selecting a specific pattern token will generate suggestions for only that particular token.

NOTE: If the application has inferred that the dataset is fixed-width, then the All Patterns area is the only available selection. If the dataset is not inferred as fixed-width, you should see multiple categories of patterns.

6. From the suggestion cards, click the Split one.
7. Close the Column Details panel.
8. In the Transform preview window, verify that the column splits look ok.
 - a. If a column contains multiple columns of data, click **Edit**.
 - b. Verify that you are splitting based on position numbers, which means that column splits are done based on the number of characters from the left side of each line.
 - c. Your recipe step might look similar to the following:

Transformation Name	Split columns by positions
Parameter: Column to split	column1
Parameter: Option	By positions
Parameter: Positions	7, 67, 117, 167, 217, 221, 239, 251, 253, 303, 315, 317, 329, 341, 391, 400, 512, 560, 610, 630, 650, 660

- d. In the list of values for positions, insert a new position number for the column or columns that contain multiple columns of data.
 - e. Verify your changes in the Transform Preview panel.
9. Click **Add**.
 10. Verify that the columns are split correctly.
 11. You can use the following step to remove the whitespace from each cell value.

Transformation Name	Edit column with formula
Parameter: Column	*
Parameter: Formula	TRIM(\$col)

12. Click **Add**.

Generate a Sample

Contents:

- *When to Take a New Sample*
 - *Change Sample Size*
 - *Limitations*
 - *Collect a New Sample*
 - *Example - Random sample*
 - *Example - Filter-based sample*
 - *Example - Anomaly-based sample*
 - *Cancel Sample*
 - *Load Sample*
 - *Delete Sample*
 - *Invalid Samples*
 - *Collected Samples*
 - *Review Sample Jobs*
 - *Best Practices*
-

When you transform your data in the Transformer page, you are performing these transformations on a sample of the total dataset. As needed, you can generate new samples using a variety of algorithms to acquire other slices of your data.

The initial data sample is collected from the initial rows of the dataset. Whenever you create a recipe and open the dataset in the Transformer page, Trifacta application automatically generates the initial sample.

- By default, the initial sample is the first 10 MB of your dataset.
 - The size of the sample can be modified by an administrator.
 - For file-based sources, the initial sample is taken from a limited number of files.
 - By default, this limit is set to 50 files.
 - The maximum number of files from which a sample can be generated can be defined by an administrator.
- If your dataset is less than 10 MB, then the entire dataset may be loaded as an initial sample.
- For datasets larger than 10 MB, the first 10MB of rows are loaded into the Transformer page.

Tip: On the Transformer page, this first sample is listed as **Initial Data**. For more information on how this special sampling type is generated, see *Overview of Sampling*.

When to Take a New Sample

The initial sample allows you to get started immediately building your recipe steps. However, your recipe and dataset may require additional samples. For example:

- If you have a very long dataset with many rows, there may be statistically significant values that are not part of the first 10MB of data. The recipe steps that you create may not affect those rows properly, since you have not seen any data from them.
- If you have a very wide dataset with many columns, you may need to take additional filter-based samples to focus on the separate segments of your data. For example, if your dataset contains mismatched or missing values, you may consider taking an Anomaly-based sample that can look for mismatched, or missing, or both values in your dataset.
- As you add steps in your recipe, the current state of the Transformer page is rendered based on the currently valid sample (initial sample, in this case) plus all of the recipe steps between the step where the sample was taken and your current step. All of these steps must be rendered in the browser. As you add more recipe steps without taking a sample, browser performance is affected.

Tip: You should utilize sampling as much as possible to improve the browser performance and to get good coverage of the samples across recipes.

NOTE: Generation of a new sample is executed as a job. Quick scan jobs are executed through Trifacta Photon on the Trifacta node, while Full scan jobs are executed on an available clustered running environment. Depending on your deployment, there may be costs associated with generating a sample.

You can generate a new sample when:

- You are working with complex and wide datasets.
- You have complex flows.
- Your dataset has a bad data or outliers that may require a different sample.
- You have datasets with more than 10 MB of data.
- You have added one or more multi-dataset operations with steps, such as a join, union, pivot, or lookup.

Change Sample Size

If you are encountering low-memory conditions related to sampling or wish to improve the performance of the sampling process, you can adjust the size of the samples that are displayed in the browser for your current recipe. For more information, see *Change Recipe Sample Size*.

Limitations

- Advanced sampling options are available only with a full scan of the dataset.
- Undo/redo do not change the sample state, even if the sample becomes invalid.
- When a new sample is generated, sort transformations are not preserved for some type of outputs. Sort transformations must be reapplied.
- When executed on the Trifacta Photon running environment, samples taken from a dataset with parameters are limited to a maximum of 50 files.

Collect a New Sample

You can use the existing loaded sample, or you can collect a new sample to use.

Steps:

1. In the Transformer page, click the Eyedropper icon at the top of the page.
2. From the Samples panel, select the required type of sample. For more information, see *Sample Types*.
3. In the Collect new sample panel, select either Quick or Full scan.
 - a. **Quick:** Creates a sample by partial scanning of the dataset and yields quicker results.

Tip: Quick scan samples are executed by default in the Trifacta Photon running environment. If that environment is not available, the Trifacta application may attempt to run the Quick Scan job on an available clustered running environment.

- b. **Full:** Creates a sample by scanning the full dataset. This method takes a longer time depending on the size of the dataset.

Tip: Full scan samples are executed in the cluster running environment.

4. Click **Collect** to collect the sample. A sample job ID is generated for each sample you collect. When the sample is available, the Load Sample message is displayed in the Transformer page.
5. To load the sample, click **Load Sample**.

Example - Random sample

Random samples can be generated from a quick or full scan of your dataset.

Tip: A random sample is a fast way to get another randomized slice of your dataset. Often, this can be a first sample to generate after loading a new dataset into the Transformer page.

Steps:

1. In the Transformer page, click the Eyedropper icon at the top of the page.
2. From the Samples panel, select Filter-based sample.
3. In the Collect new sample panel, select the type of scan: Quick or Full.
4. Click **Collect**.
5. When sample collection is complete, a confirmation message is displayed. Click **Load sample**.
6. The random sample is loaded into the Transformer page.

Example - Filter-based sample

The Filter-based sample is helpful when you want to filter the data based on specific values or formulas. The following example filters the required values in the `Region` column for calculating discounts, and then generates a random sample from the matching rows only. For example, you may have a dataset with many values for `Region` such as Atlantic, North East, West Coast and want to calculate discounts only for North East region, you can collect a Filter-based sample.

Steps:

1. In the Transformer page, click the Eyedropper icon at the top of the page.
2. From the Samples panel, select Filter-based sample.
3. In the Collect new sample panel, enter the following details:
 - a. From the Scan column, select **Quick**. For more information, see "Collect a New Sample" above.
 - b. In the Filter field, enter `Region == 'North East'`.
4. Click **Collect**. A confirmation message is displayed.
5. Click **Load sample**. The Filter-based sample is loaded with only the `North East` values for the `Region` column.

Example - Anomaly-based sample

If your dataset has missing values or mismatched values, you can use Anomaly-based sample type to filter the missing values. The following example is based on the missing values in a `Discount` column. When you apply the Anomaly-based sample, the sample displays only rows that have missing values for the `Discount` column.

Steps:

1. In the Transformer page, click the Eyedropper icon at the top of the page.
2. From the Samples panel, select Anomaly-based sample.
3. In the Collect new sample panel, enter the following details:
 - a. From the Scan column, select **Quick**. For more information, see "Collect a New Sample" above.
 - b. Select the required column: `Discount`.
 - c. From the anomaly type, select **Find missing values only**.
4. Click **Collect**. A confirmation message is displayed.
5. Click **Load sample**. The Anomaly-based sample is loaded with the missing values for the `Discount` column.

Cancel Sample

To cancel a sample collection, click the X next to the progress bar. The interrupted sample is listed as unavailable in the Collected samples panel.

Load Sample

You can create as many samples as required based on your dataset. All collected samples are available in the Collected samples panels, where you can review and load them as required.

Steps:

1. In the Samples panel, click **See all collected samples**.
2. From the Collected samples panel, select the required sample from the Available tab. For more information, see "Collected Samples" below.

NOTE: Samples listed under the Unavailable tab are invalid for the current state of your recipe. You cannot select these samples for use.

3. If you want to edit the sample name, click the Pencil icon against the sample.

Delete Sample

After you have created a sample, you cannot delete it through the application.

NOTE: Trifacta does not support deletion of samples after they have been created. For more information, contact your IT administrator.

Invalid Samples

NOTE: Samples are valid based on the state of your flow and recipe at the step where the sample was collected.

Whenever you add or modify a step to the recipe, Trifacta verifies if the current sample is valid. The current sample can become invalid if you add a new step before the step where the sample was created. For example, if you have created a sample in 30th step and if you add a new step that breaks the sample before the 30th step, then the sample becomes invalid.

After the sample becomes invalid, the Transformer page reverts to the recently collected sample that is valid.

NOTE: If the sample is reverted to an earlier sample, then more steps between when that sample was generated and your current location in the recipe are generated in the browser's memory. Browser performance may be impacted.

NOTE: If you modify a SQL statement for an imported dataset, any samples based on the old SQL statement are invalidated.

Collected Samples

The collected samples store the details of your samples collected for your dataset. In the Samples panel, click **See all collected samples** link.

Collected samples	
Available	Unavailable All
Initial 3,430 rows Collected 02/15/2022 by solson+s4@trifacta.com	Loaded
Random Full scan • New • Job 3469759 Collected 02/15/2022 by solson+s4@trifacta.com	Load
Anomaly-based On POS_Qty • New • Job 3556014 Collected Today at 2:33 PM by anagarajan@trifacta.com	Load

Figure: Collected samples

The collected samples contain the following tabs:

- **Available:** Displays the available samples that can be used. You can click **Load** to load the required sample.
- **Unavailable:** Displays the invalid samples, which cannot be selected for use. If subsequent steps make a sample valid again, it is moved to the **Available** tab.
- **All:** Displays both the available and unavailable samples.

You can click the sample name to view the sample details.

Sample details	
Anomaly-based	Load Rename
Rows	New
Job ID	3556014
Method	Anomaly-based
Collected	Today at 2:33 PM
Collected by	anagarajan@trifacta.com
Scan	Quick
Includes	Both missing and invalid values
Columns	On POS_Qty

Figure: Sample details

- **Load:** Click **Load** to load the sample.
- **Rename:** Click **Rename** to rename the sample

Review Sample Jobs

You can review and manage all of your samples like transformation jobs. For more information, see *Sample Jobs Page*.

Best Practices

For more information on best practices, troubleshooting, and browser crashes, see <https://community.trifacta.com/s/article/Best-Practices-Managing-Samples-in-Complex-Flows>.

Change Recipe Sample Size

By default, samples displayed in the Trifacta® application can be up to 10 MB in size. In some cases, you may want to increase or decrease the size of samples displayed in the browser to include more data or to prevent browser or Trifacta Photon memory issues.

NOTE: Samples are still generated using the preset limit and stored in full on the base storage layer. This setting changes the volume of the data delivered to the browser.

NOTE: Trifacta administrators can increase the maximum size of the Trifacta Photon samples up to 40 MB. For more information, see *Configure Application Limits*.

This size reflects the maximum permitted size of a sample that is delivered to the browser. If the available data is less than the maximum permitted size, then the actual sample size may be smaller.

- When this setting is reduced, the sample currently loaded in the browser is immediately reduced to the new maximum size. Other available samples for the recipe are resized when they are reloaded.
- If you raise the maximum sample size, the volume of the currently loaded sample can be expanded to the maximum size, if the data is available in the sample.

NOTE: Maximum sample sizes are configured on a per-recipe basis. Changes to your sample size affect sample sizes in downstream recipes. For example, if the maximum sample size on Recipe A is set to 5MB, any sample from Recipe A that is used in Recipe B, which is downstream of Recipe A, is also constrained to 5MB in maximum size. However, Recipe B may have a different maximum sample size, so you can generate a new sample in Recipe B to acquire a different-sized sample.

Steps:

1. In the Transformer page, click the name of the sample in the top menu.
2. The Sample Indicator is displayed
3. In the Sample Indicator, click **Edit**.
4. Use the slider bar to change the maximum size of samples delivered to the browser for this recipe:

Set sample size

Decrease the sample size when you need to alleviate low memory situations and improve application performance.

Increase the sample size when you need more data to work with, but please consider that larger sample sizes use more browser memory and may slow performance.

20MB (43,071 rows)

1MB

20MB

40MB

Please consider that large samples may slow performance

Learn More

Cancel

Save

Figure: Set sample size

NOTE: The range of the slider indicates that maximum available data for the sample. For example, if the sample is 7 MB then the slider shows a maximum of 7 MB.

Copyright © 2022 Trifacta Inc.

Page #116

NOTE: A warning message may displayed if the sample size exceeds the recommended size.

NOTE: For datasources that are uncompressed or converted when ingested to the backend, the actual storage size may exceed the specified maximum limit.

Tip: The Trifacta application provides a recommendation for the new sample size. You should set your sample size to this value or a smaller value.

5. To apply your changes, click **Save**.
6. The current sample is immediately updated to reflect the new maximum sample size. The size of all subsequent samples that are delivered to the browser for this recipe are capped at this new maximum size. Samples are still generated at the preset size in backend storage.

For more information, see *Sample Indicator*.

Validation Tasks

You can detect issues in your data or validate it against source or target schemas.

Profile Your Source Data

You might want to execute a profile of the data that you imported from the source. As soon as you create a recipe from a source, you can execute a job to profile the dataset.

By profiling the data as soon as you load it into the Transformer page, you can assess the following:

- Identify problems in the source and potentially correct them in the source system.
- Create a baseline to evaluate the data wrangling work you do in Trifacta®.
- Identify mismatched or missing values.

Tip: You can also use this technique to generate an output of your source data, which is useful if you do not have read access to the source outside of Trifacta.

Steps:

1. In the Import Data page, create an imported dataset from your source. Add it to a flow.
2. In Flow View, create a recipe for your imported dataset.
3. In Flow View, edit the newly created recipe. It is opened in the Transformer page.
4. If needed, add a header step to your dataset.
5. Click **Run**.
6. In the Run Job page, select the following options:
 - a. If you have the option of selecting a running environment, select the default one. This option may not be available in your product.
 - b. CSV format (you need at least one format to generate your dataset's profile).
 - c. Select to profile results.
7. Click **Run**.
8. When the results are generated, click the Profile tab in the Job Details page.
9. A profile of your dataset is displayed.

In the generated profile, you can identify:

- Missing or mismatched values in each column
- Statistical break-out by quartile
- Beginning dataset size and baseline job execution speed

Tip: You can download the profile and output for review.

For more information, see *Job Details Page*.

Preserve Source Visual Profile

If you wish to preserve the capability of running a profile or gathering results from your source, you can do the following:

1. In Flow View, select the recipe that was used to create the source profile.
2. Rename this recipe to something like, `SourceData`.
3. Create an output off of this recipe. Run the job to create the visual profile.
4. Select the recipe again. Now, click **Add New Recipe**.
5. Edit this new recipe and build out your transformation steps.
6. Whenever you need to regenerate the profile for the source, select the `SourceData` recipe and select the output from it. Then, run a job for it.

Tip: This technique is useful if you are replacing the source dataset with refreshed data on a periodic basis.



Validate Your Data

Contents:

- *Before You Begin*
 - *Verify downstream requirements*
 - *Identify important fields*
 - *Profile your source data*
 - *Generate a new random sample*
 - *Transformations vs. Data Quality Rules*
 - *Validate Consistency*
 - *Mismatched values*
 - *Outlying values*
 - *Data range checks*
 - *Duplicate rows*
 - *Uniqueness checks*
 - *Permitted character checks*
 - *Validate Completeness*
 - *Missing values*
 - *Null values*
 - *Validate data against other data*
 - *After Transformation*
 - *Generate output profile*
 - *Decisions*
-

The process of cleansing, enhancing, and transforming your data can introduce significant changes to it, some of which might not be intended. This page provides some tips and techniques for validating your dataset, from start to finish for your data wrangling efforts.

Data validation can be broken down into the following categories:

- **Consistency** - Does your data fit into expected values for it? Do field values match the data type for the column? Are values within acceptable ranges? Are rows unique? Duplicated?
- **Completeness** - Are all expected values included in your data? Are some fields missing values? Are there expected values that are not present in the dataset?

Before You Begin

Before you begin building your data pipeline, you should identify your standards for data quality.

NOTE: Depending on your source system, you might be able to generate data quality reports from within it. These reports can be used as the basis for validating your work in Trifacta®.

If your source system does not enable generation of these reports, you should consider profiling your dataset as soon as you load your data into Trifacta.

Verify downstream requirements

Before you begin modifying your dataset, you should review the columns and ranges of values in those columns that are expected by the downstream consumer of your dataset. A quick review can provide guidance to identify the key areas of your dataset that require end-to-end validation.

Identify important fields

For datasets with many columns, it might be problematic to apply consistent validation across all columns. In these situations, you might need to decide the columns whose consistency, completeness, and accuracy are most important.

Profile your source data

Before you get started building your recipe on your dataset, it might be a good idea to create a visual profile of your source data. This process involves creating a minimal recipe on a dataset after you have loaded into the Transformer page. Then, you run a job to generate a profile of the data, which can be used as a baseline for validating the data and as an assistant in debugging the origin of any data problems you discover.

Visual profiling also generates statistics on the values in each column in the dataset. You can use this statistical information to assess overall data quality of the source data. This visual profile information is part of the record for the job, which remains in the system after execution.

Generate a new random sample

When a dataset is first loaded into the Transformer, the default sampling collects the first N rows of data, depending on the size and density of each row. However, your dataset might contain variations in the data that are not present in this first sample. New samples can be generated through the Samples panel.

Transformations vs. Data Quality Rules

You can perform data quality rules through the following general methods:

1. **Transformations:** You can verify the quality of your data by creating transformations to check values for consistency and completeness and, if needed, taking action on the data itself for deviations.
 - a. Transformations are built in the Transformer page to add steps to your recipe.

Tip: If you need to take actions in the data itself based on data quality checks, it may be better to use a transformation.

2. **Data quality rules:** You can create data quality rules, which are persistent checks of columnar data against rules that you define. You can perform a variety of checks that exist outside of the recipe, so as you transform your data, the data quality rules automatically show the effects of your transformations on the overall quality of your data.
 - a. Data quality rules are not recipe steps. They exist outside of recipes and persist in the Transformer page to help you to build steps to transform your data.
 - b. Data quality rules are built in the Data Quality Rules panel in the Transformer page.
 - c. For more information, see *Overview of Data Quality*.

Tip: If you are attempting to transform the data to get all values in a column to pass one or more data quality checks, use data quality rules.

Examples of both types of data quality checks are provided below.

Validate Consistency

Trifacta provides useful features for checking that your data is consistent across its rows. With a few recipe steps, you can create custom validation checks to verify values.

Mismatched values

In the data quality bar at the top of a column, you can review the valid (green), mismatched (red), and missing (gray) values.

When you click the red bar:

- The rows that contain mismatched values are highlighted in the data grid.
- The application provides suggestions in the form of suggestion cards for ways that you can transform your data.

Transformation:

Maybe you are unsure of what to do with your data. If you would like to examine all of the rows together, you can insert a transformation like the following in your recipe.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	ismismatched(Primary_Website_or_URL, ['Url'])
Parameter: New column name	mismatched_Primary_Website_or_URL

The above checks the values in the `Primary_Website_or_URL` column against the `Url` data type. If the value in the source column is not a valid URL, then the new column value is `true`.

Data quality rule:

The following data quality rule checks the `Primary_Website_or_URL` column against the `Url` data type:

Data Quality Rule	Valid
Parameter: Column	Primary_Website_or_URL
Parameter: Data type	'Url'

Outlying values

Through the Column Details panel, you can review statistical information about individual columns. To open, select **Column Details...** from a column's drop-down menu.

In the Summary area, you can review the count of Outlier values. In Trifacta, an **outlier** is defined as any value that is more than 4 standard deviations from the mean for the set of column values.

The Column Details panel also contains:

- Counts of valid, unique, mismatched, and missing values.
- Breakdowns by quartile and information on maximum, minimum, and mean values.

Available statistics depend on the data type for the column.

Data range checks

Standard deviation ranges

For example, your range of values does not match the application's definition of an outlier, and you need to identify values that are more than 5 standard deviations from the mean.

You can create your custom transforms to evaluate standard deviations from mean for a specific column. For more information, see *Locate Outliers*.

Fixed value ranges

Transformation:

If you need to test a column of values compared to two fixed values, you can use the following transformation. This one tests evaluates a column value. If the value in `Rating` column is less than 10 or greater than 90, then the generated column value is `true`.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	((Rating < 10) (Rating > 90))
Parameter: New column name	Outlier_Rating

Data quality rule:

The following data quality rule performs the same evaluation as the previous transformation yet persists in the Transformer page.

Data Quality Rule	Formula
Parameter: Formula	((Rating < 10) (Rating > 90))
Parameter: Group rows by	(empty)

Duplicate rows

Entire rows can be tested for duplication. The `deduplicate` transform allows you to remove identical rows. Note that whitespace and case differences are evaluated as different rows.

Uniqueness checks

For an individual column, the Column Details panel contains an indicator of the number of unique values in the column. If this value does not match the count of values and the count of rows in the sample, then some values are duplicated. Remember that these counts apply to just the sample in the Transformer page and may not be consistent measures across the entire dataset.

You can perform ad-hoc tests for uniqueness of individual values.

Data quality rule:

The following data quality rule verifies that all of the values in the `custId` column are unique:

Data Quality Rule	Unique
-------------------	--------

Parameter: Column	custId
-------------------	--------

Permitted character checks

You can test for the presence of permitted characters in individual columns by using a regular expression test.

Transformation:

The following transformation evaluates to `true` if all of the characters in a column field are alphanumeric or the space character:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>MATCHES(MarketName, /^[a-zA-Z0-9]*\$ /)</code>

You can add additional permitted characters inside the square brackets. For more information, see *Text Matching*.

Data quality rule:

This data quality rule performs the same test as the above transformation:

Data Quality Rule	Match
Parameter: Column	MarketName
Parameter: Matches pattern	<code>/^[a-zA-Z0-9]*\$ /</code>

Validate Completeness

Trifacta provides easy methods for identifying if cells are missing values or contain null values. You can also create lookups to identify if values are not represented in your dataset.

Missing values

At the top of each column, the data quality bar includes a gray bar indicating the number of cells in the column that do not contain values. This set of values includes missing values.

Click the gray bar to prompt for a set of suggestion cards for handling those values.

Null values

While null values are categorized with missing values, they are not the same thing. In some cases, it might be important to distinguish the actual null values within your dataset, and several Wrangle can assist in finding them.

Validate data against other data

You can also test if your dataset contains at least one instance of a set of values.

For example, your dataset contains businesses throughout the United States. You might want to check to see if each state is represented in your dataset.

Steps:

1. Create a reference dataset that contains a single instance of each item you are checking. In this example, it'd be a simple CSV file with the name of each state on a separate line.

Tip: To your second dataset, you might want to add a second column containing the value `true`, which allows you to keep separate validation data from the columns that you join.

2. Add this CSV file as a new dataset to your flow.
3. Open your source dataset. In the Search panel, enter `join datasets`.
4. In the Join window:
 - a. Select the reference dataset you just created. Click **Accept**. Click **Next**.
 - b. Select the type of join to perform:
 - i. **Right outer join:** Select this join type if you want to delete rows in your source dataset that do not have a key value in the reference dataset. In the example, all rows that do not have a value in the State column would be removed from the generated dataset.
 - ii. **Full outer join:** Select this type to preserve all data, including the rows in the source that do not contain key values.
 - c. Select the two fields that you want to use to join. In the example, you would select the two fields that identify state values. Click **Next**.
 - d. Select the fields that you want to include in the final dataset. Click **Review**.
 - e. Click **Add to Recipe**.
5. The generated dataset includes all of the fields you specified.
6. For one of your key values, click the gray bar and select the link for the number of affected rows, which loads them into the data grid. Review the missing values in each key column.
7. To remove these rows, select the missing value category in the data quality bar for the appropriate column and apply a delete statement.
8. The generated command should look like the following:

Transformation Name	Delete rows
Parameter: Condition	ISMISSING([State])

For a detailed example, see *Validate Column Values against a Dataset*.

After Transformation

Generate output profile

After you have completed your recipe, you should generate a profile with your executed job. You can open this profile and the profile you created for the source data in separate browser tabs to evaluate how consistent and complete your data remains from beginning to end of the wrangling process.

NOTE: The statistical information in the generated profile should be compared to the statistics generated from the source, so that you can identify if your changes have introduced unwanted changes to these values.

Decisions

After you have performed your data validation checks, you might need to make some decisions about how to address any issues you might have encountered:

- Some problems in the data might have been generated in the source system. If you plan to use additional sources from this system, you should try to get these issues corrected in the source and, if necessary, have your source data regenerated.

- Some data quality issues can be ignored. For the sake of downstream consumers of the data, you might want to annotate your dataset with information about possible issues. Be sure to inform consumers on how to identify this information.

Validate Column Values against a Dataset

Contents:

- *Example*
- *Prepare Validation Dataset*
- *Import Validation Dataset*
- *Join with Validation Dataset*
- *Triage Invalid Data*
 - *Insert error messages*
 - *Delete invalid rows*
 - *Update validation dataset with new values*
 - *Standardize invalid data*

When needed, you can validate a column of values against a pre-defined set of values maintained in a separate dataset. This method of data validation is most useful for String-based data that does not easily map to a specific pattern of values.

Tip: This method is a suitable replacement for custom data types maintained using a dictionary file.

Overview

This method of validation is completed through the following general steps:

1. **Prepare your validation dataset.** Create a dataset containing the unique values against which you wish to validate.
2. **Import your validation dataset.** After you have prepared the dataset externally, you should import it into the Trifacta application.
3. **Join your data to your validation dataset.** You perform a join from the dataset you're wrangling to the validation dataset that you imported. Validation errors should be identifiable as missing values in the validation column.
4. **Triage defects as necessary.** For rows that cannot be resolved, additional wrangling may be necessary.
5. **Standardize data.** You can use the Standardize tool to review the differences between invalid data and valid data.

Example

This approach is best demonstrated by example. Below, you can see a set of orders for product.

productName	customerName	Qty	totalSales
Product ADA	Customer ABC	2	26
Product AEV	Customer DEF	4	100
Product DXL	Customer EFG	6	42
Product EDM	Customer ABC	1	26
Product JTO	Customer DEF	3	75
Product JUB	Customer EFG	5	35
Product NRS	Customer ABC	6	26
product NSE	Customer DEF	8	200
Product ZZZ	Customer EFG	10	80

Notes:

- You can see that this set of orders is spread across 10 different products for three different customers.
- In the productName column, there is a mismatch in capitalization.
- The final productName value (Product ZZZ) does not exist.

The product names in this list must be validated against a dataset containing the list of all available products. This list is 100 product names long.

You can use the links below to download these datasets as CSV files for exploration in your project or workspace.

- [Dataset-ProductNames.csv](#)
- [Dataset-ProductNames-Orders.csv](#)

Prepare Validation Dataset

If you haven't done so already, you should prepare your validation dataset for use in the Trifacta application. Below, you can see the first 10 rows of the ProductNames dataset:

productName
Product ADA
Product AEV
Product ANH
Product ARA
Product ARM
Product AUJ
Product BAD
Product BAP
Product BEI
Product BEZ

Notes:

- A column header is provided in the dataset. This is helpful for identifying the column to use later as the join key.
- You may wish to enter a validation column, simply contains the value `TRUE`.

Tip: If your dataset does not contain this column, you can create a new formula within the Transform Builder to insert this value. This step is covered later.

Import Validation Dataset

If you have prepared your dataset, you must import into in the application.

Steps:

1. In the Trifacta application, click **Library**.
2. Click **Import Data**.
3. Navigate to the file or files to import.

Tip: If you are using the example files, you can right-click them above, download them to your desktop, and then drag and drop them into the Import Data page.

4. Import the file as a new dataset.

Tip: If you are using the example datasets, you can call it `Reference-ProductNames`.

5. It may be helpful to import the file into a new file and create a recipe from it.

For the example dataset, there is a single column of values. To make this dataset useful as a validation dataset, add the following transformation, which adds a second column called `validation` containing the value `true` for each row.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	'true'
Parameter: New column name	validation

The example reference dataset now looks like the following:

productName	validation
Product ADA	TRUE
Product AEV	TRUE
Product ANH	TRUE
Product ARA	TRUE
Product ARM	TRUE
Product AUJ	TRUE
Product BAD	TRUE
Product BAP	TRUE
Product BEI	TRUE
Product BEZ	TRUE

Join with Validation Dataset

Now you can join your existing dataset with the new validation dataset. A **join** performs a comparison of the column values in one dataset compared to the column values in another dataset. Where matches are detected, columns and values of the joined-in dataset are inserted into the source dataset. For more information on joins, see *Join Types*.

Steps:

1. Edit the recipe of the source dataset.
2. As a new step for the recipe, enter `join datasets` in the Search panel.
3. Select the source of the joined-in data:
 - a. If you created a recipe and added steps (as in the example), then click the Recipes in current flow tab.
 - b. If you imported a clean dataset, then click one of the Datasets tabs.

- For Join type, select **Left**.

Tip: A left join includes all rows from the left (source) dataset and only the matching rows from the right dataset for a specified set of column values (**join keys**) in the left dataset. If a column value in the left dataset does not exist in the right dataset, then null values are listed for that row's entry for all columns imported from the right dataset.

- For the Join keys, select the column containing values to check from the left (source) dataset and then column containing the reference values in the right dataset.

Tip: In the example datasets, both of these columns are called `ProductNames`, which assists the join tool in identifying the join key columns.

- Under Join Keys, hover over one of the column names. Then, click the Pencil icon.

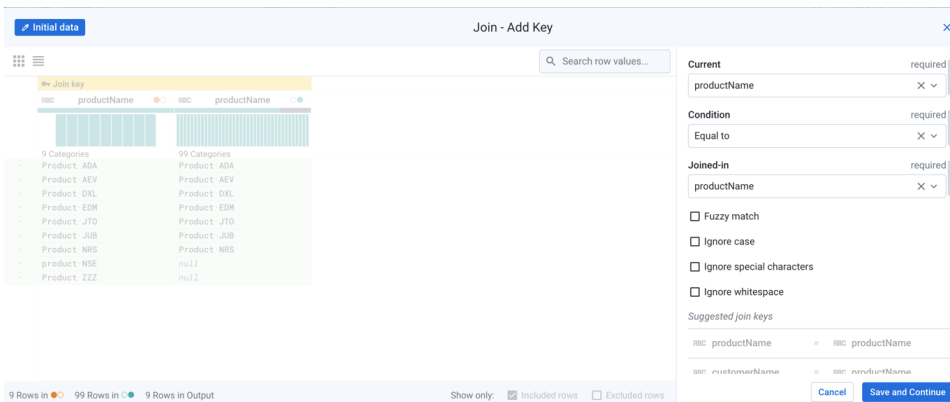


Figure: Edit join keys

- When you edit the join keys, you can specify the Condition, which defines the type of comparison that is performed to determine a match.
- The other options allow you to fine-tune how matching is performed. In particular, the **Ignore case** option is off, which means that by default, joins are case-sensitive. So, `product 01` does not match to `Product 01`.

Tip: In the example data, you can see that the `product NSE` entry does not have a match, which is due to differences in case. If `Ignore case` is enabled, then this entry may find a match. However, you may wish to maintain case-sensitive searches to ensure that you can clean up the data correctly.

- Click **Save and continue**.
- Click **Next**.

11. Select all columns, and click **Review**.

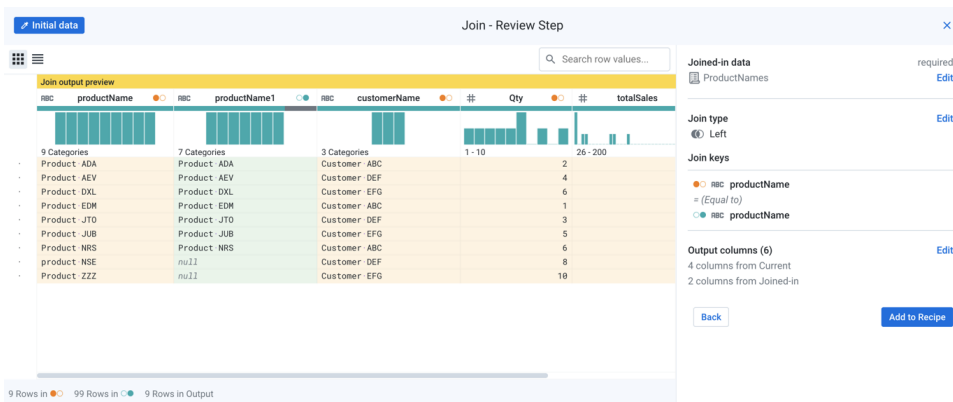


Figure: Review join

Tip: In the example, you can see that two rows failed to match.

12. Click **Add to recipe**.

Triage Invalid Data

You should now have a dataset containing all columns from both datasets.

Tip: In the example dataset, a second column called `validation` was added. This column contains null values for the mismatched rows. So, you can delete the duplicate `productName` column, which contains null values in two rows.

You can make some decisions on how to triage the invalid values in the `ProductNames` column.

Insert error messages

You can use a transformation to replace the null values in the `validation` column with a meaningful message:

Transformation Name	Edit column with formula
Parameter: Columns	validation
Parameter: Formula	IF (\$col == NULL(), 'Error - invalid Product Name', 'ok')

The transformation replaces the null values with `Error - invalid Product Name` and writes `ok` for the other rows.

Delete invalid rows

If the entire row of data is invalid because of the invalid value, then you simply delete the row. This transformation deletes rows, where the `validation` column contains a null value:

Transformation Name	Filter rows
Parameter: Condition	Custom formula

Parameter: Type of formula	Custom single
Parameter: Condition	(validation == NULL())
Parameter: Action	Delete matching rows

Update validation dataset with new values

If you discover that some of the mismatched rows should be part of your validation dataset, you can follow these general steps to add them.

Steps:

1. In Flow View, add a new recipe off of the recipe where the join occurs.
2. Edit this new recipe.
3. Insert this transformation to filter the dataset down to the rows containing the new values:

Transformation Name	Filter rows
Parameter: Condition	Custom formula
Parameter: Type of formula	Custom single
Parameter: Condition	(validation == NULL())
Parameter: Action	Keep matching rows

4. You can then delete all columns except (in the example) the `ProductName` column. From the column menu of the `ProductName` column, select **Delete others**.
5. You should now have a single column containing the missing values. You can create a union between this dataset and the validation dataset to add the values. See *Append Datasets*.
6. Run a job on this recipe to generate the output file.
7. This output file can then be used to replace the source data. In Flow View, select the imported dataset. From the context panel, open the More menu and select **Replace**.

Standardize invalid data

You can standardize values in join key column for your source data, which may address some of the invalid data. The Standardization tool attempts to cluster similar values together within the column, so that you can try to identify if your values in your source dataset can be matched to values in the target dataset.

Tip: After you have identified issues, this step may be best to apply before the join. By applying it before the join, some or all of the mismatched value issues may be addressed.

For your source column, you can select **Standardize** from the column menu.

In the example, the `ProductNames` column values are being standardized. In this case, the lower case `product NSE` value has been corrected to be `Product NSE`.

Clustering options

Row count ▾	Source value	New value
		9 unclustered values · 9 rows
<input checked="" type="checkbox"/>	1 product : NSE	Product : NSE
<input type="checkbox"/>	1 Product : ADA	Product : ADA
<input type="checkbox"/>	1 Product : AEV	Product : AEV
<input type="checkbox"/>	1 Product : DXL	Product : DXL
<input type="checkbox"/>	1 Product : EDM	Product : EDM
<input type="checkbox"/>	1 Product : JTO	Product : JTO
<input type="checkbox"/>	1 Product : JUB	Product : JUB
<input type="checkbox"/>	1 Product : NRS	Product : NRS
<input type="checkbox"/>	1 Product : ZZZ	Product : ZZZ

9 unique source values 9 rows
1 selected (1 rows)

Standardize

New value

Product NSE

Source value
product : NSE

Row count
1

[Revert to source ↶](#)

Apply

Summary

Source column	productName
Unique new values	9
Source values updated	1 / 9 (11.11%)
Rows updated	1 / 9 (11.11%)

Cancel

Save to Recipe

Figure: Standardize join key values

For more information, see *Overview of Cluster Clean*.

Find Bad Data

Contents:

- *Locate mismatched values*
- *Methods for fixing mismatched data*
 - *Mismatched values in transform code*
- *Trim data*
- *Set values using other columns*
- *Use functions to fix mismatched values*
- *Bad data typing*

You might encounter problems with how data has been structured or formatted that you must fix prior to providing the content to your target system. You can use the methods in this section to locate problems with the content or data typing of your data.

Locate mismatched values

When Trifacta® evaluates a dataset sample, it interprets the values in a column against its expectations for the values. Based on the column's specified data type and internal pattern matching, values are categorized as valid, mismatched, or missing. These value categories are represented in a slender bar at the top of each column.

- A **mismatched value** is any value that seems to be of a different data type than the type specified for the column. For example, if the value `San Francisco` appears in a column of Zip Code type, it would be marked as a mismatched value.

In the data quality bar, mismatched values are identified in red:

Tip: Before you start performing transformations on your data based on mismatched values, you should verify the data type for these columns to ensure that they are correct. The type against which values are checked is displayed to the upper left of the data quality bar. Below, the data type is `ZIP` for U.S. Zip code data.



Figure: Mismatched values in red

Mismatched values can be sourced from a variety of issues:

- Values may be miskeyed into the source system.
- The source system may introduce errors in output, particularly if the data is generated for export using a customized structure.
- Incorrect use of column delimiters may create offsets among fields in individual rows.
- Data may be badly structured across a set of rows.
- The column may be assigned the wrong data type.

Tip: When cleaning up bad data, you should look to work from bigger problems to smaller problems. If a higher percentage of a column's values have been categorized as mismatched data, it may indicate a wider problem with the data. In affected rows, verify if other columns' values are also mismatched. These rows should be reviewed and fixed first. When fixed, other mismatches may be fixed in other rows, too.

To locate data:

NOTE: Remember that you are working on a sample of your data. For small datasets, the Initial Data sample includes all rows of the dataset and is unsampled.

- From the Transformer page, click the mismatched values in a column's data quality bar to see their count, highlight them in the rows of the data grid, and trigger a set of suggestions for your review.
- To refine the data grid view, click the Show Only Affected Rows checkbox in the status bar at the bottom of the screen. Only the rows that are affected by the previewed transform are displayed.

Tip: This step highlights specific values that are mismatched. You can take note of individual values.

- To locate a specific value, click the Filters icon on the right side of the screen. In the Rows tab, enter the specific value to locate. Rows containing this value are highlighted. Back in the data grid, you can select one of these highlighted values to be prompted for suggestions.

Methods for fixing mismatched data

When you discover mismatched data in your dataset, you have the following basic methods of fixing it:

1. **Change the data type.** If the percentage of mismatched rows is significant, you may need to change the data type for a better match.
2. **Replace the values with constant values.** This method works if it is clear to you that the values should be a single, consistent value. Select the mismatched values in the column, and then select one of the highlighted mismatched values. Use the `replace` transform to change the mismatched values to corrected values.

Tip: One easy way to fix isolated problems with mismatched values is to highlight a mismatched value in the data grid. A new set of suggestions is displayed. You can select the `replace` suggestion and then modify it to include the replacement value.

3. **Set values with other columns' values.** You can use the `set` transform to fix mismatched values by replacing them with the corresponding values from other columns.
4. **Use functions.** Data can be fixed by using a function in conjunction with the `set` transform to replace mismatched values.
5. **Delete rows.** Select the mismatched values and use the `delete` transform to remove the problematic rows.
6. **Hide the column for now.** You can remove the column from display if you want to focus on other things. Select **Hide** from the column drop-down. Note that hidden columns appear in any generated output.
7. **Delete the column.** If the column data is unnecessary or otherwise unusable, you can delete the entire column from your dataset. Select **Delete** from the column drop-down.

Tip: Delete unnecessary columns as early as possible. Less data is easier to work with in the application and improves job execution performance.

NOTE: You might need to review and fixed mismatched data problems multiple times in your dataset. For example, if you unnest the data, additional mismatches might be discovered. Similarly, joins and lookups can reveal mismatches in data typing.

Mismatched values in transform code

In your transforms, mismatched data can be identified references as in the following:

Transformation Name	Edit column with formula
Parameter: Columns	postal_code
Parameter: Formula	IF(ISMISMATCHED(postal_code, ['Zipcode']), '00000', postal_code)

Note that the single quotes are important around the value, which identifies the value as a constant.

Tip: In the above, note that the value `Zipcode` identifies the data type that is used for matching the column values. In this case, for greater specificity, you might want to identify the mismatched values in the column against the data type `Integer`, since all U.S. postal codes are positive integers. For more information on how to explicitly reference data types in your steps, see *Valid Data Type Strings*.

Trim data

To trim whitespace out of a column, use the following transformation:

Transformation Name	Edit column with formula
Parameter: Columns	column1
Parameter: Formula	TRIM(\$col)

The `$col` token is a reference to the column name to which the formula is being applied. For more information, see *Source Metadata References*.

This step may increase the number of missing values (for values that contain only whitespace characters) and the number of instances of matching values (for values that have spaces before and after an alphanumeric value).

You can modify the above transformation to trim leading and trailing spaces across all columns in your dataset. The wildcard (*) applies the formula to all columns in the dataset.

Transformation Name	Edit column with formula
Parameter: Columns	*
Parameter: Formula	TRIM(\$col)

You can extend the above transformation further by removing any leading or trailing single- and double-quote marks using the `TRIMQUOTES` function wrapped around the `TRIM` reference:

Tip: Keep in mind that nested functions are evaluated from the inside out. In this case, the `TRIM` function is evaluated first, which removes any surrounding whitespace. Then, the `TRIMQUOTES` function is applied.

Transformation Name	Edit column with formula
----------------------------	--------------------------

Parameter: Columns	*
Parameter: Formula	TRIMQUOTES(TRIM(\$col))

Set values using other columns

You can use values from other columns to replace mismatched values in your current column. Using the previous example, mismatched postal codes are replaced by the corresponding value in the parent entity's postal code column (`parent_postal_code`):

Transformation Name	Edit column with formula
Parameter: Columns	parent_postal_code
Parameter: Formula	IF(ISMISMATCHED(postal_code, ['Zipcode']), '00000', postal_code)

Use functions to fix mismatched values

In your transforms, you can insert a predefined function to replace mismatched data values. In the following example, the value for mismatched values in the `score` column are computed as the average of all values in the column:

Transformation Name	Edit column with formula
Parameter: Columns	score
Parameter: Formula	IF(ISMISMATCHED(score, ['Decimal']), AVERAGE(score), score)

Tip: You can also use the `IFMISMATCHED` function to test for mismatched values. Unlike the above construction, however, `IFMISMATCHED` does not support an else clause when the value does match the listed data type.

Bad data typing

Tip: Particularly for dates, data is often easiest to manage as String data type. Trifacta has a number of functions that you can deploy to manage strings. After the data has been properly formatted, you can change it to the proper data type. If you change data type immediately, you may have some challenges in reformatting and augmenting it. Do this step last.

For columns that have a high percentage of mismatched values, the column's data type may have been mis-assigned. In the following example, a column containing data on precipitation in inches has been mis-typed as Boolean data:

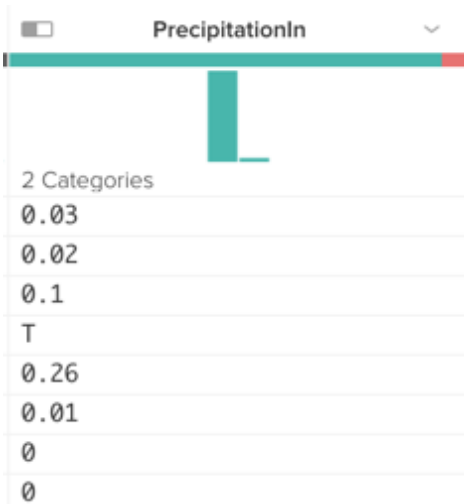


Figure: Mis-typed column data type

To change a column's data type, click the type identifier at the top of the column and select a new type. In this case, you would select `Decimal`.

NOTE: After you change the type, review the data quality bar again. If there are still mismatched values, review them to see if you can categorize the source of the mismatch.

As you can see in the previous example, the precipitation column contains values set to `T`, which may be short for `true`. When the data type is set to `Decimal`, these values now register as mismatched data. To fix, you can replace all `T` values with `1.0` using the `set` transform.

Select an instance of `T` in the column and click the `Set` suggestion card. Click **Modify**. For the `value` in the transform, enter `1.0`. Your transform should look like the following:

Transformation Name	Edit column with formula
Parameter: Columns	PrecipitationIn
Parameter: Formula	IFMATCHES([PrecipitationIn], `{start}{bool}{end}`), '1.0', PrecipitationIn)

Tip: If possible, you should review and refer to an available schema of your dataset, as generated from the source system. If the data has also been mis-typed in the source system, you should fix it there as well, so any future exports from that system show the correct type.

Find Missing Data

Contents:

- *Locate missing values*
- *Methods for fixing missing data*
- *Insert constants for missing values*
- *Copy values from another column*
- *Use functions to populate missing values*
- *Manage Missing Metadata*
 - *Example - Change Type*
 - *Example - Insert Year*
 - *Example - Insert Timezone*

When data is imported from another system, you might discover that some values are missing in it. In some cases, these values simply contain no content. In other cases, these values are non-existent. Depending on how the missing values entered the data, you may end up processing them in different ways. This section describes how to identify and manage missing data in your datasets.

NOTE: If you are unsure of the meaning of a column of data that contains missing values, you should attempt to review the source data or contact the individual who generated the data to identify why values may be missing and how to effectively manage them in Trifacta® and downstream systems.

Locate missing values

When your dataset sample is evaluated, each column is validated against the column's type definition. Based on that validation, values in the column are categorized as valid, mismatched, or missing. These values are categorized in the data quality bar at the top of each column.

- A **missing value** is any value that either contains no content or is non-existent.
 - An example of a non-existent value is a cell in a column of integers that has no value in it. In this special case, the missing value is called a **null value**.
 - Null values are converted to missing values during import.
- Values that are spaces (one or more presses of the `SPACEBAR`) or tabs (one or more presses of the `TAB` key) are not missing values.

Tip: To trim whitespace out of a column, use the following transformation:

Transformation Name	Edit column with formula
Parameter: Columns	column1
Parameter: Formula	TRIM(column1)

This step may increase the number of missing values (for values that contain only whitespace characters) and the number of instances of matching values (for values that have spaces before and after an alphanumeric value).

- Return (`\n`) and newline (`\1`) are considered missing.

In the data quality bar, missing values are identified in gray:



Figure: Missing values in gray

Tip: From the Transformer page, click the missing values in a column to see their count, highlight them in the rows of the data grid, and trigger a set of suggestions for your review.

Missing values can be sourced from a variety of issues:

- Values may be miskeyed into the source system.
- The source system may enable optional fields that do not contain values. For example, U.S. zip codes can contain a second, four-digit qualifier for the base 5-digit zip code (an extended Zip+4 code). This second value may not be required and may therefore be missing.
- For columns of generated values, a computation may not be possible from the source data, which may indicate problems with other column data.
- A set of missing values within a row may indicate a problem with the entire record.
- The source system may introduce errors in output, particularly if the data is generated using a customized structure.

Tip: When cleaning up missing data, you should look to work from bigger problems to smaller problems. If a higher percentage of a column's values have been categorized as missing data, you should look across affected rows to see if it's a wider problem. If other records look ok, you should consider deleting the column or figuring out how to manage the missing values, including populating them.

Data may also be considered missing if you don't have sufficient information about the data. For example, timestamps that do not have a timezone identifier may not be usable in the target system.

Methods for fixing missing data

When you discover mismatched data in your dataset, you have the following basic methods of fixing it:

1. **Identify if the column values are required.**
 - a. Check the target system to determine if the field must have a value. If values are not required, don't worry about it. Consider deleting the column.
 - b. Remember that null values imported into Trifacta are exported as missing values, which are easier to consume in most systems.
 - c. Check the column header and data type to determine if values are required. For example, in transactional data, a field called `coupon_code` requires data only if every transaction is processed with one.
 - d. If it's available, check the source system to see if it requires entry into the field. If an entry is required and your data contains missing values, then there is an issue in how the data was exported from the source system.
2. **Insert a constant value.** You can replace a missing value with a constant, which may make it easier to locate more important issues in the application.
3. **Use a function.** Particularly if the missing data can be computed, you can use one of the available functions to populate the missing values.
4. **Copy values from another column.** If a value from another column or a modified form of it can be used for the missing value, you can use the `set` transform to overwrite the missing values.

5. **Delete rows.** Select the missing values bar and use the `delete` transform to remove the problematic rows.

NOTE: Since missing data may not be an explicit problem, you should avoid deleting rows or the column itself until other options have been reviewed.

6. **Hide the column for now.** You can remove the column from display if you want to focus on other things. Select **Hide** from the column drop-down. Note that hidden columns still appear in any generated output.
7. **Delete the column.** If the column data is unnecessary or otherwise unusable, you can delete the entire column from your dataset. Select **Delete** from the column drop-down.

Tip: Delete unnecessary columns as early as possible. Less data is easier to work with in the application and increases job execution performance.

Insert constants for missing values

NOTE: Generally speaking, inserting constants in place of missing values is not a recommended practice, especially if downstream consuming applications and individuals may not be known. In particular, you should not replace missing numeric values with a fixed numeric value, which will skew analysis. Use this method only if your entire data chain is aware of the constants.

Steps:

1. Click the gray missing values segment of the data quality bar for the column to fix.

Tip: Select a missing value in the data grid. Then, select the `replace` suggestion and then modify it to include the replacement value.

2. In the suggestion cards, click the `set` suggestion.
3. By default, this transform sets the missing value to be a null value. Click **Edit**.
4. You might see something like the following:

Transformation Name	Edit column with formula
Parameter: Columns	country
Parameter: Formula	IF(ISMISSING([country]),NULL(),country)

5. The missing data is identified using the `row: ISMISSING` reference. To apply a constant, replace the `NULL()` reference with a constant value, as in the following:

Transformation Name	Edit column with formula
Parameter: Columns	country
Parameter: Formula	IF(ISMISSING([country]),'USA',country)

Note that the single quotes around the value are required, since it identifies the value as a constant.

6. Click **Add**.

Tip: You can also use the `IFMISSING` function to test for empty values. Unlike the above construction, however, `IFMISSING` does not support an else clause when the value is present.

Copy values from another column

You can populate missing values with values from another column. In the following example, the `nickname` column is populated with the value of `first_name` if it is missing:

Transformation Name	Edit column with formula
Parameter: Columns	nickname
Parameter: Formula	<code>IF(ISMISSING([nickname]),first_name,nickname)</code>

Use functions to populate missing values

Particularly for numeric data, you can use functions to populate missing values. In the following example, missing values for the `unit_price` column are derived from a computation of the `weight_kg` column and the `price` column:

Tip: Be careful using functions such as averages to compute missing values. These computations may factor outliers that have not yet been removed or may fail to account for local trends relative to the data. Study the values and their meaning in the column before performing replacements. When in doubt, a median value may be your best best, assuming outliers and spurious data have been properly addressed.

Transformation Name	Edit column with formula
Parameter: Columns	unit_price
Parameter: Formula	<code>IF(ISMISSING([unit_price]),(price / weight_kg),unit_price)</code>

Manage Missing Metadata

In some cases, a column may contain valid values, but the meaning of those values is missing from the data. For example, your data contains the following Timestamp information:

Timestamp
19 May 02:45:38
19 May 02:42:24
19 May 02:41:33

This timestamp information may be considered problematic for the following reasons:

- The format may be incorrect for the target system.
- There is no year information. If the target system contains multi-year datasets, it may cause issues. The month element should be interpretable by Trifacta.
- There is no timezone information. In what timezone were these entries recorded?

The following examples demonstrate how to insert this information into your timestamps.

Example - Change Type

On import, timestamp data may be classified as String data. For now, this is ok.

Tip: Particularly for dates, data is often easiest to manage as String data type. Trifacta has a number of functions that you can deploy to manage strings. After the data has been properly formatted, you can change it to the proper data type. If you change data type immediately, you may have some challenges in reformatting and augmenting it. Do this step last.

After you have added back missing elements, you can change the data type to Date/Time through the data type drop-down for the column.

Before you begin reformatting your data, you should identify the target date format to which you want to match your timestamps. From the data type drop-down, select **Date/Time**. The dialog shows the following supported date formats:

Tip: When wrangling your data, you should start with the target structure or format of your data and work back to your source. This principle applies to both column management and overall dataset management.

Date / Time Type

☐ mm

☐ yy

☐ mm-yy

☐ mm-dd

☐ dd-mm

☐ mm-dd-yy

☐ dd-mm-yy

☐ yy-mm-dd

☐ yy-dd-mm

☐ mm-dd-yy hh:mm:ss

☐ dd-mm hh:mm:ss

☐ mm-dd hh:mm:ss

☒ dd-mm-yy hh:mm:ss

☐ yy-mm-dd hh:mm:ss

☐ yy-dd-mm hh:mm:ss

☐ hh:mm:ss

dd*mm*yy*hh:MM:SS.sssa

Cancel

Save

Figure: Available Date/Time formats

NOTE: Each available option has a set of sub-options in the displayed drop-down.

In this timestamp example, the target format is the following:

dd-mm-yy hh:mm:ss (dd*shortMonth*yyyy*HH:MM:SS)

Example - Insert Year

The easiest way to handle the insertion of year information is to split out the timestamp data into separate components and then to merge back the content together with the inserted year information. Since the above timestamp data essentially contains three separate fields (Day of Month, Month, and Time), you can use a split command to break this information into three separate columns. Highlight one of the spaces between Day of Month and Month and select the `split` suggestion. The Wrangle step should look similar to the following:

Transformation Name	Split column on delimiter
Parameter: Column	column1
Parameter: Option	By delimiter
Parameter: Delimiter	' '
Parameter: Number of columns to create	2

Now, your data should be stored in three separate columns.

Tip: You may notice that new data types have been applied to the generated columns. The data may be easier to handle if all column types are converted to String type for now.

The next step involves merging all of these columns back into a single field, augmented with the appropriate year information. Select the columns in the order in which you would like to see them in the new timestamp field. In this case, you can select them in the order that they were originally listed. When all three columns are selected, choose the `merge` suggestion.

You may notice that the data has been formatted without spaces (19May02:45:38), and there is no year information yet. You can create new columns containing a year value (`myYear`) then merge the columns together:

Transformation Name	Merge columns
Parameter: Columns	column2, myYear, column3, column4
Parameter: Separator	' '

After you have inserted the year information and merged the columns, you should be able to change the column data type to the appropriate version of Date/Time.

Example - Insert Timezone

Timestamps do not natively support different timezones, so this information must be stored in a separate column. For U.S. data, timezones can be determined based on the zip code.

NOTE: If missing metadata is not supported as part of the value in the target system, you can insert the metadata as a separate column and then apply the metadata to the data inside the target system.

Manage Null Values

Contents:

- *Important notes on null values*
- *Locate null values*
- *High percentage of nulls*
- *Null values in transformations*
- *Write null values*

In general terms, a null value is a definition that points to nothing. A container for a value, such as a row-column combination or a variable, exists, but the container points to no actual value.

Important notes on null values

NOTE: In the platform, null values are a subset of the category identifying missing values. For technical reasons, however, Trifacta® displays null values as missing values and visually treats them as the same. Internally, they are understood to be different values.

Implications:

- Null values are visually represented as missing values.
 - In the data quality bar, null and missing values are represented in the dark bar (missing values).
- Computationally, they are different types of values.
 - Most functions applied to null and missing values return the same results.
 - For example, the `ISMISSING` function returns `true` for null and missing values.
 - However, the `ISNULL` function returns `true` for a null value and `false` for a missing value. See below.
 - If you use a function to generate null values, they are displayed as missing values, although they are recorded as nulls.
 - For example, the following transform generates a column of null values, which are represented as missing values in the data quality bar.

Transformation Name	New formula
Parameter: Formula	NULL()
Parameter: New column name	nulls

- When a set of results is generated, both null and missing values are written as missing values, unless the output format has a specific schema associated with it.

NOTE: When a recipe containing a user-defined function is applied to text data, any null characters cause records to be truncated when the job is run on Trifacta Photon. In these cases, please execute the job in the Spark running environment.

Locate null values

Null values are displayed with missing values in the Missing values category of the data quality bar (in gray).

You can use the following transform to distinguish between null and missing values. This transform generates a new column of values, which are set to `true` if the value in `isActive` is a null value:

Transformation Name	New formula
Parameter: Formula	ISNULL(<code>isActive</code>)
Parameter: New column name	<code>nulls2</code>

High percentage of nulls

On import, if a column has a high enough percentage of null values, the platform may retype the column as a `String` column, which may yield mismatched values in addition to the missing values that were imported from null values.

Null values in transformations

Functions:

- Applying a null value as an input to a scalar function returns a null value, propagating the null value.
- In aggregate or window functions, null values are ignored, as a single null value could corrupt an entire column of calculations.

Transforms:

- In a join, a null value in one dataset never matches with a null value in another dataset. Rows with null values in join key columns are never included in the output. See *Join Types*.

Write null values

If needed, you can write a null value to a set of data. In the following example, all missing values in a column are replaced by nulls, using the `NULL` function.

NOTE: The `NULL` function is typically used to pass null values into functions that have been designed to specifically address them.

The following example tests all columns in the range between `column1` and `column255` for whether a missing value is detected. If so, a null value is written. Otherwise, the column value is written back to the column:

Transformation Name	Edit column with formula
Parameter: Columns	<code>column1~column255</code>
Parameter: Formula	<code>IF(ISMISSING([<code>\$col</code>]), null(), <code>\$col</code>)</code>

The above transform writes null values, but these values are converted to missing values on export.

Structuring Tasks

These tasks describe different methods for changing the shape of your data. Some of these tasks are applied on data import, while others can be managed through a single transformation in your recipe.

Tip: Some transformations may add or remove data, and the source data is lost. To retain the original data, you may choose to create chains or branching sets of recipes before you apply restructuring steps. For more information, see *Create Branching Outputs*.

Initial Parsing Steps

Contents:

- *File Encoding*
 - *Automatic Structure Detection*
 - *Overview*
 - *Splitting Columns*
 - *Header Row*
 - *Converted data*
 - *Excel*
 - *JSON*
 - *Database Tables*
 - *Known Issues*
 - *Troubleshooting*
 - *Fixing parsing issues from structured source after recipe has been created*
-

When a dataset is initially loaded into the Transformer page, one or more steps may be automatically added to the new recipe in order to assist in parsing the data. The added steps are based on the type of data that is being loaded and the ability of the application to recognize the structure of the data.

File Encoding

When a text file is used as an imported dataset, Trifacta® assumes that the imported files are encoded in UTF-8, by default.

NOTE: Assessing the file encoding type based on parsing an input file is not an accurate method. Instead, Trifacta assumes that the file is encoded in the default encoding. If it is not, the Trifacta application should be prompted with the appropriate encoding type.

NOTE: In some cases, imported files are not properly parsed due to issues with encryption types or encryption keys in the source datastore. For more information, please contact your datastore administrator.

As needed, you can change the encoding to use when parsing individual files. In the Import Data page, click **Edit Settings** in the right-hand panel.

Automatic Structure Detection

NOTE: By default, these steps do not appear in the recipe panel due to automatic structure detection. If you are having issues with the initial structuring of your dataset, you may choose to re-import the dataset with Detect structure disabled. Then, you can review this section to identify how to manually structure your data.

This section provides information on how to apply initial parsing steps to unstructured imported datasets. These steps should be applied through the recipe panel.

NOTE: Imported datasets whose schema has not been detected are labeled, **unstructured datasets**. These datasets are marked in the application. When a recipe for this dataset is first loaded into the Transformer page, the structuring steps are added as the first steps to the associated recipe, where they can be modified as needed.

Overview

When data is first loaded, it is initially contained in a single column, so the initial steps apply to `column1`.

Step 1: Split the rows. In most cases, the first step added to your recipe is a Splitrows transformation, which breaks up the individual rows based on a consistently recognized pattern at the end of each line. Often, this value is a carriage return or a carriage return-new line. These values are written in Wrangle as `\r` and `\r\n`, respectively. See the example below.

NOTE: The maximum permitted length of any individual record on input is 20 MB.

Step 2: Split the columns. Next, the application attempts to break up individual rows into columns.

- If the dataset contains no schema, the Split Column transformation used. This transformation attempts to find a single consistent pattern or a sequence of patterns in row data to demarcate the end of individual values (fields).

NOTE: Avoid creating datasets that are wider than 1000 columns. Performance can degrade significantly on very wide datasets.

- If the dataset contains a schema, that information is used to demarcate the columns in the dataset.

When the above steps have been successfully completed, the data can be displayed in tabular format in the data grid.

Step 3: Add column headers. If the first row of data contains a recognizable set of column names, a Rename Columns with Rows transformation might be applied, which turns the first row of values into the names of the columns.

Example recipe:

1.	Transformation Name	Split into rows
	Parameter: Column	column1
	Parameter: Split on	\r
	Parameter: Ignore matches between	\ "
	Parameter: Quote escape character	\ "
2.	Transformation Name	Split column
	Parameter: Column	column1
	Parameter: Option	on pattern
	Parameter: Match pattern	' , '
	Parameter: Number of matches	9
	Parameter: Ignore matches between	\ "

3. Transformation Name	Add header
Parameter: Row number	1

After these steps are completed, the data type of each column is inferred from the data in the sample. See *Supported Data Types*.

Splitting Columns

When you import a dataset, the application can automatically split your column into separate columns based on one or more delimiters.

NOTE: Avoid importing datasets that are wider than 1000 columns. Particularly with previewing transformations in the data grid, very wide datasets can consume a significant amount of memory, which can cause browser crashes. Depending on your local environment, you may be able to work with these wide datasets. However, if the dataset is joined with other datasets or shared with other users, crashes can occur.

Tip: If you select the delimiter in a column with a very large number of delimiters, any suggestion card limits the split to a maximum of 250 columns. You can edit the suggested transformation to increase the number of split columns as needed. Increasing the limit can impact browser performance.

Header Row

When a dataset is imported, the application may infer the names of your columns from the first row of the dataset.

Tip: Avoid importing data that contains missing or empty values in the first row. These gaps can cause problems in your headers.

- In some cases, the application may be unable to create this header row. Instead, the columns are titled `column1`, `column2`, `column3` and so on.
- If the column names are split across multiple rows in your dataset, you may need to modify the column naming transformation step.

Converted data

Some formats, such as binary data or JSON, are converted to a format that is natively understood by the product before the data is available for sampling and transformation.

Excel

Microsoft Excel files are internally converted to CSV files and then loaded into the Transformer page. CSV files are treated using the general parsing steps. See previous section.

JSON

If 80% of the records in an imported dataset are valid JSON objects, then the data is parsed as JSON through a conversion process.

Notes:

- For JSON files, it is important to import them in unstructured format.

- Trifacta® requires that JSON files be submitted with one valid JSON object per line.
 - Multi-line JSON import is not supported.
 - Consistently malformed JSON objects or objects that overlap linebreaks might cause import to fail.

For more information, see *Working with JSON v2*.

Database Tables

Properly formatted database tables with a provided schema should not require any initial parsing steps.

Known Issues

- Some characters in imported datasets, such as `NUL` (ASCII character 0) characters, may cause problems with recognizing line breaks. If initial parsing is having trouble with line breaks, you may need to fix the issue in the source data prior to import, since the Splitrows transformation must be the first step in your recipe.

Troubleshooting

Fixing parsing issues from structured source after recipe has been created

If you discover that your dataset has issues related to initial parsing of a structured source after you have started creating your recipe, you can use the following steps to attempt to rectify the problem.

Steps:

1. Open the flow containing your recipe.
2. Select the imported dataset. From the context menu, select **Remove structure...**
3. For the imported dataset, click **Add new recipe**.
4. Make any changes to the initial parsing steps in this recipe.
5. Select the recipe you were initially modifying. From its context menu, select the new recipe as its source.

The new initial parsing steps are now inserted into recipe flow before the recipe steps in development.

Reshaping Steps

Recipe steps can change the number of rows in the dataset and apply wider impacts to your dataset and its samples.

These **reshaping steps** include the following transformations:

Transformation	Documentation
Splitrows	<i>Initial Parsing Steps</i>
Expand Arrays into Rows	<i>Working with Arrays</i>
Filter Rows (keep or delete)	<i>Remove Data</i>
Pivot Table	<i>Pivot Data</i>
Unpivot Columns	<i>Unpivot Columns</i>
Join Datasets	<i>Join Window</i>
Union Datasets	<i>Union Page</i>
Select Lookup from the column menu	<i>Lookup Wizard</i>
Remove Duplicate Rows	<i>Remove Data</i>

Samples:

When one of these transformations is applied and rows are removed from your dataset:

- Any samples generated before the step was added are invalidated and cannot be used.
- If you edit steps in your recipe before this added transformation, any samples that you generated after the step are invalidated and cannot be used.
- A valid initial sample is always available for use.

For more information, see *Samples Panel*.

Split Column

Contents:

- *Split by Delimiter*
 - *Split on single delimiter*
 - *Split column by multiple delimiters*
 - *Split column between delimiters*
- *Split by Position*
 - *Split column by positions*
 - *Split columns between positions*
 - *Split column at regular interval*
- *Encoding Issues*
- *Splitting Rows*

For many recipes, the first step is to split data from a single column into multiple columns. This section describes the various methods that can be used for splitting a single column into one or more columns, based on character- or pattern-matching or position within the column's values.

Split by Delimiter

When data is initially imported into Trifacta®, data in each row may be split on a single delimiter. In the following example, you can see that the tab key is a single clear delimiter:

<IMSI^MSIDN^IMEI>	DATETIME/TIMEZONE	OFFSET/DURATION	MSWCNT:BASCNT^BASTRA	CALL_TYPE
/CORRESP_IDN/DISCONNECT REASON				
<310170097665881^13011330554^011808005351311>	2014-12-12T00:06:13/-5/1.55			MSC001:
BSC002^BTS783	MOT/00000000000:11			
<310170097665881^13011330554^011808005351311>	2014-12-12T02:27:26/-5/0.00			MSC001:
BSC002^BTS783	SMS/00000000000:			
<310-170-097665881^13011330554^011808005351311>	2014-12-12T03:24:20/-5/0			MSC001:
BSC001^BTS783	SMS/00000000000:			

However, when this data is imported, it may be rendered in the data grid in the following structure:

column2	column3	column4	column5
<IMSI^MSIDN^IMEI>	DATETIME/TIMEZONE OFFSET/DURATION	MSWCNT: BASCNT^BASTRA	CALL_TYPE/CORRESP_IDN: DISCONNECT REASON
<310170097665881^13011330554^011808005351311>	2014-12-12T00:06:13/-5/1.55	MSC001: BSC002^BTS783	MOT/00000000000:11
<310170097665881^13011330554^011808005351311>	2014-12-12T02:27:26/-5/0.00	MSC001: BSC002^BTS783	SMS/00000000000:
<310-170-097665881^13011330554^011808005351311>	2014-12-12T03:24:20/-5/0	MSC001: BSC001^BTS783	SMS/00000000000:

Notes:

- When the data is first imported, all of it is contained in a single column named column1. The application automatically splits the columns on the tab character for you and removes the original column1.

Tip: This auto-split does not appear in your recipe by default. For most formats, a set of initial steps is automatically applied to the dataset. Optionally, you can review and modify these steps, but you must deselect Detect Structure during the import.

- Because the application was unable to determine clear headers for each column's data, generic ones are used. So, before you apply a header to your data, you must split out the data within each column.
- The delimiters within each column vary.
 - column2 uses the caret, while column3 uses the forward slash.
 - column4 and column5 use multiple delimiters.
- There is sparseness in the data. Note that in column5, the second row contains the value 11 at the end, while the other two data rows do not have this value.

Split on single delimiter

For column2, you can split the column into separate columns based on the caret delimiter:

Transformation Name	Split by delimiter
Parameter: Column	column2
Parameter: Option	By delimiter
Parameter: Delimiter	' ^ '
Parameter: Number of columns to create	2

NOTE: The Number of columns to create value reflects the total number of new columns to generate.

Results:

Below is how the data in column2 is transformed:

column1	column6	column7
<IMSI	MSIDN	IMEI>
<310170097665881	13011330554	011808005351311>
<310170097665881	13011330554	011808005351311>
<310-170-097665881	13011330554	011808005351311>

- Since column1 was unused as a name, it re-appears here. column6 and column7 are the next available generic column names.
- There is a small bit of cleanup to do in column1 and column7 to remove the symbols at the beginning and end of these column values. You can do this cleanup before the split in the original column2 if desired.

For column3, suppose that you want to keep the DATETIME and TIMEZONE OFFSET values in the same column, preserving the forward slash to demarcate these two values. The DURATION values are to be split into a separate column:

Transformation Name	Split by delimiter
Parameter: Column	column2

Parameter: Option	By delimiter
Parameter: Delimiter	' / '
Parameter: Start to split after	`/(-{digit} {digit})`

- The above uses `Patterns`, which are simplified versions of regular expressions for matching patterns.
 - In this case, the expression is the following:

```
`/(-{digit}|{digit})`
```

- For the Start to split after value, the above indicates that the application should start to look for matches on the delimiter (forward slash) only after the above pattern has been detected in the column values.
 - In this case, the pattern describes values that appear after a forward slash and could be a negative digit or a positive digit, which matches the pattern for the `TIMEZONE OFFSET` values in the column.
 - For more information on how to use `Patterns`, see *Text Matching*.
- Since you are splitting the column into two columns, you do not need to specify the number of new columns to create. The default is 1.

Split column by multiple delimiters

After splitting `column3`, the data resembled the following:

column3
DATETIME/TIMEZONE OFFSET
2014-12-12T00:06:13/-5
2014-12-12T02:27:26/-5
2014-12-12T03:24:20/-5

Suppose you want to break down the components of this date-time data into separate columns for year, month, day, hour, minute, second, and offset. The following could be use to do so:

Transformation Name	Split by delimiter
Parameter: Column	column2
Parameter: Option	By multiple delimiters
Parameter: Delimiter 1	' - '
Parameter: Delimiter 2	' - '
Parameter: Delimiter 3	' T '
Parameter: Delimiter 4	' : '
Parameter: Delimiter 5	' : '
Parameter: Delimiter 6	' / '

- Each delimiter is entered on a separate row.
- Delimiters are processed in the listed order.

Split column between delimiters

Suppose that for column4, you want to split the column such that the middle part section is removed. You could use the previous transformation and then delete the middle column. You can also use the following transformation, which identifies that starting and editing delimiters that demarcate the separator between fields, effectively removing the middle column:

Transformation Name	Split by delimiter
Parameter: Column	column4
Parameter: Option	By two delimiters
Parameter: Start delimiter	' : '
Parameter: Include as part of split	Selected
Parameter: End delimiter	' ^ '
Parameter: Include as part of split	Selected

- The separator between the columns is all of the content between the forward slashes. This content is removed from the dataset.
- The two selected options include the forward slashes as part of the separator, which removes them from the dataset.

Split by Position

You can also perform column splits based on numerical positions in column values. These splitting options are useful for highly regular data that is of consistent length.

Tip: When specifying numeric positions, you do not have to list the positions in numeric order. You can now do faster iteration since you can add new positions as needed when previewing the transformation.

Suppose you have the following coordination information in three dimensions (x, y, and z). Note that the data is very regular, with leading zeroes for values that are less than 1000.

column1
POSXPOSYPOSZ
000100040001
012405210555
100220046554
202056789011
379274329832

Split column by positions

The above data could be split based on positions within a column's value:

Transformation Name	Split by character position
Parameter: Column	column1

Parameter: Option	By positions
Parameter: Position 1	4
Parameter: Position 2	8

Results:

column2	column3	column4
POSX	POSY	POSZ
0001	0004	0001
0124	0521	0555
1002	2004	6554
2020	5678	9011
3792	7432	9832

Split columns between positions

Suppose that you wish to split the above source data such that the middle column is removed:

Transformation Name	Split by character position
Parameter: Column	column1
Parameter: Option	Between two positions
Parameter: Position 1	4
Parameter: Position 2	8

Results:

column2	column3
POSX	POSZ
0001	0001
0124	0555
1002	6554
2020	9011
3792	9832

Split column at regular interval

The above transformation could be simplified even further, since the splits happen at regular intervals:

Transformation Name	Split by character position
Parameter: Column	column1
Parameter: Option	At regular interval
Parameter: Interval	4
Parameter: Number of times to	2

Results:

The results would be the same as the first example.

Encoding Issues

If you are attempting to split columns based on non-ASCII characters that appear in the dataset, your transformations may fail.

In these cases, you should change the encoding that is applied to the dataset.

Steps:

1. In the Import Data page, select the dataset to import.
2. When the dataset card appears in the right column, click the Edit Settings link.
3. From the drop-down, select a more appropriate encoding to apply to the file.
4. Import the data and wrangle.
5. Try your split transformation on the dataset.

Splitting Rows

When a dataset is imported, the application attempts to split the data into individual rows, based on any available end of line delimiters. This transformation is performed automatically and is not included in your initial set of steps.

If the data is not consistently formatted, the rows may not be properly split. If so, you can disable the automatic splitting of rows.

Steps:

1. In the Import Data page, select the dataset to import.
2. When the dataset card appears in the right column, click the Edit Settings link.
3. Deselect the Detect Structure checkbox.
4. Import the data and wrangle.

The steps used to detect structure are listed as the first steps of your recipe, which allows you to modify them as needed.

Move Columns

Contents:

- *Cut and Paste Columns*
 - *Move using Column Menus*
 - *Move using Column Icons*
 - *Move using Transform Builder*
 - *Move multiple columns*
 - *Move range of columns*
 - *Move set of columns*
 - *Move using RapidTarget*
-

You can move or reorder individual columns or multiple columns through multiple methods.

Cut and Paste Columns

To move an individual column or multiple columns, perform the following:

Steps:

1. Select an individual column or select multiple columns. For example, select Column B and select **Cut** from the column menu.
2. Navigate to the location where you want to paste the column then select **Paste > (Paste before or Paste after)** from the column menu.

In the following example, you can see what happens when Column B is moved after Column D.

Source:

Column A	Column B	Column C	Column D
Cell A.1	Cell B.1	Cell C.1	Cell D.1
Cell A. 2	Cell B.2	Cell C.2	Cell D.2

Results:

Column A	Column C	Column D	Column B
Cell A.1	Cell C.1	Cell D.1	Cell B.1
Cell A. 2	Cell C.2	Cell D.2	Cell B.2

Move using Column Menus

You can use the **Move** option from the drop-down caret of the column context menu to move an individual column or multiple columns.

To move an individual column or multiple columns, perform the following:

Steps:

1. To select an individual column, click its column header. To select multiple columns:
 - a. You can **SHIFT**-click a range of columns.

- b. To select multiple discrete columns, press `CTRL/COMMAND` + click.
2. Select **Move** from the column menu of one of the selected columns. Choose one of the following options to move a column:
 - **to beginning**: Moves the column to the beginning of the dataset.
 - **to end**: Moves the column to the end of the dataset.
 - **after/before**: Moves the column either before or after the specified columns of the dataset.

The specified transformation is displayed in the Transform Builder. For example, the following transformation moves Column A just after Column C:

The Column(s) option defines the method by which you specify the set of columns. In this case, `Multiple` simply means that you specify each column one after another in the transformation. To add this step to your recipe, click **Add**. The columns are moved.

Tip: You can use suggestion cards to explore and select the appropriate transformation to move the columns. For more information on suggestions, see *Explore Suggestions*.

Move using Column Icons

Select the Column View icon in the Transformer bar to move columns in the Column Browser panel.

To move an individual column or multiple columns, perform the following:

Steps:

1. When you select an individual column or multiple columns, you are prompted with a set of suggestions.
2. Select the appropriate suggestion from the suggestion cards.
3. **Edit** or **Add** the steps, as required to move columns. For more information, see below examples.

Move using Transform Builder

In the Transform Builder, you can select one or more columns to move using finer-grained controls.

To move an individual column or multiple columns, perform the following:

Steps:

1. Enter **Move columns** in the Search panel of the Transform Builder.
2. Select an individual column or multiple columns, as required. The following options are available when specifying one or more columns in a transformation:
 - **Multiple**: Select one or more columns from the drop-down list. See below example.
 - **Range**: Specify a start column and ending column. All columns inclusive are selected. See below example.
 - **Advanced**: Specify the columns using a comma-separated list. You can combine multiple and range options under Advanced. Ranges of columns can be specified using the tilde (~) character. See below example.
3. Select the required option from the **Option** drop-down list.
4. Select the required column to move after or before the column.
5. Click **Add**. The selected columns are moved based on your inputs.

Move multiple columns

This example moves two discrete columns (`Column A` and `ColumnC`), before `Column E`. These columns are not next to each other, so they can be specified using the `Multiple column(s)` option.

Source:

Column A	Column B	Column C	Column D	Column E
Cell A.1	Cell B.1	Cell C.1	Cell D.1	Cell E.1
Cell A. 2	Cell B.2	Cell C.2	Cell D.2	Cell E. 2

Transformation:

Transformation Name	Move Columns
Parameter: Column(s)	Multiple
Parameter: Column	A, C
Parameter: Option	Before
Parameter: Column	E

Results:

Column B	Column D	Column A	Column C	Column E
Cell B.1	Cell D.1	Cell A.1	Cell C.1	Cell E.1
Cell B.2	Cell D. 2	Cell A.2	Cell C.2	Cell E.2

Move range of columns

You can move a range of columns to a specified location. For example, you can move Column A through Column C after Column D.

Source:

Column A	Column B	Column C	Column D
Cell A.1	Cell B.1	Cell C.1	Cell D.1
Cell A. 2	Cell B.2	Cell C.2	Cell D.2

Transformation:

Transformation Name	Move Columns
Parameter: Column(s)	Range
Parameter: Column	A~C
Parameter: Option	After
Parameter: Column	D

Results:

Column D	Column A	Column B	Column C
Cell D.1	Cell A.1	Cell B.1	Cell C.1
Cell D. 2	Cell A.2	Cell B.2	Cell C.2

Move set of columns

Using the Advanced option, you can move combinations of column ranges and discrete columns to a new location. In the following example, ColumnA through ColumnC and ColumnE are moved after ColumnF:

Source:

Column A	Column B	Column C	Column D	Column E	Column F
Cell A.1	Cell B.1	Cell C.1	Cell D.1	Cell E.1	Cell F.1
Cell A. 2	Cell B.2	Cell C.2	Cell D.2	Cell E.2	Cell F.2

Transformation:

In the transformation, you select the Advanced column(s) option where you can specify columns on a single line.

Tip: The tilde character (~) can be used to specify the range of columns between two listed columns. Ranges and individual columns should be separated by a comma.

ColumnA~ColumnC, ColumnE

Transformation Name	Move Columns
Parameter: Column(s)	Advanced
Parameter: Column	A~C, E
Parameter: Option	After
Parameter: Column	F

Results:

Column D	Column F	Column A	Column B	Column C	Column E
Cell D.1	Column F.1	Cell A.1	Cell B.1	Cell C.1	Cell E.1
Cell D.2	Column F.2	Cell A.2	Cell B.2	Cell C.2	Cell E.2

For more information, see *Column Reference Syntax*.

Move using RapidTarget

RapidTarget allows you to associate a target set of columns with your recipe. When you specified a target, you can often reposition your source columns with the targets by clicking in the interface. For more information, see *Overview of RapidTarget*.

Delete Data

Contents:

- *Delete Columns*
 - *By selection*
 - *Through transformation*
 - *Delete Rows*
 - *By selection*
 - *By custom conditions*
-

A key task in cleaning up your data is to remove unwanted columns and rows, which can simplify future transformations and improve job execution performance. Trifacta® provides multiple mechanisms for removing data from your dataset.

Tip: When you are deleting data, you should consider if that data may have other uses in the future or for other users. If so, you should consider doing the data removal through a separate recipe off of your current recipe, which preserves the data for other uses in the current recipe.

Delete Columns

You can delete one or more columns based on the following:

- By selection
- Through transformation

Tip: When you delete through transformation steps, you have additional controls at your disposal.

By selection

You can delete a single column or multiple columns:

- To delete a column from your dataset, click the column and select **Delete** from the column drop-down.
- If you select **Delete others**, all other remaining columns are deleted except the selected column.

Tip: To delete multiple columns, select them in the data grid or column browser. Then select **Delete** from the column menu.

The column or columns are removed from the data grid, and a new step is added to your recipe.

Through transformation

You can delete columns through the transformation steps.

Steps:

1. In the Transformer page, click **Delete columns**.
2. The Delete columns transformation is populated in the Transformer Builder.
3. Select one or more columns, as required:
 - a. **Multiple:** Select one or more columns from the drop-down list.
 - b. **All:** Select all columns in the dataset.

NOTE: This step removes all columns in your dataset.

- c. **Range:** Specify a start and ending columns. All columns inclusive of start and end are deleted.
- d. **Advanced:** Specify the columns using a comma-separated list. Ranges of columns can be specified using the tilde (~) character. Examples:

Entry	Description
Store_Nbr ~ Daily	Columns from Store_Nbr to Daily in the dataset are deleted.
Store_Name , Store_Manager , Store_Nbr ~ Daily	The following columns are deleted: Store_Name Store_Manag er Store_Nbr to Daily

- 4. From the **Action** area, select one of the following options:
 - a. **Delete selected columns:** Deletes only the selected columns.
 - b. **Delete unselected columns:** Deletes all other remaining columns except the selected columns.
- 5. To delete columns, click **Add**.

Example transformation:

The following transformation deletes the columns between Store_Nbr and Daily, inclusive.

Transformation Name	Delete columns
Parameter: Columns	Advanced
Parameter: Column	Store_Nbr~Daily
Parameter: Action	Delete selected columns

Delete Rows

Since rows do not have an identifying header, you must identify the rows to remove in your dataset based on a specified condition. You can delete rows based on the following:

- By selection
- By custom conditions

By selection

You can delete rows by selecting values. You are prompted for data filtering suggestions when you select values in:

- column histograms
- column data quality bars
- cells or values within a cell

When you make a selection, select the Delete rows transformation in the context panel. The Transform Builder contains a transformation to filter rows based on the the condition that you have selected. For example, if you selected the value California in the State column, then the transformation is specified to filter out rows in which State=California.

In the Transform Builder, you must decide if the transformation keeps matching rows (deleting all others) or deletes matching rows. In the following example, rows in which State=California are selected for deletion:

Transformation Name	Filter rows
----------------------------	-------------

Parameter: Condition	Custom formula
Parameter: Type of formula	Custom single
Parameter: Condition	State == "California"
Parameter: Action	Delete matching rows

By custom conditions

You can delete a set of rows based on a condition specified in the `condition` column . If the conditional expression is `true` , then the selected rows are deleted.

1. In the Transformer page, click the **Recipe** icon. The Recipe panel is displayed.
2. In the Search Transformations panel, enter `Filter` in.
3. In the Filter rows transformation, enter the required details:
 - a. **Condition:** Filter based on the condition type that you select in the drop-down. Some condition types do not support specifying the condition by formula.
 - b. **Column:** The column containing the values to filter. For example, `action_count`.
 - c. **Values or Formula:** Specify the values or the formula used to determine the condition.
 - i. If these values are present, then the condition evaluates to `true`.
 - ii. The formula must evaluate to `true` or `false`.
 - d. **Action:** The action to be performed to the rows based on the specified conditions.
 - e. In the following example, the rows where the `action_count` column values fall between 1 and 10 are deleted:

Transformation Name	Filter rows
Parameter: Condition	Custom formula
Parameter: Type of formula	Custom single
Parameter: Condition	<code>(action_count >= 1) && (action_count <= 10)</code>
Parameter: Action	Delete matching rows

Tip: You can apply logical operators such as `&&` (logical AND) above to build more sophisticated logical tests.

4. To add the recipe to the step, click **Add**. The dataset rows are filtered based on the configured transformation.

Select

You can completely replace the columns in your dataset by selecting source columns, functions computed from the source, and constant values.

NOTE: This transformation completely replaces the existing table, which could have significant effects on any downstream recipes or reference datasets that already exist.

Create Your Table

Steps:

1. In the Transformer page, open the Recipe panel.
2. In the recipe, locate the step where you wish to insert the transformation to create your new table.

NOTE: If your Create Table transformation renames or omits columns, references to them later in your recipe or in other downstream objects may be broken.

3. In the search bar, enter `Select`. Choose the transformation.
4. In the Transform Builder, you can create the columns in order for your new table. For each column:
 - a. In the upper field, enter the source of the column. The source can be one of the following:
 - i. A column name in your source
 - ii. A function. Example:

```
POW(myBaseVal, 5)
```

NOTE: When creating a table, aggregate and window functions are not supported. After you have created your table, you can apply these functions are normal.

- iii. A constant value. Example:

```
&apos;valid&apos;
```

- b. In the lower field, you enter a name for the column in the new table.
5. To add a new column, click **Add**. Repeat the previous steps.
 - a. You can remove columns, if needed. Click **Remove** next to the column entry.
 6. To create the new table when you've specified your columns, click **Add**.

The new table replaces your previous set of columns.

Tip: After you have created your new table, you can disable or delete the step to revert to the previous state.

Use RapidTarget

This transformation is added to your recipe when you perform column matching between your source dataset and a target schema. The results of your column matching work are rendered as a single Create Table transformation in your recipe.

Tip: If you have a target schema to which you can assign to your recipe, you may find it easier to create your new table using RapidTarget, which provides a visual interface for performing these remappings.

NOTE: RapidTarget does not support inserting columns containing constants or generated by functions. You can insert those column as a later step.

For more information, see *Overview of RapidTarget*.

Create Aggregations

Contents:

- *Limitations*
 - *Example Data*
 - *Aggregating across all rows (no grouping)*
 - *Aggregate grouped-by rows*
 - *Generate new aggregation table*
-

You can apply aggregate functions to groups of values in one or more columns to generate aggregated data. Depending on how you configure the Group By transformation, the output of these transformations is a new table or one or more columns in the current dataset.

Limitations

- The Group By transformation does not support nested expressions. You cannot insert multiple nested expressions in your computed value.
- The Group By transformation supports aggregation functions only.

Example Data

The following table contains test score data from a set of students for four separate tests, spread over two days:

Student	TestDate	TestNum	TestScore
Anna	09/08/2018	1	84
Ben	09/08/2018	1	71
Caleb	09/08/2018	1	76
Danielle	09/08/2018	1	87
Anna	09/08/2018	2	92
Ben	09/08/2018	2	86
Caleb	09/08/2018	2	99
Danielle	09/08/2018	2	73
Anna	09/15/2018	3	86
Ben	09/15/2018	3	99
Caleb	09/15/2018	3	86
Danielle	09/15/2018	3	80
Anna	09/15/2018	4	85
Ben	09/15/2018	4	87
Caleb	09/15/2018	4	79
Danielle	09/15/2018	4	93

Aggregating across all rows (no grouping)

You can perform basic computations across all rows of the dataset. For example, the following transformation creates a new column containing the average test score for all students:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	ROUND(AVERAGE(Score), 2)
Parameter: New column name	avg_TestScore

The above results in a new column called, `average_TestScore`, containing the single value 85.19, which is the average of all students' test scores rounded to two decimal places.

NOTE: These types of aggregations are known as **flat aggregations**. In larger datasets, performing flat aggregations can be computationally intensive. Be careful in computing any aggregation functions across a large number of rows.

Aggregate grouped-by rows

For the above example data, suppose you are interested in the average score for each student. In this case, you must compute the average (`AVERAGE(TestScore)`) for each student.

In the previous transformation, you used the New Formula transformation. When you are computing aggregations across groups of values in a column, you must use the Group By transformation:

Transformation Name	Group By
Parameter: Group By	Student
Parameter: Values	AVERAGE(TestScore)
Parameter: Type	Group by as new column(s)

Note that the above transformation does not contain the rounding function. Nested expressions are not supported in the Group By transformation. To round the values, add the following transformation as the next step:

Transformation Name	Edit column with formula
Parameter: Columns	average_TestScore
Parameter: Formula	ROUND(average_TestScore, 2)

You may wish to rename the newly generated column to something like `average_TestScorePerStudent` instead.

The output data should look like the following:

Student	TestDate	TestNum	TestScore	average_TestScorePerStudent	average_TestScore
Anna	09/08/2018	1	84	86.75	85.19
Ben	09/08/2018	1	71	85.75	85.19
Caleb	09/08/2018	1	76	85	85.19
Danielle	09/08/2018	1	87	83.25	85.19
Anna	09/08/2018	2	92	86.75	85.19
Ben	09/08/2018	2	86	85.75	85.19
Caleb	09/08/2018	2	99	85	85.19

Danielle	09/08/2018	2	73	83.25	85.19
Anna	09/15/2018	3	86	86.75	85.19
Ben	09/15/2018	3	99	85.75	85.19
Caleb	09/15/2018	3	86	85	85.19
Danielle	09/15/2018	3	80	83.25	85.19
Anna	09/15/2018	4	85	86.75	85.19
Ben	09/15/2018	4	87	85.75	85.19
Caleb	09/15/2018	4	79	85	85.19
Danielle	09/15/2018	4	93	83.25	85.19

Generate new aggregation table

Suppose you wish to calculate the minimum, maximum, and average scores for each test. In this case, it may be more useful to create a new table in which the student names have been removed:

Transformation Name	Group By
Parameter: Group By	TestNum
Parameter: Values1	MAX(TestScore)
Parameter: Values2	MIN(TestScore)
Parameter: Values3	AVERAGE(TestScore)
Parameter: Type	Group by as new table

The resulting data looks like the following:

TestNum	max_TestScore	min_TestScore	average_TestScore
1	87	71	79.5
2	99	73	87.5
3	99	80	87.75
4	93	79	86

Tip: In this case, when you replace the existing table with a completely new table, data that is not included in the aggregation is lost. You can add columns to the list of values if you wish to bring forward untouched columns into the new table. You may also consider building aggregation tables in a recipe that is extended from the previous recipe, so that you can continue to work with the other columns in your dataset.

Nest Your Data

In Trifacta®, you can nest columns into arrays and objects (maps) using a variety of transformations.

Nest Columns into Array

This section provides simple examples of nesting columns into Arrays by extracting values from a column or nesting one or more columns into an Array column.

Create by extraction:

You can create an array of values by extracting pattern-based values from a specified column. The following transformation extracts from the `msg` column a list of all values where all letters are capitalized and places them into the new `acronyms` column:

Transformation Name	Extract matches into Array
Parameter: Column	msg
Parameter: Pattern matching elements in the list	<code>`{upper}+`</code>
Parameter: New column name	acronyms

msg	acronyms
SCUBA, IMHO, is the greatest sport in the world.	["SCUBA","IMHO"]
	[]
LOL, that assignment you finished is DOA. You need to fix it PDQ.	["LOL","DOA","Y","PDQ"]

Notes:

- An empty input column value renders an empty array.
- In the final row, the Pattern matches on the "Y" value. To fix this, you can change the Pattern matching value to the following, which matches on two or more uppercase letters in a row:

```
`{upper}{upper}+`
```

Create by nesting:

You can create arrays by nesting together the values from multiple columns:

num1	num2	num3
11	12	13
14	15	16
17	18	19

You can nest the values in `num1` and `num2` into a single array and then to nest the array with `num3`:

NOTE: If you are nesting a multi-level array, you should nest from the lowest level to the top level.

Transformation Name	Nest columns into Objects
----------------------------	---------------------------

Parameter: Columns1	num1
Parameter: Columns2	num2
Parameter: Nest columns to	Array
Parameter: New column name	nest1

Then, you can perform the nesting of the top-level elements:

NOTE: The order in which you list the columns to nest determines the order in which the elements appear in the generated array.

Transformation Name	Nest columns into Objects
Parameter: Columns1	nest1
Parameter: Columns2	num3
Parameter: Nest columns to	Array
Parameter: New column name	nest2

In the generated columns, you notice that all values are quoted, even though these values are integers.

NOTE: Elements that are generated into arrays using a nest transformation are always rendered as quoted values.

You can use the following transformation to remove the quotes from the `nest2` column:

Transformation Name	Replace text or patterns
Parameter: Column	nest2
Parameter: Find	' '
Parameter: Replace	(empty)
Parameter: Match all occurrences	true

num1	num2	num3	nest2
11	12	13	[[11,12],13]
14	15	16	[[14,15],16]
17	18	19	[[17,18],19]

Nest Columns into Objects

You can nest multiple columns into a single column of objects using `nest` transform.

This section provides a simple example of nesting columns into a new column of Object data type.

Source:

In the following example, furniture product dimensions are stored in separate columns in cm.

Category	Name	Length_cm	Width_cm	Height_cm
bench	Hooska	118.11	74.93	46.34
lamp	Tansk	30.48	30.48	165.1
bookshelf	Brock	27.94	160.02	201.93
couch	Loafy	95	227	83

Transformation:

Use the `nest` transform to bundle the data into a single column.

Transformation Name	Nest columns into Objects
Parameter: Columns	Length_cm,Width_cm,Height_cm
Parameter: Nest columns to	Object
Parameter: New column name	'Dimensions'

Results:

Category	Name	Length_cm	Width_cm	Height_cm	Dimensions
bench	Hooska	118.11	74.93	46.34	{"Length_cm":"118.11","Width_cm":"74.93","Height_cm":"46.34"}
lamp	Tansk	30.48	30.48	165.1	{"Length_cm":"30.48","Width_cm":"30.48","Height_cm":"165.1"}
bookshelf	Brock	27.94	160.02	201.93	{"Length_cm":"27.94","Width_cm":"160.02","Height_cm":"201.93"}
couch	Loafy	95	227	83	{"Length_cm":"95","Width_cm":"227","Height_cm":"83"}

Unnest Your Data

Contents:

- *Flatten Array Values into Rows*
 - *Unnest Array Values into New Columns*
 - *Flatten and Unnest Together*
 - *Unnest Object Values into New Columns*
 - *Extract a Set of Values*
-

You can unnest Array or Object values into separate rows or columns using the following transformations.

Flatten Array Values into Rows

Array values can be flattened into individual values in separate rows.

This section describes how to flatten the values in an Array into separate rows in your dataset.

Source:

In the following example dataset, students took the same test three times, and their scores were stored in any array in the `Scores` column.

LastName	FirstName	Scores
Adams	Allen	[81,87,83,79]
Burns	Bonnie	[98,94,92,85]
Cannon	Chris	[88,81,85,78]

Transformation:

When the data is imported, you might have to re-type the `Scores` column as an array:

Transformation Name	Change column data type
Parameter: Columns	<code>Scores</code>
Parameter: New type	<code>Array</code>

You can now flatten the `Scores` column data into separate rows:

Transformation Name	Expand Array into rows
Parameter: Column	<code>Scores</code>

Results:

LastName	FirstName	Scores
Adams	Allen	81
Adams	Allen	87
Adams	Allen	83

Adams	Allen	79
Burns	Bonnie	98
Burns	Bonnie	94
Burns	Bonnie	92
Burns	Bonnie	85
Cannon	Chris	88
Cannon	Chris	81
Cannon	Chris	85
Cannon	Chris	78

Tip: You can use aggregation functions on the above data to complete values like average, minimum, and maximum scores. When these aggregation calculations are grouped by student, you can perform the calculations for each student.

Unnest Array Values into New Columns

You can also split out the individual values in an array into separate columns.

This section describes how to unnest the values in an Array into separate columns in your dataset.

Source:

In the following example dataset, students took the same test three times, and their scores were stored in any array in the `Scores` column.

LastName	FirstName	Scores
Adams	Allen	[81,87,83,79]
Burns	Bonnie	[98,94,92,85]
Cannon	Chris	[88,81,85,78]

Transformation:

When the data is imported, you might have to re-type the `Scores` column as an array:

Transformation Name	Change column data type
Parameter: Columns	Scores
Parameter: New type	Array

You can now unnest the `Scores` column data into separate columns:

Transformation Name	Unnest Objects into columns
Parameter: Column	Scores
Parameter: Parameter: Paths to elements	[0]
Parameter: Parameter: Paths to elements	[1]

Parameter: Parameter: Paths to elements	[2]
Parameter: Parameter: Paths to elements	[3]
Parameter: Remove elements from original	true
Parameter: Include original column name	true

In the above transformation:

- Each path is specified in a separate row.
 - The [*x*] syntax indicates that the path is the *x*th element of the array.
 - The first element of an array is referenced using [0].
- You can choose to delete the element from the original or not. Deleting the element can be a helpful way of debugging your transformation. If all of the elements are gone, then the transformation is complete.
- If you include the original column name in the output column names, you have some contextual information for the outputs.

Results:

LastName	FirstName	Scores_0	Scores_1	Scores_2	Scores_3
Adams	Allen	81	87	83	79
Burns	Bonnie	98	94	92	85
Cannon	Chris	88	81	85	78

Flatten and Unnest Together

The following example illustrates how flatten and unnest can be used together to reshape your data.

This example illustrates you to use the flatten and unnest transforms.

Source:

You have the following data on student test scores. Scores on individual scores are stored in the `Scores` array, and you need to be able to track each test on a uniquely identifiable row. This example has two goals:

1. One row for each student test
2. Unique identifier for each student-score combination

LastName	FirstName	Scores
Adams	Allen	[81,87,83,79]
Burns	Bonnie	[98,94,92,85]
Cannon	Charles	[88,81,85,78]

Transformation:

When the data is imported from CSV format, you must add a `header` transform and remove the quotes from the `Scores` column:

Transformation Name	Rename column with row(s)
---------------------	---------------------------

Parameter: Option	Use row(s) as column names
Parameter: Type	Use a single row to name columns
Parameter: Row number	1

Transformation Name	Replace text or pattern
Parameter: Column	colScores
Parameter: Find	'\''
Parameter: Replace with	' '
Parameter: Match all occurrences	true

Validate test date: To begin, you might want to check to see if you have the proper number of test scores for each student. You can use the following transform to calculate the difference between the expected number of elements in the `Scores` array (4) and the actual number:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>(4 - arraylen(Scores))</code>
Parameter: New column name	'numMissingTests'

When the transform is previewed, you can see in the sample dataset that all tests are included. You might or might not want to include this column in the final dataset, as you might identify missing tests when the recipe is run at scale.

Unique row identifier: The `Scores` array must be broken out into individual rows for each test. However, there is no unique identifier for the row to track individual tests. In theory, you could use the combination of `LastName-FirstName-Scores` values to do so, but if a student recorded the same score twice, your dataset has duplicate rows. In the following transform, you create a parallel array called `Tests`, which contains an index array for the number of values in the `Scores` column. Index values start at 0:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>range(0,arraylen(Scores))</code>
Parameter: New column name	'Tests'

Also, we will want to create an identifier for the source row using the `sourcerownumber` function:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>sourcerownumber()</code>
Parameter: New column name	'orderIndex'

One row for each student test: Your data should look like the following:

LastName	FirstName	Scores	Tests	orderIndex
Adams	Allen	[81,87,83,79]	[0,1,2,3]	2
Burns	Bonnie	[98,94,92,85]	[0,1,2,3]	3
Cannon	Charles	[88,81,85,78]	[0,1,2,3]	4

Now, you want to bring together the `Tests` and `Scores` arrays into a single nested array using the `arrayzip` function:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>arrayzip([Tests,Scores])</code>

Your dataset has been changed:

LastName	FirstName	Scores	Tests	orderIndex	column1
Adams	Allen	[81,87,83,79]	[0,1,2,3]	2	[[0,81],[1,87],[2,83],[3,79]]
Adams	Bonnie	[98,94,92,85]	[0,1,2,3]	3	[[0,98],[1,94],[2,92],[3,85]]
Cannon	Charles	[88,81,85,78]	[0,1,2,3]	4	[[0,88],[1,81],[2,85],[3,78]]

Use the following to unpack the nested array:

Transformation Name	Expand arrays to rows
Parameter: Column	<code>column1</code>

Each test-score combination is now broken out into a separate row. The nested Test-Score combinations must be broken out into separate columns using the following:

Transformation Name	Unnest Objects into columns
Parameter: Column	<code>column1</code>
Parameter: Paths to elements	<code>'[0]','[1]'</code>

After you delete `column1`, which is no longer needed you should rename the two generated columns:

Transformation Name	Rename columns
Parameter: Option	Manual rename
Parameter: Column	<code>column_0</code>
Parameter: New column name	<code>'TestNum'</code>

Transformation Name	Rename columns
Parameter: Option	Manual rename
Parameter: Column	<code>column_1</code>

Parameter: New column name	'TestScore'
-----------------------------------	-------------

Unique row identifier: You can do one more step to create unique test identifiers, which identify the specific test for each student. The following uses the original row identifier `OrderIndex` as an identifier for the student and the `TestNumber` value to create the `TestId` column value:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>(orderIndex * 10) + TestNum</code>
Parameter: New column name	'TestId'

The above are integer values. To make your identifiers look prettier, you might add the following:

Transformation Name	Merge columns
Parameter: Columns	'TestId00', 'TestId'

Extending: You might want to generate some summary statistical information on this dataset. For example, you might be interested in calculating each student's average test score. This step requires figuring out how to properly group the test values. In this case, you cannot group by the `LastName` value, and when executed at scale, there might be collisions between first names when this recipe is run at scale. So, you might need to create a kind of primary key using the following:

Transformation Name	Merge columns
Parameter: Columns	'LastName', 'FirstName'
Parameter: Separator	' - '
Parameter: New column name	'studentId'

You can now use this as a grouping parameter for your calculation:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>average(TestScore)</code>
Parameter: Group rows by	<code>studentId</code>
Parameter: New column name	'avg_TestScore'

Results:

After you delete unnecessary columns and move your columns around, the dataset should look like the following:

TestId	LastName	FirstName	TestNum	TestScore	studentId	avg_TestScore
TestId0021	Adams	Allen	0	81	Adams-Allen	82.5
TestId0022	Adams	Allen	1	87	Adams-Allen	82.5

TestId0023	Adams	Allen	2	83	Adams-Allen	82.5
TestId0024	Adams	Allen	3	79	Adams-Allen	82.5
TestId0031	Adams	Bonnie	0	98	Adams-Bonnie	92.25
TestId0032	Adams	Bonnie	1	94	Adams-Bonnie	92.25
TestId0033	Adams	Bonnie	2	92	Adams-Bonnie	92.25
TestId0034	Adams	Bonnie	3	85	Adams-Bonnie	92.25
TestId0041	Cannon	Chris	0	88	Cannon-Chris	83
TestId0042	Cannon	Chris	1	81	Cannon-Chris	83
TestId0043	Cannon	Chris	2	85	Cannon-Chris	83
TestId0044	Cannon	Chris	3	78	Cannon-Chris	83

Unnest Object Values into New Columns

This example shows how you can unnest Object data into separate columns. The example contains vehicle identifiers, and the `Properties` column contains key-value pairs describing characteristics of each vehicle.

This example shows how you can unpack data nested in an Object into separate columns.

Source:

You have the following information on used cars. The `VIN` column contains vehicle identifiers, and the `Properties` column contains key-value pairs describing characteristics of each vehicle. You want to unpack this data into separate columns.

VIN	Properties
XX3 JT4522	year=2004,make=Subaru,model=Impreza,color=green,mileage=125422,cost=3199
HT4 UJ9122	year=2006,make=VW,model=Passat,color=silver,mileage=102941,cost=4599
KC2 WZ9231	year=2009,make=GMC,model=Yukon,color=black,mileage=68213,cost=12899
LL8 UH4921	year=2011,make=BMW,model=328i,color=brown,mileage=57212,cost=16999

Transformation:

Add the following transformation, which identifies all of the key values in the column as beginning with alphabetical characters.

- The `valueafter` string identifies where the corresponding value begins after the key.
- The `delimiter` string indicates the end of each key-value pair.

Transformation Name	Convert keys/values into Objects
Parameter: Column	Properties
Parameter: Key	`{alpha}+`
Parameter: Separator between key and value	`=`
Parameter: Delimiter between pair	`,`

Now that the Object of values has been created, you can use the `unnest` transform to unpack this mapped data. In the following, each key is specified, which results in separate columns headed by the named key:

NOTE: Each key must be entered on a separate line in the Path to elements area.

Transformation Name	Unnest Objects into columns
Parameter: Column	extractkv_Properties
Parameter: Paths to elements	year
Parameter: Paths to elements	make
Parameter: Paths to elements	model
Parameter: Paths to elements	color
Parameter: Paths to elements	mileage
Parameter: Paths to elements	cost

Results:

When you delete the unnecessary Properties columns, the dataset now looks like the following:

VIN	year	make	model	color	mileage	cost
XX3 JT4522	2004	Subaru	Impreza	green	125422	3199
HT4 UJ9122	2006	VW	Passat	silver	102941	4599
KC2 WZ9231	2009	GMC	Yukon	black	68213	12899
LL8 UH4921	2011	BMW	328i	brown	57212	16999

Extract a Set of Values

This example shows how to extract values (for example, hashtag values) from a column and convert them into a column of arrays.

In this example, you extract one or more values from a source column and assemble them in an Array column.

Suppose you need to extract the hashtags from customer tweets to another column. In such cases, you can use the `{hashtag}` Trifacta pattern to extract all hashtag values from a customer's tweets into a new column.

Source:

The following dataset contains customer tweets across different locations.

User Name	Location	Customer tweets
James	U.K	Excited to announce that we've transitioned Wrangler from a hybrid desktop application to a completely cloud-based service! #dataprep #businessintelligence #CommitToCleanData # London
Mark	Berlin	Learnt more about the importance of identifying issues in your data—early and often #CommitToCleanData #predictivetransformations #realbusinessintelligence
Catherine	Paris	Clean data is the foundation of your analysis. Learn more about what we consider the five tenets of sound #dataprep, starting with #1a prioritizing and setting targets. #startwiththeuser #realbusinessintelligence #Paris

Dave	New York	Learn how #NewYorklife onboarded as part of their #bigdata #dataprep initiative to unlock hidden insights and make them accessible across departments.
Christy	San Francisco	How can you quickly determine the number of times a user ID appears in your data?#dataprep #pivot #aggregation#machinelearning initiatives #SFO

Transformation:

The following transformation extracts the hashtag messages from customer tweets.

Transformation Name	Extract matches into Array
Parameter: Column	customer_tweets
Parameter: Pattern matching elements in the list	`{hashtag}`
Parameter: New column name	Hashtag tweets

Then, the source column can be deleted.

Results:

User Name	Location	Hashtag tweets
James	U.K	["#dataprep", "#businessintelligence", "#CommitToCleanData", " # London"]
Mark	Berlin	["#CommitToCleanData", "#predictivetransformations", "#realbusinessintelligence", "0"]
Catherine	Paris	["#dataprep", "#startwiththeuser", "#realbusinessintelligence", "# Paris"]
Dave	New York	["#NewYorklife", "dataprep", "bigdata", "0"]
Christy	SanFrancisco	["dataprep", "#pivot", "#aggregation", "#machinelearning"]

Pivot Data

Contents:

- *Building a Pivot Table*
- *Available Aggregations*
- *Simple Pivot Table*
- *Conditional Aggregations*
- *Multiple Aggregation Levels*
- *Group By*
- *Values to Columns*

A **pivot table** summarizes data that is sourced from another table. Using pivot tables, you can calculate aggregating functions, such as sums, maximums, and averages for one or more columns of data.

Optionally, these sums can be performed across groups of values from one column and broken out in columns based on the values in another. In Trifacta®, a pivot table is composed of the following basic elements:

Pivot table element	Description
Column labels	List of one or more columns whose values are represented as the columns in the generated pivot table.
Row labels	List of one or more columns whose values become the rows in the generated pivot table.
Values	Also known as facts , these values are one or more aggregation formulas, which are calculated in the following manner: <i>"Show me the value of this formula computed by each row value for every value represented in the generated table."</i>

NOTE: If your aggregation does not include the kind of transformation listed above, in which the data is pivoted from rows into columns, you can use the Group By transformation and an aggregate function. See *Create Aggregations*.

Building a Pivot Table

Pivot tables are very powerful tools for summarizing and visualizing large-scale volumes of data. In Trifacta, search for `pivot table` in the Search panel to create one.

NOTE: A pivot table completely replaces the source table. Data that is not captured in the pivot definition is lost.

Tip: In your flows, you may find it useful to create your pivot tables in independent recipes that are chained from your primary recipe.

Example Data

Pivot tables are perhaps best explained by example. The following table snippet captures transactional data from a number of stores for a range of products across a set of dates. Transactional values include total sales, quantity, and cost (`POS_Sales`, `POS_Qty`, and `POS_Cost`):

Daily	Store_Nbr	POS_Sales	POS_Qty	POS_Cost	PRODUCT_DESC
-------	-----------	-----------	---------	----------	--------------

2/8/13	1	70	7	4.97	ACME LAWN GARDEN BAG CLEAR
2/7/13	2	10.62	9	8.37	ACME COOKIES CHOC CHIP
2/7/13	2	0	0	0	ACME SANDWICH BAG
2/7/13	2	7.08	6	5.58	ACME SODAS SALTED
2/7/13	2	3.92	2	2.82	ACME SCENTED OIL REFILL-CTRY SUN
2/7/13	2	13.44	7	10.36	ACME LARGE FUDGE GRAHAMS COOKIES
2/7/13	2	0	0	0	ACME SUGAR ICE WAFERS VANILLA
2/7/13	3	3.16	2	2.86	ACME ZOO ANIMAL FRUIT SNACKS 6'S
2/7/13	3	3.16	2	2.78	ACME WAFERS SUGER ICE
2/7/13	3	3.16	2	2.82	ACME SCENTED OIL REFILL-CTRY SUN
2/7/13	3	6.32	4	5.92	ACME RICE CRACKERS ONION
2/2/13	9	150	30	16.2	ACME FROSTED OATMEAL COOKIE SQUA
2/2/13	9	3.5	2	4.86	ACME FRUIT SNACK CASTLE ADVENTRS
2/2/13	9	90	9	8.37	ACME COOKIES CHOC CHIP
2/2/13	9	30	6	3.24	ACME ASSORTED COOKIES DRP
2/2/13	9	70	7	6.51	ACME KITCHEN BAG
2/2/13	9	170	17	15.81	ACME SNACK BAGS RESEALABLE
2/2/13	9	20	4	2.16	ACME CHEDDARY SN CRACKERS/PROCES
2/2/13	9	6.5	2	8.98	ACME RICE CRACKERS TERIYAKI
2/2/13	9	1.5	3	1.62	ACME COOKIE MAPLE LEAF CREME
2/2/13	9	30	6	3.24	ACME RICE CHIPS CHEDDAR
2/1/13	7	190	38	20.52	ACME FROSTED OATMEAL COOKIE SQUA
2/1/13	7	20	2	1.86	ACME COOKIES CHOC CHIP
2/1/13	7	10	1	0.82	ACME DIGESTIVE RICH TEA BISCUITS
2/1/13	7	120	24	12.96	ACME ASSORTED COOKIES DRP
2/1/13	7	120	12	11.16	ACME KITCHEN BAG
2/1/13	7	90	9	8.37	ACME SNACK BAGS RESEALABLE
2/1/13	7	10	1	0.71	ACME FUDGE MINT COOKIES SQUARES
2/1/13	7	9.5	19	10.26	ACME CHEDDARY SN CRACKERS/PROCES
2/1/13	7	10	1	0.82	ACME COOKIES MAPLE CREAM
2/1/13	7	40	8	4.32	ACME COOKIE MAPLE LEAF CREME

Available Aggregations

The Pivot data transformation supports use of any aggregation function. For more information, see *Aggregate Functions*.

Simple Pivot Table

From the above, suppose you are interested in the sales from each store for each product. You can use the following transformation to compute these aggregated calculations:

Transformation Name	Pivot table
----------------------------	-------------

Parameter: Column labels	Store_Nbr
Parameter: Row labels	PRODUCT_DESC
Parameter: Values	SUM(POS_Sales)
Parameter: Max number of columns to create	500

In the above transformation:

- The Column labels entry specifies the column whose values make up the calculated columns of the pivot table. The calculation is performed *across each* of these values. In this case, each column contains calculations for separate store numbers.
- The Row labels entry specifies the column whose values define the grouping of the calculations. In this case, the sum of the sales column is performed for each product description value for each store.
- The Values entry specifies the aggregation function to compute for each cell in the new table. In this case, you are generating the sum of sales for each product description in each store.
- By default, this transformation generates a maximum of 50 new columns. However, if the column used for your Column labels contains more than 50 values, you may want to raise this value.

NOTE: Avoid creating datasets wider than 2500 columns. Very wide datasets can cause performance degradation.

Results:

PRODUCT_DESC	sum_POS_Sales_1	sum_POS_Sales_2	sum_POS_Sales_3	sum_POS_Sales_7	sum_POS_Sales_9
ACME LAWN GARDEN BAG CLEAR	70	0	0	0	0
ACME COOKIES CHOC CHIP	0	10.62	0	20	90
ACME SANDWICH BAG	0	0	0	0	0
ACME SODAS SALTED	0	7.08	0	0	0
ACME SCENTED OIL REFILL-CTRY SUN	0	3.92	3.16	0	0
ACME LARGE FUDGE GRAHAMS COOKIES	0	13.44	0	0	0
ACME SUGAR ICE WAFERS VANILLA	0	0	0	0	0
ACME ZOO ANIMAL FRUIT SNACKS 6'S	0	0	3.16	0	0
ACME WAFERS SUGER ICE	0	0	3.16	0	0
ACME RICE CRACKERS ONION	0	0	6.32	0	0
ACME FROSTED OATMEAL COOKIE SQUA	0	0	0	190	150
ACME FRUIT SNACK CASTLE ADVENTRS	0	0	0	0	3.5

ACME ASSORTED COOKIES DRP	0	0	0	120	30
ACME KITCHEN BAG	0	0	0	120	70
ACME SNACK BAGS RESEALABLE	0	0	0	90	170
ACME CHEDDARY SN CRACKERS /PROCES	0	0	0	9.5	20
ACME RICE CRACKERS TERIYAKI	0	0	0	0	6.5
ACME COOKIE MAPLE LEAF CREME	0	0	0	40	1.5
ACME RICE CHIPS CHEDDAR	0	0	0	0	30
ACME DIGESTIVE RICH TEA BISCUITS	0	0	0	10	0
ACME FUDGE MINT COOKIES SQUARES	0	0	0	10	0
ACME COOKIES MAPLE CREAM	0	0	0	10	0

Conditional Aggregations

Suppose you are interested in only in the sum of sales for store numbers 1-3. To capture a more limited dataset, you can use the `SUMIF` aggregation function:

Transformation Name	Pivot table
Parameter: Row labels	PRODUCT_DESC
Parameter: Values	SUMIF(POS_Sales, Store_Nbr<4)
Parameter: Max number of columns to create	500

Most aggregation functions have a conditional (`*IF`) variant.

Multiple Aggregation Levels

None of the axes of a pivot table is limited to a single dimension. You can have multiple Column labels, Row labels, and Values (formulas). In the following transformation, aggregations have been further broken out by date, and an additional formula (Value) has been added.

NOTE: Adding multiple Column labels and Values can greatly expand the width of the dataset. Generally, adding Row labels does not expand the total count of rows.

Transformation Name	Pivot table

Parameter: Column labels	Store_Nbr
Parameter: Row labels1	Date
Parameter: Row labels2	PRODUCT_DESC
Parameter: Values1	SUM(POS_Qty)
Parameter: Values2	SUM(POS_Sales)
Parameter: Max number of columns to create	500

Results:

NOTE: Following results table is incomplete. Some columns have been omitted for space reasons.

Daily	PRODUCT_DESC	sum_POS_Qty_1	sum_POS_Sales_1	sum_POS_Qty_2	sum_POS_Sales_2	sum_POS_Qty
2/8/13	ACME LAWN GARDEN BAG CLEAR	7	70	0	0	0
2/7/13	ACME COOKIES CHOC CHIP	0	0	9	10.62	0
2/7/13	ACME SANDWICH BAG	0	0	0	0	0
2/7/13	ACME SODAS SALTED	0	0	6	7.08	0
2/7/13	ACME SCENTED OIL REFILL-CTRY SUN	0	0	2	3.92	2
2/7/13	ACME LARGE FUDGE GRAHAMS COOKIES	0	0	7	13.44	0
2/7/13	ACME SUGAR ICE WAFERS VANILLA	0	0	0	0	0
2/7/13	ACME ZOO ANIMAL FRUIT SNACKS 6'S	0	0	0	0	2
2/7/13	ACME WAFERS SUGER ICE	0	0	0	0	2
2/7/13	ACME RICE CRACKERS ONION	0	0	0	0	4
2/2/13	ACME FROSTED OATMEAL COOKIE SQUA	0	0	0	0	0
2/2/13	ACME FRUIT SNACK CASTLE ADVENTRS	0	0	0	0	0
2/2/13	ACME COOKIES CHOC CHIP	0	0	0	0	0
2/2/13	ACME ASSORTED COOKIES DRP	0	0	0	0	0
2/2/13	ACME KITCHEN BAG	0	0	0	0	0
2/2/13	ACME SNACK BAGS RESEALABLE	0	0	0	0	0

2/2/13	ACME CHEDDARY SN CRACKERS /PROCES	0	0	0	0	0
2/2/13	ACME RICE CRACKERS TERIYAKI	0	0	0	0	0
2/2/13	ACME COOKIE MAPLE LEAF CREME	0	0	0	0	0
2/2/13	ACME RICE CHIPS CHEDDAR	0	0	0	0	0
2/1/13	ACME FROSTED OATMEAL COOKIE SQUA	0	0	0	0	0
2/1/13	ACME COOKIES CHOC CHIP	0	0	0	0	0
2/1/13	ACME DIGESTIVE RICH TEA BISCUITS	0	0	0	0	0
2/1/13	ACME ASSORTED COOKIES DRP	0	0	0	0	0
2/1/13	ACME KITCHEN BAG	0	0	0	0	0
2/1/13	ACME SNACK BAGS RESEALABLE	0	0	0	0	0
2/1/13	ACME FUDGE MINT COOKIES SQUARES	0	0	0	0	0
2/1/13	ACME CHEDDARY SN CRACKERS /PROCES	0	0	0	0	0
2/1/13	ACME COOKIES MAPLE CREAM	0	0	0	0	0
2/1/13	ACME COOKIE MAPLE LEAF CREME	0	0	0	0	0

Group By

If you wish to maintain the original dataset values, you can apply an aggregate function within a single column.

Values to Columns

Similar to pivot, the Convert values to columns transformation converts individual values within a column to independent columns in the dataset. For each row, if the value represented by the column is present in the original data, one value is added (e.g. Yes). If it's missing, another value is inserted (e.g. No).

Tip: This type of conversion can be useful for preparing data for machine learning systems. You can convert the presence or absence of specific values in a row to 1 or 0, respectively.

In the following, the values in the `Store_Nbr` column have been converted to individual columns:

Transformation Name	Convert values to columns
Parameter: Column	Store_Nbr

Parameter: Fill when present	Yes
Parameter: Max number of columns to create	250

In the above:

- Fill when present identifies the string literal value to insert if the row contains the column's value (Yes).
- Fill when missing identifies the string literal value to insert if the row does not contain the column's value (empty).
- Max number of columns to create places a limit on the total number of columns that the application is permitted to create. In this case, the limit is set to 250 since the known number of stores is 250.

Tip: It's a good habit to set limits on the maximum number of columns to create. Data can become sparse or unwieldy if limits are not considered.

Results:

Daily	Store_Nbr	POS_Sales	POS_Qty	POS_Cost	PRODUCT_DESC	column_1	column_2	column_3	column_4
2/8/13	1	70	7	4.97	ACME LAWN GARDEN BAG CLEAR	Yes			
2/7/13	2	10.62	9	8.37	ACME COOKIES CHOC CHIP		Yes		
2/7/13	2	0	0	0	ACME SANDWICH BAG		Yes		
2/7/13	2	7.08	6	5.58	ACME SODAS SALTED		Yes		
2/7/13	2	3.92	2	2.82	ACME SCENTED OIL REFILL-CTRY SUN		Yes		
2/7/13	2	13.44	7	10.36	ACME LARGE FUDGE GRAHAMS COOKIES		Yes		
2/7/13	2	0	0	0	ACME SUGAR ICE WAFERS VANILLA		Yes		
2/7/13	3	3.16	2	2.86	ACME ZOO ANIMAL FRUIT SNACKS 6'S			Yes	
2/7/13	3	3.16	2	2.78	ACME WAFERS SUGER ICE			Yes	
2/7/13	3	3.16	2	2.82	ACME SCENTED OIL REFILL-CTRY SUN			Yes	
2/7/13	3	6.32	4	5.92	ACME RICE CRACKERS ONION			Yes	
2/2/13	9	150	30	16.2	ACME FROSTED OATMEAL COOKIE SQUA				Yes
2/2/13	9	3.5	2	4.86	ACME FRUIT SNACK CASTLE ADVENTRS				Yes
2/2/13	9	90	9	8.37	ACME COOKIES CHOC CHIP				Yes
2/2/13	9	30	6	3.24	ACME ASSORTED COOKIES DRP				Yes

2/2/13	9	70	7	6.51	ACME KITCHEN BAG				Yes
2/2/13	9	170	17	15.81	ACME SNACK BAGS RESEALABLE				Yes
2/2/13	9	20	4	2.16	ACME CHEDDARY SN CRACKERS /PROCES				Yes
2/2/13	9	6.5	2	8.98	ACME RICE CRACKERS TERIYAKI				Yes
2/2/13	9	1.5	3	1.62	ACME COOKIE MAPLE LEAF CREME				Yes
2/2/13	9	30	6	3.24	ACME RICE CHIPS CHEDDAR				Yes
2/1/13	7	190	38	20.52	ACME FROSTED OATMEAL COOKIE SQUA				
2/1/13	7	20	2	1.86	ACME COOKIES CHOC CHIP				
2/1/13	7	10	1	0.82	ACME DIGESTIVE RICH TEA BISCUITS				
2/1/13	7	120	24	12.96	ACME ASSORTED COOKIES DRP				
2/1/13	7	120	12	11.16	ACME KITCHEN BAG				
2/1/13	7	90	9	8.37	ACME SNACK BAGS RESEALABLE				
2/1/13	7	10	1	0.71	ACME FUDGE MINT COOKIES SQUARES				
2/1/13	7	9.5	19	10.26	ACME CHEDDARY SN CRACKERS /PROCES				
2/1/13	7	10	1	0.82	ACME COOKIES MAPLE CREAM				
2/1/13	7	40	8	4.32	ACME COOKIE MAPLE LEAF CREME				

Unpivot Columns

Contents:

- *Single-column Unpivot*
- *Multi-column Unpivot*
 - *Ranges*
 - *Wildcards*

You can convert columns into rows of values. A conversion transformations extracts the values from a specified column or columns and turns the column name and each extracted value into key-value pairs.

- Unpivot can be applied to one or more columns.
- Often, this transformation is applied to datasets containing pivoted or aggregated data.

NOTE: Depending on the number of source columns, an unpivot operation can significantly increase the number of rows in your dataset.

Single-column Unpivot

When you unpivot a single column of data, the column is separated into two new columns in your dataset:

New column name	Values
key	All values are the name of the source column.
value	Each row contains one of the row values from the source column.

NOTE: These columns replace the source column in the dataset. To retain the source column, create a copy of it first and then unpivot the copied column.

Source:

The following example contains a very simple set of data:

Name	favoriteColor	favoriteDessert
Anna	red	ice cream
Bella	pink	cookies
Callie	blue	pie

Transformation:

You can unpivot these columns one-by-one into row data:

Transformation Name	Unpivot columns
Parameter: Columns	favoriteColor
Parameter: Group size	1

Results:

The new unpivoted columns are placed at the end of the dataset, and the source column is removed.

Name	favoriteDessert	key	value
Anna	ice cream	favoriteColor	red
Bella	cookies	favoriteColor	pink
Callie	pie	favoriteColor	blue

Multi-column Unpivot

This example turns the data from multiple columns into a single set of key-value pairs, where the key is the column name associated with the source of the data in the value column.

Source:

The following dataset shows student test scores per test. Each row represents the scores of individual students.

StudentId	test1Score	test2Score	test3Score
001	75	79	77
002	84	81	86
003	79	82	87
004	92	94	92

Transformation:

You can use the following transformation to turn the dataset into one row per student-test combination:

Transformation Name	Unpivot columns
Parameter: Columns	test1Score, test2Score, test3Score
Parameter: Group size	1

Results:

The results are as follows:

StudentId	key	value
001	test1Score	75
002	test2Score	79
003	test3Score	77
001	test1Score	84
002	test2Score	81
003	test3Score	86
001	test1Score	79
002	test2Score	82
003	test3Score	87

001	test1Score	92
002	test2Score	94
003	test3Score	92

You can then rename the `key` and `value` columns as needed.

Ranges

You can specify a range of columns in your dataset. In the previous example, you can specify the three test score columns using the following value in the Columns textbox:

test1Score~test3Score

All three columns are unpivoted.

Wildcards

NOTE: You can use the asterisk (*) wildcard in the Columns textbox to apply the unpivot to the entire dataset, which generates a `key` and a `value` column, containing all column-row entries from the source columns. However, unpivoting a large number of columns can significantly increase the number of rows in your dataset.

Window Transformations

Contents:

- *Basic Structure*
 - *Group by parameter*
 - *Order by parameter*
 - *Compute over Time Windows*
 - *Calculate over preceding and following rows*
 - *Fill Empty Values*
 - *Calculate Rank*
 - *Calculate Rolling Functions*
 - *Rolling date functions*
-

A **window** transformation performs calculations on a row based on row values that are related to it. Windowing functions can perform calculations based on time, relative row positions, and rolling windows.

For example, you might wish to calculate the average percentage of CPU usage over 24-hour intervals based on log entries. From the rows of data, you can create a window function that calculates the average value in the CPU usage column over the 24-hour period, as defined based on date values for each log entry.

Key distinction:

- In a window function, the output of each row's calculation is specific to the row.
- In an aggregate function, the output for a row is the same value for all rows that are used in the calculation.
- For more information on aggregation, see *Create Aggregations*.

Basic Structure

You can use windowing functions with the following transformation types:

- window - creates a new column called `window`
- New formula - creates a new column that you name
- Edit with formula - modifies the values in a column based on a formula that you specify.

Group by parameter

You can use the Group by parameter to define the column of values by which rows of data are grouped for calculation purposes. For example, if your Group by column contains months, your calculations are computed for each month represented in the column values.

NOTE: Transforms that use the `group` parameter can result in non-deterministic re-ordering in the data grid. However, you should apply the `group` parameter, particularly on larger datasets, or your job may run out of memory and fail. To enforce row ordering, you can use the `sort` transform. For more information, see *Sort Transform*.

Order by parameter

When using window functions, you can use the Order by parameter to specify the column or columns by which to sort the output.

Source:

The following table contains the sales data of a company for all the four regions in the last three months.

Month	Sales	Region
2021-01-01	800	East
2021-01-01	1500	West
2021-01-01	1000	North
2021-01-01	2000	South
2021-02-01	1250	East
2021-02-01	800	West
2021-02-01	1100	North
2021-02-01	700	South
2021-03-01	900	East
2021-03-01	1000	West
2021-03-01	1400	North
2021-03-01	800	South

Transformation:

In the following transformation, you can calculate the rolling average of sales . You apply the `ROLLINGAVERAGE` and specify that the results are to be ordered by the Sales column.

Transformation Name	Window
Parameter: Formulas	<code>ROLLINGAVERAGE (Sales, 0,1)</code>
Parameter: Order by	Sales

Results:

The following dataset shows the `ROLLINGAVERAGE` ordered by Sales column.

Month	Sales	Region	RollingAverage
2021-02-01	700	South	750
2021-01-01	800	East	800
2021-02-01	800	West	800
2021-03-01	800	South	850
2021-03-01	900	East	950
2021-01-01	1000	North	1000
2021-03-01	1000	West	1050
2021-02-01	1100	North	1175
2021-02-01	1250	East	1325
2021-03-01	1400	North	1450
2021-01-01	1500	West	1750
2021-01-01	2000	South	2000

Compute over Time Windows

You may need to create windows of time within your data that are not cleanly segmented by basic units of time measurement. For example, you may need to create a custom time period, called a **session**, based on timestamps recorded in event-based data.

A session is usually defined as a group of events that occur within a given time frame. For example, you may need to perform calculations based on five-minute intervals within your logging data. If a user opens your shopping website, logs in, searches items, and then logs out within a five-minute interval, that can be grouped under a single session. However, if the user's interaction lasted six minutes, the logged events may span multiple windowed sessions in the data.

You can use the `SESSION` function to create time boxes based on a time period that you specify. When the function is applied to your column of timestamp values, the application assigns an ID to events that belong to the same session.

From the following example, you can create a Session ID. After you create the session ID, you can find the volume of data consumed by the individual user.

Source:

User Name	TimeStamp	Activity	Volume (in Kb)
Bob	02/11/21 08:01:13	Read	1024
William	02/11/21 08:01:00	Read	1024
John	02/11/21 08:01:17	Read	1024
Christy	02/11/21 08:01:17	Read	1024
William	02/11/21 08:03:33	Read	520
Christy	02/11/21 08:02:01	Password change	1024
Bob	02/11/21 08:07:23	Adding items to cart	2048
William	02/11/21 08:05:45	Read	520
William	02/11/21 08:11:56	Account settings	2048
John	02/11/21 08:15:11	Password change	2048
Bob	02/11/21 08:34:00	Proceeding to payment	2048
Bob	02/11/21 08:43:03	logout	2048
Christy	02/11/21 09:03:43	Read	1024
Christy	02/11/21 09:10:00	logout	1024

Transformation:

Transformation Name	Window
Parameter: Formulas	<code>SESSION (TimeStamp, 5, minute)</code>
Parameter: Group by	User Name
Parameter: Order by	TimeStamp

Since the new column is named `window`, you should rename it:

Transformation Name	Rename columns
Parameter: Option	Manual rename

Parameter: Column	window
Parameter: New column name	SESSIONID

With this session ID, you can calculate the maximum volume of data consumed by each session ID and by each user.

Transformation Name	New formula
Parameter: Formula type	Multiple row formula
Parameter: Formula	MAX(Volume (in Kb))
Parameter: Sort rows by	SessionID
Parameter: Group rows by	User Name, SessionID
Parameter: New column name	Volume_Consumed (in Kb)

Results:

User Name	TimeStamp	Activity	Volume (in Kb)	SessionID	max_Volume (in Kb)
William	02/11/21 08:01:00	Read	1024	1	1024
William	02/11/21 08:03:33	Read	520	1	1024
William	02/11/21 08:05:45	Read	520	1	1024
William	02/11/21 08:11:56	Account settings	2048	2	2048
Bob	02/11/21 08:01:13	Read	1024	1	1024
Bob	02/11/21 08:07:23	Adding items to cart	2048	2	2048
Bob	02/11/21 08:34:00	Proceeding to payment	2048	3	2048
Bob	02/11/21 08:43:03	logout	2048	4	2048
Christy	02/11/21 08:01:17	Read	1024	1	1024
Christy	02/11/21 08:02:01	Password change	1024	1	1024
Christy	02/11/21 09:03:43	Read	1024	2	1024
Christy	02/11/21 09:10:00	logout	1024	3	1024
John	02/11/21 08:01:17	Read	1024	1	1024
John	02/11/21 08:15:11	Password change	2048	2	2048

Calculate over preceding and following rows

The `PREV` and `NEXT` functions enable you to fetch data from a previous row or a subsequent row, which is helpful for identifying relative changes or trends in your data.

Source:

The following dataset contains orders for different product types over a given time period. You can apply the `PREV` and `NEXT` functions to calculate the previous orders and the next orders to analyze the trend of orders and derive the average of orders for a product group.

--	--	--

Product_Type	Order_date	Order
Laptop	2021-01-05	300
Laptop	2021-01-26	1780
Laptop	2021-01-09	500
Laptop	2021-01-31	1200
SmartPhone	2021-01-24	1400
SmartPhone	2021-01-26	2200
SmartPhone	2021-01-07	700
Tablet	2021-01-21	600
Tablet	2021-01-23	900

Transformation:

You can also calculate the percentage of change in orders over time. The following transformation calculates the change between the current order and the previous one and then divides that value over the previous value to calculate the percent change between the rows:

Transformation Name	Window
Parameter: Formulas	$(\text{Order} - \text{PREV}(\text{Order}, 1)) / \text{PREV}(\text{Order}, 1) * 100$
Parameter: Group by	Product_Type
Parameter: Order by	Order

After you rename the column to ChangeinOrder, you can apply the `NUMFORMAT` function to clean up and format the ChangeinOrder values. The following transformation reformats the ChangeinOrder column to display two decimal places.

Transformation Name	Edit with formula
Parameter: Column	ChangeinOrder
Parameter: Formula	<code>NUMFORMAT(ChangeinOrder, '##.##')</code>

Similarly, you can apply the `NEXT` function and calculate the Change in orders for upcoming months.

Results :

Product_Type	Order_date	Order	NEXTOrder	ChangeinOrder
Laptop	2021-01-05	300	500	
Laptop	2021-01-09	500	1200	66.67
Laptop	2021-01-31	1200	1780	140
Laptop	2021-01-26	1780		48.33
SmartPhone	2021-01-07	700	400	
SmartPhone	2021-01-24	1400	2200	100
SmartPhone	2021-01-26	2200		57.14
Tablet	2021-01-21	600	900	
Tablet	2021-01-23	900		50

Fill Empty Values

You can use the `FILL` function to fill empty or null values in your data with the last non-empty value in the group.

Source:

For example, the following dataset contains the daily orders received. Note the missing values due to weekends. You can assume that the no orders were received for Saturday and Sunday ,

Date	DayOfWeek	OrdersDay	OrdersTotal
2021-03-10	Wednesday	100	100
2021-03-11	Thursday	112	212
2021-03-12	Friday	320	532
2021-03-13	Saturday		
2021-03-14	Sunday		
2021-03-15	Monday	300	832

Transformation:

You have to clean up the data to fill the values for OrdersDay column. You can use the following function to fill the empty and null values. This function tests the the OrdersDay column to check if the column is empty or null. If so, the value ' 0 ' is written in the column, else the value of the column (`$col`) is written.

Transformation Name	Edit with formula
Parameter: Column	OrdersDay
Parameter: Formula	<code>IF(OrdersDay == '' ISNULL(OrdersDay), '0', \$col)</code>

You can see the values of Friday is taken for Saturday and Sunday and filled it accordingly as per the `FILL` function.

Transformation Name	Edit with formula
Parameter: Column	OrdersTotal
Parameter: Formula	<code>IF (OrdersDay == '0', FILL (OrdersTotal, -1,0),\$col)</code>
Parameter: Order by	Date

Results:

Date	DayOfWeek	OrdersDay	OrdersTotal
2021-03-10	Wednesday	100	100
2021-03-11	Thursday	112	212
2021-03-12	Friday	320	532
2021-03-13	Saturday	0	532
2021-03-14	Sunday	0	532
2021-03-15	Monday	300	832

Calculate Rank

The **RANK** function enables you to create rankings in your data based on calculations by returning a ranking value for each row with the specified group of values. When used, some rows might receive the same value as other rows. For example, if there are three tie values in a group, the same rank is assigned to the rows and the next three ranks are skipped.

The **DENSERANK** function enables you to generate a ranked order of values within a group. If there are tie values in a group, it does not skip rank in case of tie values. For example, if two rows are listed as rank 2, then the fourth row receives rank 3.

Source:

The following dataset contains total Sales information by quarter. You can use the **RANK** and **DENSERANK** to identify the quarters with the highest sales.

Year	Quarter	Sales
2018	1	1000
2018	2	2000
2018	3	3000
2018	4	2000
2019	1	1000
2019	2	500
2019	3	9000
2019	4	3000
2020	1	500
2020	2	500
2020	3	200
2020	4	400

Transformation:

RANK:

Transformation Name	Window
Parameter: Formula type	Multiple row formula
Parameter: Formula	RANK()
Parameter: Sort rows by	Sales
Parameter: New column name	SalesRank

DENSERANK:

Transformation Name	Window
Parameter: Formula type	Multiple row formula
Parameter: Formula	DENSERANK()

Parameter: Sort rows by	Sales
Parameter: New column name	SalesDenseRank

Results:

For the RANK function, when multiple rows share the same rank, the next rank is not consecutive, whereas for the DENSERANK function, the next rank is consecutive.

Year	Quarter	Sales	SalesDenseRank	SalesRank
2020	3	200	1	1
2020	4	400	2	2
2020	2	500	3	3
2020	1	500	3	3
2019	2	500	3	3
2019	1	1000	4	6
2018	1	1000	4	6
2018	4	2000	5	8
2018	2	2000	5	8
2019	4	3000	6	10
2018	3	3000	6	10
2019	3	9000	7	12

Calculate Rolling Functions

Rolling calculations enable you to compute a function over a changing set of rows. Rolling calculations are useful for computing the current state of a measure within your data.

For example, in the above sample data, you can find the rolling sum and rolling average of the sales for the year. You can use the above example data to find the rolling sum and rolling average.

Source:

From the following dataset, you can calculate the rolling calculations such as ROLLINGSUM, ROLLINGAVERAGE, ROLLINGMAX, and ROLLINGMIN.

Year	Quarter	Sales
2018	1	1000
2018	2	2000
2018	3	3000
2018	4	2000
2019	1	1000
2019	2	500
2019	3	9000
2019	4	3000
2020	1	500

2020	2	500
2020	3	200
2020	4	400

Transformation:

Transformation Name	Window
Parameter: Formulas	ROLLINGSUM (Sales, 0,1)
Parameter: Formulas	ROLLINGAVERAGE (Sales, 0,1)
Parameter: Formulas	ROLLINGMAX (Sales, 0, 1)
Parameter: Formulas	ROLLINGMIN (Sales, 0,1)
Parameter: Order by	Sales

You can rename the required columns accordingly.

Results:

Year	Quarter	Sales	RollingSumSales	RollingAverageSales	RollingMinSales	RollingMaxSales
2020	3	200	600	300	200	400
2020	4	400	900	450	400	500
2020	2	500	1000	500	500	500
2020	1	500	1000	500	500	500
2019	2	500	1500	750	500	1000
2019	1	1000	2000	1000	1000	1000
2018	1	1000	3000	1500	1000	2000
2018	4	2000	4000	2000	2000	2000
2018	2	2000	5000	2500	2000	3000
2019	4	3000	6000	3000	3000	3000
2018	3	3000	12000	6000	3000	9000
2019	3	9000	9000	9000	9000	9000

Rolling date functions

The Rolling date functions enable you to calculate forward or backward of the current row within the specified column. For example, when dealing with business calendars, you might want to know if the date falls on a holiday or weekend; based on that, you can roll the date forward or backward according to the business calendar.

Source:

The following example dataset shows the order date, order quantity that belongs to a product group. You are interested in finding the rolling minimum and maximum dates for the product group, as well as the rolling mode value. You can use ROLLINGMINDATE , ROLLINGMAXDATE , and ROLLINGMODEDATE functions.

Order_date	Order_quantity	Product_Group
2021-04-14	750	PG001
2021-07-13	1500	PG001

2021-08-31	355	PG002
2021-02-16	2000	PG002
2021-05-13	867	PG002
2021-06-18	1010	PG002
2021-11-15	909	PG003
2021-10-16	200	PG003
2021-09-09	200	PG004
2021-01-01	900	PG004
2021-12-07	707	PG004

Transformation:

Transformation Name	Window
Parameter: Formulas	ROLLINGSUM (Sales, 0,1)
Parameter: Formulas	ROLLINGMAXDATE (Order_date, 0,1)
Parameter: Formulas	ROLLINGMINDATE (Order_date, 0, 1)
Parameter: Formulas	ROLLINGMODEDATE (Order_date, 0,1)
Parameter: Order by	Order_date

Results:

Order_date	Order_quantity	Product_Group	RollingMaxdate	RollingMindate	RollingModedate
2021-01-01	900	PG004	2021-02-16	2021-01-01	2021-01-01
2021-02-16	2000	PG002	2021-04-14	2021-02-16	2021-02-16
2021-04-14	750	PG001	2021-05-13	2021-04-14	2021-04-14
2021-05-13	867	PG002	2021-06-18	2021-05-13	2021-05-13
2021-06-18	1010	PG002	2021-07-13	2021-06-18	2021-06-18
2021-07-13	1500	PG001	2021-08-31	2021-07-13	2021-07-13
2021-08-31	355	PG002	2021-09-09	2021-08-31	2021-08-31
2021-09-09	200	PG004	2021-10-16	2021-09-09	2021-09-09
2021-10-16	200	PG003	2021-11-15	2021-10-16	2021-10-16
2021-11-15	909	PG003	2021-12-07	2021-11-15	2021-11-15
2021-12-07	707	PG004	2021-12-07	2021-12-07	2021-12-07

Working with Arrays

Contents:

- *Array Types*
 - *Create Arrays*
 - *Create by extraction*
 - *Create by nesting*
 - *Create from column values*
 - *Create from Object type*
 - *Read from Arrays*
 - *Compute from Arrays*
 - *Combine Arrays*
 - *Break out Arrays*
 - *Expand arrays into rows*
 - *Unnest array elements into columns*
-

This section describes how to work with the Array data type in the Trifacta® application . An **array** is a set of delimited values. Any individual value in the list can be a separate array, which allows for the creation of nested data arrays.

Array Types

To be recognized as an array, a source column must contain values that are:

- Bracketed by square brackets
- Values in cell are delimited by commas

Such columns are likely to be recognized as Array data type.

The following are valid arrays:

```
[1,2,3]
["A","B"]
["C",["D","E"],"F",["G",["H","I"]]]
```

- **Ragged arrays:** If the number of elements varies between two arrays, they are considered ragged. In the above, all three arrays have a different number of top-level elements (3,2,4).
- **Nested arrays:** When an array element is an array itself, the element is considered a nested array. See the last example above.

For more information, see *Array Data Type*.

Create Arrays

Within Trifacta®, you can generate arrays using values from one or more columns to do so.

Create by extraction

You can create an array of values by extracting pattern-based values from a specified column. The following transformation extracts from the `msg` column a list of all values where all letters are capitalized and places them into the new `acronyms` column:

Transformation Name
Extract matches into Array

Parameter: Column	msg
Parameter: Pattern matching elements in the list	`{upper}+`
Parameter: New column name	acronyms

msg	acronyms
SCUBA, IMHO, is the greatest sport in the world.	["SCUBA","IMHO"]
	[]
LOL, that assignment you finished is DOA. You need to fix it PDQ.	["LOL","DOA","Y","PDQ"]

Notes:

- An empty input column value renders an empty array.
- In the final row, the Pattern matches on the "Y" value. To fix this, you can change the Pattern matching value to the following, which matches on two or more uppercase letters in a row:

```
`{upper}{upper}+`
```

Create by nesting

You can create arrays by nesting together the values from multiple columns.

Source:

num1	num2	num3
11	12	13
14	15	16
17	18	19

You want to nest the values in num1 and num2 into a single array and then to nest the array with num3:

NOTE: If you are nesting a multi-level array, you should nest from the lowest level to the top level.

Transformation Name	Nest columns into Objects
Parameter: Columns1	num1
Parameter: Columns2	num2
Parameter: Nest columns to	Array
Parameter: New column name	nest1

Then, you can perform the nesting of the top-level elements:

NOTE: The order in which you list the columns to nest determines the order in which the elements appear in the generated array.

Transformation Name	Nest columns into Objects
Parameter: Columns1	nest1
Parameter: Columns2	num3
Parameter: Nest columns to	Array
Parameter: New column name	nest2

In the generated columns, you notice that all values are quoted, even though these values are integers.

NOTE: Elements that are generated into arrays using a nest transformation are always rendered as quoted values.

You can use the following transformation to remove the quotes from the `nest2` column:

Transformation Name	Replace text or patterns
Parameter: Column	nest2
Parameter: Find	' '
Parameter: Replace	(empty)
Parameter: Match all occurrences	true

After removing the unused `nest1` column, the data looks like the following:

num1	num2	num3	nest2
11	12	13	[[11,12],13]
14	15	16	[[14,15],16]
17	18	19	[[17,18],19]

Create from column values

You can use one of several available functions to create arrays from a column's values.

Source:

listVals
5
TRUE
{"key1":"value1","keys2":"value2"}
[1,2,3]
My String
-5.5

The following transformation generates a new column in which each row contains an array of all of the values of the input column:

--	--

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	LIST(listVals,1000)
Parameter: New column name	listOfListVals

Results:

listVals	listOfListVals
5	["5","TRUE",{"key1\":"value1\","keys2\":"value2\"},"[1,2,3],"My String",-5.5]
TRUE	["5","TRUE",{"key1\":"value1\","keys2\":"value2\"},"[1,2,3],"My String",-5.5]
{"key1\":"value1","keys2\":"value2"}	["5","TRUE",{"key1\":"value1\","keys2\":"value2\"},"[1,2,3],"My String",-5.5]
[1,2,3]	["5","TRUE",{"key1\":"value1\","keys2\":"value2\"},"[1,2,3],"My String",-5.5]
My String	["5","TRUE",{"key1\":"value1\","keys2\":"value2\"},"[1,2,3],"My String",-5.5]
-5.5	["5","TRUE",{"key1\":"value1\","keys2\":"value2\"},"[1,2,3],"My String",-5.5]

Notes:

- The second parameter on the LIST function defines the maximum number of values to write. 1000 is the default.
- All values in the generated array are written as String values.
- Quoted values are escaped in the output.

The following functions allow you to generate various types of arrays from a column's set of values.

Function	Description
<i>LIST Function</i>	Extracts the set of values from a column into an array stored in a new column. This function is typically part of an aggregation.
<i>UNIQUE Function</i>	Extracts the set of unique values from a column into an array stored in a new column. This function is typically part of an aggregation.
<i>LISTIF Function</i>	Returns list of all values in a column for rows that match a specified condition.
<i>ROLLINGLIST Function</i>	Computes the rolling list of values forward or backward of the current row within the specified column and returns an array of these values.
<i>RANGE Function</i>	Computes an array of integers, from a beginning integer to an end (stop) integer, stepping by a third parameter. NOTE: The lower bound of the range is included, while the upper bound is not.

Tip: Additional examples are available in the above links for these functions.

Create from Object type

You can extract the keys of an Object column into an array of string values. In an Object type, the values are listed in quoted key/value pairs and can be nested. See *Object Data Type*.

Source:

Suppose your Object data looks like the following:

myObject
{"key1":"value1","key2":"value2","key3":"value3"}
{"apples":"2","oranges":"4" }
{"planes":{"boeing":"5","airbus":"4"},"trains":{"amtrak":"1","SP":"2"},"automobiles":{"toyota":"100","nissan":"50"}}

You can run the following transformation to extract the top-level keys into arrays in a new named column:

NOTE: The KEYS function retrieves only the top-level keys from the Object.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	KEYS(myObject)
Parameter: New column name	myObjectKeys

Results:

myObject	myObjectKeys
{"key1":"value1","key2":"value2","key3":"value3"}	["key1","key2","key3"]
{"apples":"2","oranges":"4"}	["apples","oranges"]
{"planes":{"boeing":"5","airbus":"4"},"trains":{"amtrak":"1","SP":"2"},"automobiles":{"toyota":"100","nissan":"50"}}	["planes","trains","automobiles"]

For more information, see *KEYS Function*.

Read from Arrays

You can read values from arrays in your dataset.

NOTE: After an array has been created, you can append to the array or otherwise combine it with another array. You cannot replace values in the array without breaking apart the array and rebuilding it.

Function	Description
<i>IN Function</i>	Returns <code>true</code> if the first parameter is contained in the array of values in the second parameter.
<i>ARRAYELEMENTAT Function</i>	Computes the 0-based index value for an array element in the specified column, array literal, or function that returns an array.
<i>ARRAYLEN Function</i>	Computes the number of elements in the arrays in the specified column, array literal, or function that returns an array.
<i>ARRAYUNIQUE Function</i>	Generates an array of all unique elements among one or more arrays.

Tip: Additional examples are available in the above links for these functions.

Compute from Arrays

You can use the following functions to perform computations on the values in your arrays:

Function	Description
<i>LISTSUM Function</i>	Computes the sum of all numeric values found in input array. Input can be an array literal, a column of arrays, or a function returning an array. Input values must be of Integer or Decimal type.
<i>LISTMAX Function</i>	Computes the maximum of all numeric values found in input array. Input can be an array literal, a column of arrays, or a function returning an array. Input values must be of Integer or Decimal type.
<i>LISTMIN Function</i>	Computes the minimum of all numeric values found in input array. Input can be an array literal, a column of arrays, or a function returning an array. Input values must be of Integer or Decimal type.
<i>LISTAVERAGE Function</i>	Computes the average of all numeric values found in input array. Input can be an array literal, a column of arrays, or a function returning an array. Input values must be of Integer or Decimal type.
<i>LISTVAR Function</i>	Computes the variance of all numeric values found in input array. Input can be an array literal, a column of arrays, or a function returning an array. Input values must be of Integer or Decimal type.
<i>LISTSTDEV Function</i>	Computes the standard deviation of all numeric values found in input array. Input can be an array literal, a column of arrays, or a function returning an array. Input values must be of Integer or Decimal type.
<i>LISTMODE Function</i>	Computes the most common value of all numeric values found in input array. Input can be an array literal, a column of arrays, or a function returning an array. Input values must be of Integer or Decimal type.

Combine Arrays

You can combine arrays together using a variety of methods of combining.

Source:

array1	array2
["1","2","3"]	["A","B","C"]
["4","5","6"]	["D","E","F"]
["7","8","9"]	["G","H","I"]

The following transformation concatenates the above arrays into a single single array:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	ARRAYCONCAT([array1,array2])
Parameter: New column name	arrayConcat

Results:

array1	array2	arrayConcat
["1","2","3"]	["A","B","C"]	["1","2","3","A","B","C"]
["4","5","6"]	["D","E","F"]	["4","5","6","D","E","F"]
["7","8","9"]	["G","H","I"]	["7","8","9","G","H","I"]

These functions can be used to combine arrays together:

Function	Description
<i>ARRAYCONCAT Function</i>	Combines the elements of one array with another, listing all elements of the first array before listing all elements of the second array.
<i>ARRAYCROSS Function</i>	Generates a nested array containing the cross-product of all elements in two or more arrays.
<i>ARRAYINTERSECT Function</i>	Generates an array containing all elements that appear in multiple input arrays, referenced as column names or array literals.
<i>ARRAYSTOMAP Function</i>	Combines one array containing keys and another array containing values into an Object of key-value pairs.
<i>ARRAYZIP Function</i>	Combines multiple arrays into a single nested array, with element 1 of array 1 paired with element 2 of array 2 and so on. Arrays are expressed as column names or as array literals.

Tip: Additional examples are available in the above links for these functions.

Break out Arrays

Expand arrays into rows

You can break out arrays into individual values using the following transformations. Here is some example data from the `nest2` column that was generated earlier. The `num3` column is retained for reference:

num3	nest2
13	[[11,12],13]
16	[[14,15],16]
19	[[17,18],19]

You can use the following simple transformation to flatten the values in `nest2` into individual values in each row:

NOTE: Depending on the number of elements in your arrays, you can significantly increase the size of your dataset.

NOTE: If a cell in the source column does not contain an array, an empty value is written into the corresponding row.

Transformation Name	Expand Array to rows
Parameter: column	<code>nest2</code>

Results:

num3	nest2
13	[11,12]

13	13
16	[14,15]
16	16
19	[17,18]
19	19

NOTE: Converting a column of arrays to rows unpacks the top level of the array only. You may have to apply this transformation multiple times.

Unnest array elements into columns

You can break out individual elements of an array into separate columns.

NOTE: Each element that you want broken out into a column must be listed on a separate line in Path to elements.

Source:

arrayNested
["A",["B","C"],"D"]
["H",["I","J"],["K","L"]]
["E","F","G"]

The following transform retrieves the second and third elements of each array:

Transformation Name	Unnest Objects into columns
Parameter: Column	arrayNested
Parameter: Paths to elements1	[1]
Parameter: Paths to elements2	[2]
Parameter: Include original column name	true

This one retrieves the first element of the array that is nested as the second element of the array:

Transformation Name	Unnest Objects into columns
Parameter: Column	arrayNested
Parameter: Paths to elements1	[1][0]
Parameter: Include original column name	true

The resulting data should look like the following:

arrayNested	arrayNested_1	arrayNested_2
["A",["B","C"],"D"]	["B","C"]	B
["H",["I","J"],["K","L"]]	["I","J"],["K","L"]	I

["E","F","G"]	F	
---------------	---	--

Working with Objects

Contents:

- *Structure of Objects*
 - *Import Objects*
 - *Import Object columns*
 - *Import JSON files*
 - *Create Objects*
 - *Create by nesting*
 - *Create by Filtering Strings*
 - *Convert from Arrays*
 - *Read from Objects*
 - *Extract Keys*
 - *Extract Object Values*
 - *Convert to String*
 - *Unnest Objects*
-

This section describes how to work with the Object data type. An **object** (or **map**) is a set of key-value pairs.

Any individual value can contain another set of key-value pairs, which enables the creation of nested data objects.

Tip: As one of its values, an object can contain an array, which can in turn contain other objects or arrays. In this manner, you can create nested hybrid data objects by combining these two data types.

Structure of Objects

An **Object** data type is a method for encoding key-value pairs. A single field value may contain one or more sets of key-value pairs. A simple example:

```
{ "Texas": "TX" },  
{ "New York": "NY" },  
{ "California": "CA" },
```

Notes:

- The above example features repeated data in a repeated format across each line.
- Effectively, these are records of data, mapping a state's formal name (e.g. `Texas`) to its two-letter abbreviation (e.g. `TX`).
- Data structures of the Object data type can be more complex.

NOTE: The Trifacta application can recognize up to 250 unique keys in a column of Object data type.

Import Objects

Import Object columns

When a column is identified as a set of key-value pairs during import, the column may be typed as an Object data type column. These key-value pairs can be extracted and converted into rows and columns in the dataset using transformations and functions in the application.

Import JSON files

The Object data type can be the basis for entire JSON files. When JSON files are formatted in a way that can be parsed by the Trifacta application, they can be converted into tabular format as part of the import process. If the preceding example is the entire file, the conversion process may display the dataset in the Transformer page as the following:

column1	column2
Texas	TX
New York	NY
California	CA

For more information, see *Working with JSON v2*.

Create Objects

Within the Trifacta application, you can use functions and transformations to create columns that are recognized as Object data type.

Create by nesting

You can nest multiple columns into a single column of objects using the `nest` transform.

This section provides a simple example of nesting columns into a new column of Object data type.

Source:

In the following example, furniture product dimensions are stored in separate columns in cm.

Category	Name	Length_cm	Width_cm	Height_cm
bench	Hooska	118.11	74.93	46.34
lamp	Tansk	30.48	30.48	165.1
bookshelf	Brock	27.94	160.02	201.93
couch	Loafy	95	227	83

Transformation:

Use the `nest` transform to bundle the data into a single column.

Transformation Name	Nest columns into Objects
Parameter: Columns	Length_cm,Width_cm,Height_cm
Parameter: Nest columns to	Object
Parameter: New column name	'Dimensions'

Results:

Category	Name	Length_cm	Width_cm	Height_cm	Dimensions
bench	Hooska	118.11	74.93	46.34	{"Length_cm":"118.11","Width_cm":"74.93","Height_cm":"46.34"}
lamp	Tansk	30.48	30.48	165.1	{"Length_cm":"30.48","Width_cm":"30.48","Height_cm":"165.1"}

bookshelf	Brock	27.94	160.02	201.93	{"Length_cm":"27.94","Width_cm":"160.02","Height_cm":"201.93"}
couch	Loafy	95	227	83	{"Length_cm":"95","Width_cm":"227","Height_cm":"83"}

Create by Filtering Strings

You can create objects by filtering strings by using the `FILTEROBJECT` function.

You can create nested objects by filtering strings. In this example, column headers and column values are nested into a single entity in a new column of Object data type.

Functions:

Item	Description
FILTEROBJECT Function	Filters the keys and values from an Object data type column based on a specified key value.
PARSEOBJECT Function	Evaluates a String input against the Object datatype. If the input matches, the function outputs an Object value. Input can be a literal, a column of values, or a function returning String values.

Source:

The following table shows a series of requests for inventory on three separate products. These are rolling requests, so inventory levels in the subsequent request are decreased based on the previous request.

date	reqProdId	reqValue	prodA	prodB	prodC
5/10/21	prodA	10	90	100	100
5/10/21	prodC	20	90	100	80
5/10/21	prodA	15	75	100	80
5/11/21	prodB	25	75	75	80
5/11/21	prodA	5	70	75	80
5/11/21	prodC	30	70	75	50
5/12/21	prodB	10	70	65	50

You must create a column containing the request information and the inventory level information for the requested product after the request has been fulfilled.

Transformation:

The five data columns must be nested into an Object. The generated column is called `inventoryLevels`.

Transformation Name	Nest columns into Objects
Parameter: Columns	reqProdId, reqValue, prodA, prodB, prodC
Parameter: Nest columns to	Object
Parameter: New column name	inventoryLevels

You can then build the inventory response column (`inventoryResponse`) using the `FILTEROBJECT` function:

Transformation Name	New formula
Parameter: Formula type	Single row formula

Parameter: Formula	<code>filterobject(parseobject(inventoryRequest), ['reqProdId','reqValue',reqProdId])</code>
Parameter: New column name	<code>inventoryResponse</code>

Results:

The `inventoryResponse` column contains the request information and the response information after the request has been fulfilled.

date	reqProdId	reqValue	prodA	prodB	prodC	inventoryLevels	inventoryResponse
5/10/21	prodA	10	90	100	100	{"reqProdId":"prodA","reqValue":"10","prodA":"90","prodB":"100","prodC":"100"}	{"reqProdId":"prodA","reqValue":"10","prodA":"90"}
5/10/21	prodC	20	90	100	80	{"reqProdId":"prodC","reqValue":"20","prodA":"90","prodB":"100","prodC":"80"}	{"reqProdId":"prodC","reqValue":"20","prodC":"80"}
5/10/21	prodA	15	75	100	80	{"reqProdId":"prodA","reqValue":"15","prodA":"75","prodB":"100","prodC":"80"}	{"reqProdId":"prodA","reqValue":"15","prodA":"75"}
5/11/21	prodB	25	75	75	80	{"reqProdId":"prodB","reqValue":"25","prodA":"75","prodB":"75","prodC":"80"}	{"reqProdId":"prodB","reqValue":"25","prodB":"75"}
5/11/21	prodA	5	70	75	80	{"reqProdId":"prodA","reqValue":"5","prodA":"70","prodB":"75","prodC":"80"}	{"reqProdId":"prodA","reqValue":"5","prodA":"70"}
5/11/21	prodC	30	70	75	50	{"reqProdId":"prodC","reqValue":"30","prodA":"70","prodB":"75","prodC":"50"}	{"reqProdId":"prodC","reqValue":"30","prodC":"50"}
5/12/21	prodB	10	70	65	50	{"reqProdId":"prodB","reqValue":"10","prodA":"70","prodB":"65","prodC":"50"}	{"reqProdId":"prodB","reqValue":"10","prodB":"65"}

Convert from Arrays

You can create objects by converting two arrays of key value pairs by using the `ARRAYSTOMAP` function.

This example illustrates how to use the `ARRAYSTOMAP` and `KEYS` functions to convert values in Array or Object data type of key-value pairs.

Functions:

Item	Description
ARRAYSTOMAP Function	Combines one array containing keys and another array containing values into an Object of key-value pairs.
KEYS Function	Extracts the key values from an Object data type column and stores them in an array of String values.

Source:

Your dataset contains master product data with product properties stored in two arrays of keys and values.

ProdId	ProdCategory	ProdName	ProdKeys	ProdProperties
S001	Shirts	Crew Neck T-Shirt	["type", "color", "fabric", "sizes"]	["crew", "blue", "cotton", "S,M,L", "in stock", "padded"]

S002	Shirts	V-Neck T-Shirt	["type", "color", "fabric", "sizes"]	["v-neck", "white", "blend", "S,M,L,XL", "in stock", "discount - seasonal"]
S003	Shirts	Tanktop	["type", "color", "fabric", "sizes"]	["tank", "red", "mesh", "XS,S,M", "discount - clearance", "in stock"]
S004	Shirts	Turtleneck	["type", "color", "fabric", "sizes"]	["turtle", "black", "cotton", "M,L,XL", "out of stock", "padded"]

Transformation:

When the above data is loaded into the Transformer page, you might need to clean up the two array columns.

Using the following transform, you can map the first element of the first array as a key for the first element of the second, which is its value. You might notice that the number of keys and the number of values are not consistent. For the extra elements in the second array, the default key of `ProdMiscProperties` is used:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	ARRAYSTOMAP(ProdProperties, ProdValues, 'ProdMiscProperties')
Parameter: New column name	'prodPropertyMap'

You can now use the following steps to generate a new version of the keys:

Transformation Name	Delete columns
Parameter: Columns	ProdKeys
Parameter: Action	Delete selected columns

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	KEYS(prodPropertyMap)
Parameter: New column name	'ProdKeys'

Results:

ProdId	ProdCategory	ProdName	ProdKeys	ProdProperties	prodPropertyMap
S001	Shirts	Crew Neck T-Shirt	["type", "color", "fabric", "sizes", "ProdMiscProperties"]	["crew", "blue", "cotton", "S,M,L", "in stock", "padded"]	{ "type": ["crew"], "color": ["blue"], "fabric": ["cotton"], "sizes": ["S,M,L"], "ProdMiscProperties": ["in stock", "padded"] }
S002	Shirts	V-Neck T-Shirt	["type", "color", "fabric", "sizes", "ProdMiscProperties"]	["v-neck", "white", "blend", "S,M,L,XL", "in stock", "discount - seasonal"]	{ "type": ["v-neck"],

					<pre> "color": ["white"], "fabric": ["blend"], "sizes": ["S", "M", "L", "XL"], "ProdMiscProperties": ["in stock", "discount - seasonal"] } </pre>
S003	Shirts	Tanktop	["type", "color", "fabric", "sizes", "ProdMiscProperties"]	["tank", "red", "mesh", "XS, S, M", "discount - clearance", "in stock"]	<pre> { "type": ["tank"], "color": ["red"], "fabric": ["mesh"], "sizes": ["XS", "S", "M"], "ProdMiscProperties": ["discount - clearance", "in stock"] } </pre>
S004	Shirts	Turtleneck	["type", "color", "fabric", "sizes", "ProdMiscProperties"]	["turtle", "black", "cotton", "M, L, XL", "out of stock", "padded"]	<pre> { "type": ["turtle"], "color": ["black"], "fabric": ["cotton"], "sizes": ["M", "L", "XL"], "ProdMiscProperties": ["out of stock", "padded"] } </pre>

Read from Objects

When a column is recognized as an Object data type, you can apply transformations to extract the keys, the values, or both from the column for use in a new column. You can use pattern-based matching to acquire the values of interest for further analysis or cleaning.

Extract Keys

You can extract keys from objects from the Object data and stores them in an array of String values.

You can extract the keys from an Object column into an Array of String values.

Functions:

Item	Description
KEYS Function	Extracts the key values from an Object data type column and stores them in an array of String values.

Source:

The following dataset contains configuration blocks for individual features, each of which has a different configuration. These example blocks are of Object type.

--

Tip: In the following example configuration, the **keys** are the values on the left (e.g. `enabled`, `maxRows`, and `maxCols`), while the **values** for those keys are on the right side.

Code formatting has been applied to the Object data to improve legibility.

FeatureName	Configuration
Whiz Widget	<pre>{ "enabled": "true", "maxRows": "1000", "maxCols": "100" }</pre>
Magic Button	<pre>{ "enabled": "false", "startDirectory": "/home", "maxDepth": "15" }</pre>
Happy Path Finder	<pre>{ "enabled": "true" }</pre>

Transformation:

The following transformation extracts the keys from the Object data in the `Configuration` column.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>keys(Configuration)</code>
Parameter: New column name	<code>'keys_Configuration'</code>

Results:

The `keys_Configuration` column contains the arrays of the key values.

FeatureName	Configuration	keys_Configuration
Whiz Widget	<pre>{ "enabled": "true", "maxRows": "1000", "maxCols": "100" }</pre>	<pre>["enabled", "maxRows", "maxCols"]</pre>
Magic Button	<pre>{ "enabled": "false", "startDirectory": "/home", "maxDepth": "15" }</pre>	<pre>["enabled", "startDirectory", "maxDepth"]</pre>

Happy Path Finder	<pre>{ "enabled": "true" }</pre>	<pre>["enabled"]</pre>
-------------------	------------------------------------	------------------------

Extract Object Values

You can extract an object's values in to a new column.

This simple example demonstrates how to extract nested values from Object elements into a separate column.

Source:

For example, suppose your restaurant dataset includes a set of characteristics in the `restFeatures` column in the following JSON format, from which you are interested in the total number of seats in the restaurant.

This example contains the data for a single restaurant, formatted as regular JSON, for simplicity:

```
{
  "Credit": "Y",
  "Accessible": "Y",
  "Restrooms": "Y",
  "EatIn": "Y",
  "ToGo": "N",
  "AlcoholBeer": "Y",
  "AlcoholHard": "N",
  "TotalTables": "10",
  "TotalTableSeats": "36",
  "Counter": "Y",
  "CounterSeats": "8"
}
```

Transformation:

You can use the following transformation to extract the values from `TotalTableSeats` and `CounterSeats` into separate columns:

NOTE: Change the column type to Object before applying the following transformation.

NOTE: Each key must be entered on a separate line in the Path to elements area.

Transformation Name	Unnest Objects into columns
Parameter: Column	restFeatures
Parameter: Paths to elements	TotalTableSeats
Parameter: Paths to elements	CounterSeats
Parameter: Include original column name	Selected

Results:

restFeatures_TotalTable Seats	restFeatures_CounterSeats
-------------------------------	---------------------------

After converting into separate columns, you can perform a simple sum of the `TotalTableSeats` and `CounterSeats` columns to determine the total number of seats in the restaurant.

The final table looks like the following:

<code>restFeatures_TotalTableSeats</code>	<code>restFeatures_CounterSeats</code>	<code>TotalSeats_Restaurant</code>
36	8	44

Convert to String

Depending on the use, it may be easier to work with your objects as String values. While Strings have no inherent structure, they do have a wide range of functions that you can use to find and extract information from the values. Some useful functions:

Function	Description
<i>FIND Function</i>	Returns the index value in the input string where a specified matching string is located in provided column, string literal, or function returning a string. Search is conducted left-to-right.
<i>RIGHTFIND Function</i>	Returns the index value in the input string where the last instance of a matching string is located. Search is conducted right-to-left.
<i>FINDNTH Function</i>	Returns the position of the nth occurrence of a letter or pattern in the input string where a specified matching string is located in the provided column. You can search either from left or right.
<i>SUBSTITUTE Function</i>	Replaces found string literal or pattern or column with a string, column, or function returning strings.

Unnest Objects

You can unnest the object data type to create new rows or columns based on the keys in the source data. The following example shows how to unnest object values into separate columns.

This example shows how you can unpack data nested in an Object into separate columns.

Source:

You have the following information on used cars. The `VIN` column contains vehicle identifiers, and the `Properties` column contains key-value pairs describing characteristics of each vehicle. You want to unpack this data into separate columns.

VIN	Properties
XX3 JT4522	year=2004,make=Subaru,model=Impreza,color=green,mileage=125422,cost=3199
HT4 UJ9122	year=2006,make=VW,model=Passat,color=silver,mileage=102941,cost=4599
KC2 WZ9231	year=2009,make=GMC,model=Yukon,color=black,mileage=68213,cost=12899
LL8 UH4921	year=2011,make=BMW,model=328i,color=brown,mileage=57212,cost=16999

Transformation:

Add the following transformation, which identifies all of the key values in the column as beginning with alphabetical characters.

- The `valueafter` string identifies where the corresponding value begins after the key.
- The `delimiter` string indicates the end of each key-value pair.

Transformation Name	Convert keys/values into Objects
Parameter: Column	Properties
Parameter: Key	`{alpha}+`
Parameter: Separator between key and value	`=`
Parameter: Delimiter between pair	`,`

Now that the Object of values has been created, you can use the `unnest` transform to unpack this mapped data. In the following, each key is specified, which results in separate columns headed by the named key:

NOTE: Each key must be entered on a separate line in the Path to elements area.

Transformation Name	Unnest Objects into columns
Parameter: Column	extractkv_Properties
Parameter: Paths to elements	year
Parameter: Paths to elements	make
Parameter: Paths to elements	model
Parameter: Paths to elements	color
Parameter: Paths to elements	mileage
Parameter: Paths to elements	cost

Results:

When you delete the unnecessary Properties columns, the dataset now looks like the following:

VIN	year	make	model	color	mileage	cost
XX3 JT4522	2004	Subaru	Impreza	green	125422	3199
HT4 UJ9122	2006	VW	Passat	silver	102941	4599
KC2 WZ9231	2009	GMC	Yukon	black	68213	12899
LL8 UH4921	2011	BMW	328i	brown	57212	16999

Working with JSON v2

Contents:

- *Enable*
 - *Requirements*
 - *JSON input*
 - *JSON output*
 - *Example*
 - *Example 1 - Rows of JSON records*
 - *Example 2 - Top-level array of JSON records*
-

Version 2: This section describes how you can import JSON files into Trifacta®, convert them to tabular format, wrangle them, and then export them back in the same JSON format.

The basic workflow is described by way of example. In the example workflow, the JSON file must be imported into Trifacta®, a new column must be inserted into the JSON, and the resulting JSON must be exported in the same structure.

Enable

This method of working with JSON is enabled by default.

You can choose to continue using the legacy method of working with JSON.

NOTE: The legacy version of JSON import is required if you are working with compressed JSON files or only Newline JSON files.

You should migrate your flows to using the new version.

NOTE: The legacy version of working with JSON is likely to be deprecated in a future release.

For more information on migrating to the new version, see *Working with JSON v1*.

Requirements

JSON input

- Recommended limit of 1 GB in source file size. Since conversion happens within the Trifacta node, this limit may vary depending on the memory of the Trifacta node.
- Each JSON record must be less than 20 MB in size.

NOTE: This maximum record length can be modified. For more information, see *Configure Application Limits*.

- Filename extensions must be `.json` or `.JSON`.
- Conversion of compressed JSON files is not supported. Compressed JSON files can be imported using the previous method.
- For best results, all keys and values should be quoted and imported as strings.

NOTE: Escape characters that make JSON invalid can cause your JSON file to fail to import.

- You can escape quote values to treat them as literals in your strings using the backslash character. For example: \"
- When the values are imported into the Transformer page, the Trifacta application re-infers the data type for each column.

JSON structure	Description	Supported?
Newline	The newline character (\n) denotes the end of a record. Each record can contain the keys (object or array) and values for the JSON object. Tip: This version is supported through both versions of JSON import, but it performs better in v1. If you are using the Newline form of JSON exclusively, you should use v1.	Supported
Top-level object	Top-level row contains keys for mapping JSON objects	Supported
Top-level array	Top-level row contains array of objects	Supported

JSON output

NOTE: JSON-formatted files that are generated by Trifacta are rendered in JSON Lines format, which is a single line per-record variant of JSON. For more information, see <http://jsonlines.org>.

- Trifacta can generate a JSON file as an output for your job. Characteristics of generated JSON files:
 - **Newline-delimited:** The end of each record is the \n character. If your downstream system is expecting comma-delineated records except for the last one, additional work is required outside of the application.
 - **Non-nested:** Each record in the generated file is flat.
 - For multi-level JSON hierarchies, you can nest columns together and leave the top level as a set of columns in the data grid. However, on output, the second and lower hierarchies appear as quoted string values in the output. Additional cleanup is required outside of the application.

Example

Example 1 - Rows of JSON records

The following example contains records of images from a website:

```
{
  "metrics": [
    {
      "rank": "1043",
      "score": "9679"
    }
  ],
  "caption": "Such a good boy!",
  "id": "9kt8ex",
  "url": "https://www.example.com/w285fpp11.jpg",
  "filename": "w285fpp11.jpg"
},
{
  "metrics": [
    {
      "rank": "1042",
      "score": "9681"
    }
  ],
  "caption": "This sweet puppy has transformed our life!",
  "id": "9x2774",
  "url": "https://www.example.com/fml10cy11.jpg",
  "filename": "fml10cy11.jpg"
},
{
  "metrics": [
    {
      "rank": "1041",
      "score": "9683"
    }
  ],
  "caption": "We sure love our fur babies.",
  "id": "a8guou",
  "url": "https://www.example.com/mljnmq521.jpg",
  "filename": "mljnmq521.jpg"
}
```

Notes:

- Each row is a complete JSON record containing keys and values.

Tip: Nested JSON, such as `metrics` above, can be inserted as part of a record. It can then be unnested within the application.

- Each key's value must have a comma after it, except for the final key value in any row.

NOTE: The end of a JSON record is the right curly bracket (`}`). Commas are not added to the end of each line in this format.

Workflow

1. Import the JSON file.
2. Any nested data must be unnested within columns. Each level in the JSON hierarchy must be un-nested in a separate step.
3. When all of the JSON data is in tabular form, perform any `Wrangle` transformations.
4. If you need to rebuild the loose JSON hierarchy, you must nest the lower levels of the JSON hierarchy back into their original form.
 - a. If it is ok to write out flat JSON records, you can export without nesting the data again.
5. Run the job, generating a JSON output.

Step - Import the file

1. Through the Import Data page, navigate and select your JSON file for import.

NOTE: File formats are detected based on the file extension. Please verify that your file extension is `.json` or `.JSON`, which ensures that it is passed through the conversion service.

- a. The file is passed through the conversion process, which reviews the JSON file and stores it on the base storage layer in a format that can be easily ingested as in row-per-record format. This process happens within the Import Data page. You can track progress on the right side of the screen.
2. After the file has been converted, click the Preview icon on the right side of the screen. In the Preview, you can review the first few rows of the imported file.
 - a. If some rows are missing from the preview, then you may have a syntax error in the first row after the last well-structured row. You should try to fix this in source and re-import.
 - b. If all of the rows are problematic, your data is likely malformed.
 3. Complete the rest of the import process.
 4. In Flow View, add the JSON-based imported dataset to your flow and create a recipe for it.
 - a. Select the recipe, and click **Edit Recipe...**

In the Transformer page, the example above should look like the following:

metrics	caption	id	url	filename
[{"rank": "1043", "score": "9679"}]	Such a good boy!	9kt8ex	https://www.example.com/w285fpp11.jpg	w285fpp11.jpg
[{"rank": "1042", "score": "9681"}]	This sweet puppy has transformed our life!	9x2774	https://www.example.com/fmll0cy11.jpg	fmll0cy11.jpg
[{"rank": "1041", "score": "9683"}]	We sure love our fur babies.	a8guou	https://www.example.com/mljnmq521.jpg	mljnmq521.jpg

Step - Unnest JSON records

Your JSON records are in tabular format. If you have nested JSON objects within your JSON records, the next step is to unnest your JSON records.

NOTE: For JSON records that have multiple levels in the hierarchy, you should unnest the top level of the hierarchy first, followed by each successive level.

Tip: The easiest way to unnest is to select the column header for the column containing your nested data. Unnest should be one of the suggested options, and the suggestion should include the specification for the paths to the key values. If not, you can use the following process.

1. In the Recipe panel, click **New Step**.
2. In the Search panel, enter `unnest values into new columns`.
3. Specify the following transformation. Substitute the Paths to elements values below with the top-level keys in your JSON records:

Transformation Name	<code>Unnest values into new columns</code>
Parameter: Column	<code>metrics</code>
Parameter: Path to elements1	<code>[0]</code>

Tip: You can choose to remove the original from the source or not. In deeper or wider JSON files, removing can help to identify what remains to be unnested.

4. In the above transformation, the bracketing array around the set of values has been broken down into raw JSON. This value may now be interpreted as a String data type. From the column drop-down, you can select Object data type.
5. Click the column head again, or specify the following transformation to unnest the Object column:

Transformation Name	<code>Unnest Objects into columns</code>
Parameter: Column	<code>0</code>
Parameter: Path to elements1	<code>rank</code>
Parameter: Path to elements2	<code>score</code>

- a. In the above, each Paths to elements entry specifies a key in the JSON record. The key's associated value becomes the value in the new column, which is given the same name as the key.
 - b. So, this step breaks out the key-value pairs for the specified keys into separate columns in the dataset.
6. Repeat the above process for the next level in the hierarchy.
 7. You can now delete the source columns. In the example, these source columns are named `metrics` and `0`.

Tip: SHIFT + click these columns and then select **Delete columns** from the right panel. Click **Add**.

8. Repeat the above steps for each nested JSON object.

Tip: If the above set of steps needs to be applied to multiple files, you might consider stopping your work and returning to Flow View. Select this recipe and click **Add New Recipe**. If you add successive steps in another recipe, the first one can be used for doing initial processing of your JSON files, separate from any wrangling that you may do for individual files.

Tip: The unnesting process may have moved some columns into positions that are different from their order in the original JSON. Use the **Move** command from the column menu to reposition your columns.

Step - Wrangle your dataset

Your JSON data is ready for wrangling. Continue adding steps until you have transformed your data as needed and are ready to run a job on it.

Step - Nest the JSON records

NOTE: If your desired JSON output does not include multiple hierarchies, you can skip this section. The generated JSON files are a single JSON record per row.

If you ran a job on the example dataset, the output would look like the following:

```
{ "rank":1043,"score":9679,"caption":"Such a good boy!","id":"9kt8ex","url":"https://www.example.com/w285fpp11.jpg","filename":"w285fpp11.jpg" }
{ "rank":1042,"score":9681,"caption":"This sweet puppy has transformed our life!","id":"9x2774","url":"https://www.example.com/fml10cy11.jpg","filename":"fml10cy11.jpg" }
{ "rank":1041,"score":9683,"caption":"We sure love our fur babies.","id":"a8guou","url":"https://www.example.com/mljnmq521.jpg","filename":"mljnmq521.jpg" }
```

Suppose you want to nest the `url` and `filename` columns into a nested array called, `resources`.

Re-nest the lower hierarchies until have you have a single flat record, containing some Object type columns that hold the underlying hierarchies. When the re-nested JSON records are exported, secondary hierarchies appear as escaped string values. More details later.

Tip: The following steps reshape your data. You may wish to create a new recipe as an output of the previous recipe where you can add the following steps.

Steps:

1. SHIFT + click the `url` and `filename` columns. Then, select **Nest columns** in the right-hand panel. This transformation should look like the following:

Transformation Name	Nest columns into Objects
Parameter: column1	url
Parameter: column2	filename
Parameter: Nest columns to	Object
Parameter: New column name	column1

2. `column1` now contains an Object mapping of the two columns. You can now nest this column again into an Array:

Transformation Name	Nest columns into Objects
Parameter: Columns	column1
Parameter: Nest columns to	Array
Parameter: New column name	resources

3. Delete column1.
4. Continue nesting other columns in a similar fashion. Repeat the above steps for the next level of the hierarchy in your dataset.
5. You must re-nest from the bottom of the target hierarchy to the top.

NOTE: Do not nest the columns at the top level of the hierarchy.

6. When the column names contain all of the keys that you wish to generate in the top-level JSON output, you can run the job.

Step - Generate JSON output

When you are ready, you can run the job. Create or modify a publishing action to generate a JSON file for output. See [Run Job Page](#).

When the job completes, you can click the JSON link in the Output Destinations tab of the Job Details page to download your JSON file. See [Job Details Page](#).

Output file for the above example should look like the following:

```
{
  "rank": 1043, "score": 9679, "caption": "Such a good boy!", "id": "9kt8ex", "url": "https://www.example.com/w285fpp11.jpg", "filename": "w285fpp11.jpg", "resources": [
    {
      "url": "https://www.example.com/w285fpp11.jpg", "filename": "w285fpp11.jpg"
    }
  ]
},
{
  "rank": 1042, "score": 9681, "caption": "This sweet puppy has transformed our life!", "id": "9x2774", "url": "https://www.example.com/fml10cy11.jpg", "filename": "fml10cy11.jpg", "resources": [
    {
      "url": "https://www.example.com/fml10cy11.jpg", "filename": "fml10cy11.jpg"
    }
  ]
},
{
  "rank": 1041, "score": 9683, "caption": "We sure love our fur babies.", "id": "a8guou", "url": "https://www.example.com/mljnmq521.jpg", "filename": "mljnmq521.jpg", "resources": [
    {
      "url": "https://www.example.com/mljnmq521.jpg", "filename": "mljnmq521.jpg"
    }
  ]
}
```

Example 2 - Top-level array of JSON records

Your JSON may be formatted as a single top-level object containing an array of JSON records. The following example contains records of messages about individual diet and exercise achievements:

```
{
  "object": [
    {
      "score": 19669,
      "title": "M/07/1'3\" [23lbs > 13lbs = 10lbs] Still a bit to go, but my owner no longer refers to me as his chunky boy!",
      "ups": 19669,
      "id": "9kt8ex",
      "url": "https://i.redd.it/bzygw285fpp11.jpg",
      "short": "bzygw285fpp11.jpg"
    },
    {
      "score": 19171,
      "title": "M/29/5'11\" [605 pounds > 375 pounds = 230 pounds lost] (14 months) Still considered super morbidly obese but I've made some good progress.",
      "ups": 19171,
      "id": "9x2774",
      "url": "https://i.redd.it/wbbufml10cy11.jpg",
      "short": "wbbufml10cy11.jpg"
    }
  ]
}
```

```

    "short": "wbbufm1l0cy11.jpg"
  },
  {
    "score": 16778,
    "title": "F/28/5\u201911 [233lbs to 130lbs] Got tired of being obese and took control of my life!",
    "ups": 16778,
    "id": "a8guou",
    "url": "https://i.redd.it/3t0kmljnmq521.jpg",
    "short": "3t0kmljnmq521.jpg"
  },
  {
    "score": 16743,
    "title": "M/22/5'11\" [99lbs > 150lbs = 51lbs] Anorexia my recovery",
    "ups": 16743,
    "id": "atla3n",
    "url": "https://i.redd.it/9t6tvsjs16i21.jpg",
    "short": "9t6tvsjs16i21.jpg"
  }
]
}

```

The outer JSON is a single key-value pair:

- key: `object`
- value: array of JSON records

When source JSON records structured in this manner are imported, each JSON record in the object is imported into a separate row. You can unnest this data by applying an Unnest values transformation.

NOTE: The object can contain only one nested array of JSON data. If the object contains multiple nested arrays, it is not broken into separate rows. All unnesting must be performed in your recipe steps

Suppose you want to compute the average of all workout scores. First, you must unnest the JSON records and then apply the `AVERAGE` function.

Steps:

Tip: The easiest way to unnest is to select the column header for the column containing your data. After you select the column header, you are provided with suggestions to Unnest Values into new columns. You can use the Unnest suggestion and click **Add**. The following steps illustrate how to create this transformation manually.

1. In the Recipe panel, click **New Step**.
2. In the Search panel, enter `unnest values into new columns`.
3. Specify the following transformation. Substitute the Paths to elements values below with the top-level keys in your JSON records:

Transformation Name	Unnest values into new columns
Parameter: Column	<code>object</code>
Parameter: Path to elements	<code>id</code>
Parameter: Path to elements	<code>score</code>
Parameter: Path to elements	<code>short</code>
Parameter: Path to elements	<code>title</code>
Parameter: Paths to elements	<code>ups</code>

Parameter: Path to elements	url
------------------------------------	-----

- The above step breaks out the key-value pairs for the specified keys into separate columns in the dataset. Each Paths to elements entry specifies a key in the JSON record, which is used to create a new column of the same name. The key's associated value becomes a cell value in the new column.
- You can now delete the source column. In the example, the source column is `object`.

Tip: You can choose to remove the original from the source or not. In deeper or wider JSON files, removing can help to identify what remains to be unnested. When you're done unnesting a column and have removed data from the original, you should have an empty column.

Results:

id	score	short	title	ups	url
9kt 8ex	19669	bzygw285fpp11.jpg	M/07/1'3" [23lbs > 13lbs = 10lbs] Still a bit to go, but my owner no longer refers to me as his chunky boy!	19669	https://i.redd.it/bzygw285fpp11.jpg
9x2 774	19171	wbbufmll0cy11.jpg	M/29/5'11" [605 pounds > 375 pounds = 230 pounds lost] (14 months) Still considered super morbidly obese but I've made some good progress.	19171	https://i.redd.it/wbbufmll0cy11.jpg
a8g uou	16778	3t0kmljnmq521.jpg	F/28/5'7" [233lbs to 130lbs] Got tired of being obese and took control of my life!	16778	https://i.redd.it/3t0kmljnmq521.jpg
atla 3n	16743	9t6tvsjs16i21.jpg	M/22/5'11" [99lbs > 150lbs = 51lbs] Anorexia my recovery	16743	https://i.redd.it/9t6tvsjs16i21.jpg

Now you can find the average score by applying average function.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	AVERAGE(score)
Parameter: New column name	Average_score

Results:

id	score	short	title	ups	url	Average_score
9kt 8ex	19669	bzygw285fpp11.jpg	M/07/1'3" [23lbs > 13lbs = 10lbs] Still a bit to go, but my owner no longer refers to me as his chunky boy!	19669	https://i.redd.it/bzygw285fpp11.jpg	18090.25
9x2 774	19171	wbbufmll0cy11.jpg	M/29/5'11" [605 pounds > 375 pounds = 230 pounds lost] (14 months) Still considered super morbidly obese but I've made some good progress.	19171	https://i.redd.it/wbbufmll0cy11.jpg	18090.25
a8g uou	16778	3t0kmljnmq521.jpg	F/28/5'7" [233lbs to 130lbs] Got tired of being obese and took control of my life!	16778	https://i.redd.it/3t0kmljnmq521.jpg	18090.25
atla 3n	16743	9t6tvsjs16i21.jpg	M/22/5'11" [99lbs > 150lbs = 51lbs] Anorexia my recovery	16743	https://i.redd.it/9t6tvsjs16i21.jpg	18090.25

Working with JSON v1

Contents:

- *Enable*
 - *Migrate to v2*
- *JSON Input and Output*
- *Example*
- *JSON Workflow*
 - *Step - Import the file*
 - *Step - Convert to one JSON record per row*
 - *Step - Convert JSON to Object type*
 - *Step - Unnest JSON records*
 - *Step - Wrangle your dataset*
 - *Step - Nest the JSON records*
 - *Step - Generate JSON output*
 - *Step - Final Cleanup*

Version 1: This section describes how you can import JSON files into Trifacta®, convert them to tabular format, wrangle them, and then export them back in the same JSON format.

The basic workflow is described by way of example. In the example workflow, the JSON file must be imported into Trifacta®, a new column must be inserted into the JSON, and the resulting JSON must be exported in the same structure.

Enable

This legacy method of working with JSON is likely to be deprecated in a future release.

If you have existing flows that were created using this legacy method, they should continue to work as expected. However, you should migrate your flows to use the newer version as soon as possible. See [Migrate](#) below.

NOTE: The legacy version of JSON import is required if you are working with compressed JSON files or only Newline JSON files.

This method can be enabled through workspace settings. For more information, see [Workspace Settings Page](#).

Migrate to v2

Any flow that you have created using the v1 version of the JSON importer should work without modification.

In the future, the v1 version will be deprecated. You can use the following method to migrate your flows to use the new version of the JSON importer.

NOTE: The v1 version of JSON import is supported for imported datasets. If you use these datasets in other workflows, they are likely to require modifications that you have done in recipes.

Basic workflow:

This migration workflow creates new versions of these imported datasets and fixes recipes accordingly.

1. Through the Library page, locate the imported datasets that are based on JSON files.
 - a. You may be able to just search for `json`.
2. For each JSON imported dataset:
 - a. Click the link.
 - b. In the Dataset Details page, copy the value for the Location. Paste it into a text file.
 - c. In the Dataset Details page, locate flow or flows where the dataset is in use.

Tip: If you copy the link address of the flow and paste it into a text file, you can paste that later into a browser and jump directly to the flow.

- d. Repeat the above steps for each JSON-based imported dataset.
3. You should now have a list of links to the source data and the flows where your JSON imported datasets are in use.
4. In the Library page, create a new version of each imported dataset:
 - a. Click **Import Data**.
 - b. Click the appropriate connection.
 - c. Paste the link to the Location where the source is stored.
 - d. The data is ingested through the conversion service.

Tip: Click the icon for the dataset in the right panel. All rows in the Preview panel should be properly structured. Nested data may not be broken out into separate columns at this time.

- e. Rename the dataset as needed.

Tip: You should give each new version of the imported dataset a consistent prefix or suffix tag, such as `-v2`. Later, you can locate these new imported datasets easily through search in the Library.

- f. Click **Continue**.
5. Repeat the above steps for each imported dataset that you are updating to v2.
6. For each of these flows:
 - a. Navigate to it.
 - b. Locate the v1 imported dataset in it. You might copy the name.
 - c. Click **Add Datasets**. Search for the v2 imported dataset. Add it to the flow.
7. In Flow View:
 - a. Click the recipe that is in use with the v1 version of the imported dataset. In the context menu in the right panel, select **Make a copy > without inputs**.
 - b. Select the copied recipe.
 - c. In the context menu in the right panel, select **Change input**. Select the v2 imported dataset.
 - d. Your v2 imported dataset is now connected to a version of your recipe.
 - e. Select the recipe object. In the right panel, you should see a preview of the recipe steps.

NOTE: In the recipe, the steps where you modified the imported dataset into tabular format are likely to be broken. This is ok.

8. Click **Edit recipe**.
9. In the Transformer page:
 - a. Disable recipe step 1.
 - b. Review the state of the data grid to see if the data is organized in tabular form.
 - c. If not, repeat the above steps for the next step in your recipe.

- d. Continue until the data is in tabular form.
10. After some additional tweaking, your recipe should contain no broken steps, and your data should appear in tabular form.
11. You may wish to run a job or download your sample data to compare it to outputs from your v1 imported dataset and steps. You may need to create an output object first.
12. You can now integrate these changes in either of the following ways:
 - a. **Apply to existing recipe:** Change the input on the existing to the v2 imported dataset. Apply any disabling of steps and other tweaks to the recipe's connected to the v1 imported dataset.

NOTE: Before applying the above changes, you might want to download the v1 recipe through the Recipe panel.

- b. **Use v2 recipe in the flow:** You could simply switch over to using the new recipe. Caveats:
 - i. You must recreate any outputs and schedules from the v1 recipe.
 - ii. Internal identifiers for the new recipe and its outputs are different from the v1 recipe. These new identifiers may impact API-based automation.
 - iii. Other application objects that reference the v1 recipes, such as flow tasks in your plans, must be fixed to use the new recipe or output objects.
13. Run a production job to verify that your flow is producing consistent data with the v2 imported dataset.
14. Repeat as needed for other flows.

JSON Input and Output

Input:

- It is easier to work with JSON in which each row of the file is a record. When a record spans multiple rows, additional steps are required in the application to render it into tabular format. The example uses multi-row JSON records.

Output:

NOTE: JSON-formatted files that are generated by Trifacta are rendered in JSON Lines format, which is a single line per-record variant of JSON. For more information, see <http://jsonlines.org>.

- Trifacta can generate a JSON file as an output for your job. Characteristics of generated JSON files:
 - **Newline-delimited:** The end of each record is the `\n` character. If your downstream system is expecting comma-delineated records except for the last one, additional work is required outside of the application.
 - **Non-nested:** Each record in the generated file is flat.
 - For multi-level JSON hierarchies, you can nest columns together and leave the top level as a set of columns in the data grid. However, on output, the second and lower hierarchies appear as quoted string values in the output. Additional cleanup is required outside of the application.

Example

This example dataset contains information on books. In this case:

- The data is submitted as one attribute per row. A single JSON record spans many rows.
- The total number of books is three.
- The JSON data has two hierarchies.

```
"book": {  
  "id": "bk101",  
  "author": "Guy, Joe",  
  "title": "Json Guide",  
  "genre": "Computer",
```

```

    "price": "44.95",
    "publish_date": "2002-04-26",
    "characteristics": {
      "cover_color": "black",
      "paper_stock": "20",
      "paper_source": "new"
    },
    "description": "An in-depth look at creating applications."
  },
  "book": {
    "id": "bk102",
    "author": "Nelson, Rogers",
    "title": "When Doves Cry",
    "genre": "Biography",
    "price": "24.95",
    "publish_date": "2016-04-21",
    "characteristics": {
      "cover_color": "white",
      "paper_stock": "15",
      "paper_source": "recycled"
    },
    "description": "Biography of a prince."
  },
  "book": {
    "id": "bk103",
    "author": "Fitzgerald, F. Scott",
    "title": "The Great Gatsby",
    "genre": "Fiction",
    "price": "9.95",
    "publish_date": "1925-04-10",
    "characteristics": {
      "cover_color": "blue",
      "paper_stock": "20",
      "paper_source": "new"
    },
    "description": "Classic American novel."
  }
}

```

JSON Workflow

1. Import the JSON file.

NOTE: During import, you should deselect the Detect Structure option. You are likely to need to rebuild the initial parsing steps to consume the file properly. Details are provided later.

2. If needed, convert loose JSON to a single JSON record per row.
3. Unnest the data into columns.
 - a. Each level in the JSON hierarchy must be un-nested in a separate step.
4. When all of the JSON data is in tabular form, perform any **Wrangle** transformations.
5. If you need to retain the hierarchy, you must nest the lower levels of the JSON hierarchy back into their original form. Leave the top level un-nested.
 - a. If it is ok to write out flat JSON records, you can export without nesting the data again.
6. Run the job, generating a JSON output.

Step - Import the file

1. Through the Import Data page, navigate and select your JSON file for import.
2. When the file has been loaded, click **Edit settings** for the dataset card in the right panel. In the Import Settings dialog, deselect the Detect Structure checkbox.
3. Complete the rest of the import process.
4. In Flow View, add the JSON-based imported dataset to your flow and create a recipe for it.
5. Select the recipe, and click **Edit Recipe....**

Step - Convert to one JSON record per row

NOTE: This step is required only if a single JSON record in your imported dataset spans multiple rows. If you have single-row JSON records in the Transformer page, please skip to the next section.

- 1. In the Transformer page, you should see your loosely formatted JSON in a single column. Each row contains a separate attribute, and a single record spans multiple rows.
- 2. Open the Recipe panel on the right side. The initial parsing steps for the data are displayed.
- 3. In Recipe panel, delete all steps except the first one.
- 4. The first one is a Break into rows transformation. This transformation can only appear in the first step of a recipe.
- 5. Select the step, and click the Pencil icon to edit it.
- 6. In the Transform Builder, the Split on value is probably the \n character.
- 7. The above signals to the application to break up the data into individual rows on the newline (\n) character. This transformation then breaks up your loose JSON on every single attribute. You must modify the Split on value so that it captures only the first attribute of each JSON record. For the above dataset, the Split on value must be the following, noting the space after the colon:

```
"book " :
```

- 8. Click **Add** to save the step again.
- 9. The above dataset should now have four rows, with the first one an empty row. This empty row is caused by the insertion of the \n in front of the first reference to the above string. In the column histogram, select the gray bar, which selects the empty row. In the Suggestions panel, locate the Delete rows suggest, and click **Add**. The row is removed.
- 10. You now have individual rows for each JSON record.

Step - Convert JSON to Object type

The next step involves converting your JSON records to a column of Object type values. The Object data type is a means of rendering records into key-value pairs. However, its structure is a little different from JSON. For more information, see *Object Data Type*.

Steps:

The following steps convert your JSON to an Object data type.

- 1. Since JSON uses character indentation to convey structure, you should remove these indentations if they appear in your dataset. For our two-layered example, you can use the following transformation:

Transformation Name	Replace text or patterns
Parameter: Column	column1
Parameter: Find	/\n\s*"/
Parameter: Replace with	\
Parameter: Match all occurrences	true

- a. In the above, the key term is the Find pattern, which is a **regular expression**:

```
/\n\s*"/
```

- b. The two forward slashes at the ends define the pattern as a regular expression.

- c. The content in the middle matches on the pattern of a newline character, an arbitrary number of spaces, and a double quote.
 - d. This pattern is replaced with just the double-quote, removing the preceding part of the pattern from the dataset.
 - e. For more information on matching patterns, see *Text Matching*.
2. In standard JSON, a comma is used to demarcate the end of a line or a record, except for the last one in a set.
- a. In the above example, the first two records have commas at the end of them. Here is a snippet of their ends:

```
... "description": "An in-depth look at creating applications."},
... "description": "Biography of a prince."},
... "description": "Classic American novel."}
```

- b. To convert these records to Object type, the commas at the end of the first two rows must be removed:

Transformation Name	Replace text or patterns
Parameter: Column	column1
Parameter: Find	`\n\}, \n{end}`
Parameter: Replace with	}
Parameter: Match all occurrences	true

- i. The above transformation is similar to the previous one. However, in this one, the Find pattern uses a `Pattern` to indicate that the pattern should only be matched at the end of a record:

```
{end}
```

- ii. This token in the pattern prevents it from matching if there are other instances of the pattern nested within the record.
3. Individual records should look similar to the following:

NOTE: Below, some values are too long for a single line. Single lines that overflow to additional lines are marked with a `\`. The backslash should not be included if the line is used as input.

```
{ "id": "bk101", "author": "Guy, Joe", "title": "Json Guide", "genre": "Computer", \
  "price": "44.95", "publish_date": "2002-04-26", {"cover_color": "black", \
  "paper_stock": "20", "paper_source": "new"}, \
  "description": "An in-depth look at creating applications." }
```

4. These records are suitable for conversion to Object data type.
5. To change the data type for the column, click the icon to the left of the column header. Select **Object**.
6. The column data type is changed to Object. The step to change data type is added to your recipe, too.
7. If the column histogram now displays some mismatched records.
 - a. Review those records to determine what is malformed.
 - b. Delete the recipe step that changes the data type to Object.
 - c. Make fixes as necessary.
 - d. Switch back to Object data type. Iterate as needed until all records are valid when the column is converted to Object type.

Step - Unnest JSON records

The next step is to convert your JSON records to tabular format.

NOTE: For JSON records that have multiple levels in the hierarchy, you should unnest the top level of the hierarchy first, followed by each successive level.

Tip: The easiest way to unnest is to select the column header for the column containing your Object data. Unnest should be one of the suggested options. If not, you can use the following process.

1. In the Recipe panel, click **New Step**.
2. In the Search panel, enter `unnest object elements`.
3. Specify the following transformation. Substitute the Paths to elements values below with the top-level keys in your JSON records:

Transformation Name	Unnest object elements
Parameter: Column	column1
Parameter: Path to elements1	id
Parameter: Path to elements2	author
Parameter: Path to elements3	title
Parameter: Path to elements4	genre
Parameter: Path to elements5	price
Parameter: Path to elements6	publish_date
Parameter: Path to elements7	description
Parameter: Remove elements from original	true

- a. In the above, each Paths to elements entry specifies a key in the JSON record. The key's associated value becomes the value in the new column, which is given the same name as the key.
- b. So, this step breaks out the key-value pairs for the specified keys into separate columns in the dataset.

Tip: You can choose to remove the original from the source or not. In deeper or wider JSON files, removing can help to identify what remains to be unnested.

4. Repeat the above process for the next level in the hierarchy. In the example, this step means unnesting the `characteristics` node:

Transformation Name	Unnest object elements
Parameter: Column	column1
Parameter: Path to elements1	<code>characteristics.cover_color</code>
Parameter: Path to elements2	<code>characteristics.paper_stock</code>

Parameter: Path to elements3	characteristics.paper_source
Parameter: Remove elements from original	true

5. You can now delete `column1`. From the column menu to the right of `column1`, select **Delete**.
6. You have now converted your JSON to tabular format.

Tip: If the above set of steps needs to be applied to multiple files, you might consider stopping your work and returning to Flow View. Select this recipe and click **Add New Recipe**. If you add successive steps in another recipe, the first one can be used for doing initial processing of your JSON files, separate from any wrangling that you may do for individual files.

Tip: The unnesting process may have moved some columns into positions that are different from their order in the original JSON. Use the **Move** command from the column menu to reposition your columns.

Step - Wrangle your dataset

Your JSON data is ready for wrangling.

In the following example, the `discount` column is created. If the publication date is before 01/01/2000, then the discount is 0.1 (10%):

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	IF(publish_date < DATE(2000, 1, 1), 0.1, 0)
Parameter: New column name	discount

Continue adding steps until you have transformed your data as needed and are ready to run a job on it.

Step - Nest the JSON records

NOTE: If your desired JSON output does not include multiple hierarchies, you can skip this section. The generated JSON files are a single JSON record per row.

If a job is run using the recipe created so far on the example data, a newline-delimited JSON file that has no hierarchies in it can be generated by the application. However, the dataset is a two-level hierarchy, so the elements in the `characteristics` hierarchy are written out in the following manner:

```
"characteristics.cover_color":"black","characteristics.paper_stock":20,"characteristics.paper_source":"new",
"characteristics.cover_color":"white","characteristics.paper_stock":15,"characteristics.paper_source":"
recycled",
"characteristics.cover_color":"blue","characteristics.paper_stock":20,"characteristics.paper_source":"new",
```

You can take one of two approaches:

1. Generate the JSON file with a flat hierarchy. Output looks like the above. Use an external tool to unnest the second and lower hierarchies appropriately.

2. Re-nest the lower hierarchies until you have a single flat record, containing some Object type columns that hold the underlying hierarchies. When the re-nested JSON records are exported, secondary hierarchies appear as escaped string values. More details later.

If you are re-nesting the lower hierarchies, you can use the following approach.

Tip: The following steps reshape your data. You may wish to create a new recipe as an output of the previous recipe where you can add the following steps.

1. When you re-nest, you want to nest from the lowest to top tier of the hierarchy.
2. In the example, the following columns should be nested together: `characteristics.cover_color`, `characteristics.paper_stock`, and `characteristics.paper_source`:

Transformation Name	Nest columns into Objects
Parameter: column1	<code>characteristics.cover_color</code>
Parameter: column2	<code>characteristics.paper_stock</code>
Parameter: column3	<code>characteristics.paper_source</code>
Parameter: Nest columns to	Object
Parameter: New column name	<code>characteristics</code>

3. In the generated `characteristics` column, you can remove the `characteristics.` from the key value:

Transformation Name	Replace text or patterns
Parameter: Column	<code>characteristics</code>
Parameter: Find	<code>`characteristics.`</code>
Parameter: Replace with	(empty)

4. Now, delete the three source columns:

Transformation Name	Delete columns
Parameter: column1	<code>characteristics.cover_color</code>
Parameter: column2	<code>characteristics.paper_stock</code>
Parameter: column3	<code>characteristics.paper_source</code>

5. Repeat the above steps for the next level of the hierarchy in your dataset.

NOTE: Do not nest the columns at the top level of the hierarchy.

Step - Generate JSON output

When you are ready, you can run the job. Create or modify a publishing action to generate a JSON file for output. See *Run Job Page*.

When the job completes, you can click the JSON link in the Output Destinations tab of the Job Details page to download your JSON file. See *Job Details Page*.

Step - Final Cleanup

Outside the application, you may need to do the following:

1. Since the JSON output is newline delimited, your downstream system may need you to add commas at the end of each record but the last one.
2. If you have re-nested JSON hierarchies into your flat records, the exported JSON for secondary hierarchies appears as quoted strings, like the following:

```
"characteristics":{"cover_color":"black","paper_stock":"20","paper_source":"new"},
"characteristics":{"cover_color":"white","paper_stock":"15","paper_source":"recycled"},
"characteristics":{"cover_color":"blue","paper_stock":"20","paper_source":"new"},
```

The quoted strings can be fixed by simple search and replace.

Cleanse Tasks

The following topics pertain to cleaning data that has been imported into Trifacta®.

Rename Columns

Contents:

- *Name Requirements*
 - *Reserved keywords*
- *Rename Individual Columns*
 - *Rename a column through column menu*
 - *Rename a column through suggestions*
 - *Rename a column through transformation*
 - *Rename a new column*
- *Auto-Generated Column Names*
- *Rename Multiple Columns*
 - *Manual rename multiple columns*
 - *Add prefix*
 - *Add suffix*
 - *Apply rename to all columns*
 - *Convert to lowercase*
 - *Convert to UPPERCASE*
 - *Keep from beginning (left)*
 - *Keep from end (right)*
 - *Find and replace*
 - *Use row(s) as column names*
 - *Combine multiple rows*

In the Trifacta® application, you can rename individual columns through the column drop-down. Through transform steps, you can apply renaming to one or more columns.

NOTE: An imported dataset requires about 15 rows to properly infer column data types and the row, if any, to use for column headers.

Name Requirements

- Column names are case-insensitive and cannot begin with whitespace.
- Column names cannot contain escaped characters, such as `\n`.

NOTE: When publishing to Avro, Parquet, or database tables, column names support alphanumeric characters and the underscore (`_`) character only. Column names cannot begin with a numeral. Other characters cause an error to occur.

NOTE: Column names with spaces or special characters in a transformation must be wrapped by curly braces. Example:

```
column1,{Column 2 with space},column3
```

Tip: To prevent potential issues with downstream systems, you should limit your column lengths to no more than 128 characters.

Reserved keywords

The following keywords should not be used as column names, as they may conflict with underlying requirements of the platform or the running environments with which it integrates:

NOTE: This list may not be complete. If your job fails with a duplicate column error, please review your column names to identify potential reserved keywords among them.

- TRIFACTA__LINEAGE_INFO
- TRIFACTA__FILE_LINEAGE_INFO

NOTE: There are two underscore characters in a row (__) after TRIFACTA in each of the above entries.

Rename Individual Columns

Rename a column through column menu

To rename a column, click the drop-down caret next to the column name. Click **Rename**.

Rename a column through suggestions

Steps:

1. If your column already exists, click the name of the column.
2. Click the Rename suggestion card.
3. Click **Modify**.
4. Replace the `newColumnName` value with your preferred column name.

Rename a column through transformation

You can use the following transformation to rename a single column through the Transform Builder. In this case, the Rename columns transformation is used to perform a manual rename of `MySourceCol` to `MyNewCol`.

Transformation Name	Rename columns
Parameter: Option	Manual rename
Parameter: Column	MySourceCol
Parameter: New name	MyNewCol

Rename a new column

Columns that are generated through transform steps are given a default name.

For the following types of transforms, however, you can specify the column name as part of the step:

- derive
- extractkv
- merge
- nest

When a transform is added to the recipe, an `as:` clause is automatically added to the transform step. You can modify your transform to change the value of the `as:` column. For example, the following transform generates a

new column with the first word from the Name column. The `as:` value renames this generated column as `FirstName`:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>FIND(Name,`{start} `,false,0)</code>
Parameter: New name	<code>FirstName</code>

Auto-Generated Column Names

When your transforms generate new columns, names are automatically assigned to these columns based on the following pattern.

1. If the transform includes a function reference, the function name is included in the new column. Example:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>LEFT(city,3)</code>

New column name: `left_city`

2. If the above step is applied again, a duplicate column is generated with the following name. Example:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>LEFT(city,3)</code>

New column name: `left_city1`

3. If the transform does not contain a function reference, the following convention is used:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>'A'</code>

New column name: `column1`

Transformation Name	New formula
Parameter: Formula	Single row formula

type	
Parameter: Formula	'B'

New column name: column2

Rename Multiple Columns

Trifacta enables to rename multiple columns using a single transformation. You can perform this batch renaming using one of the methods described in this section.

NOTE: In macros, Rename Columns transformations do not work. This is a known issue.

Tip: To prevent potential issues with downstream systems, you should limit your column lengths to no more than 128 characters.

Steps:

1. Open the Transform Builder to add a new step to your recipe.
2. From the drop-down in the first textbox, select `Rename columns`.
3. Select your method of renaming. See below.
4. Select the column or columns to which to apply the rename.

Tip: To apply the renaming across all columns in the dataset, select **All**. This option is useful for pattern-based renames, such as adding a prefix or changing case.

5. To add the step to your recipe, click **Add**.

Manual rename multiple columns

For each column that you select, you must add the new name just below the old one.

- To add additional columns to the mapping, click **Add**.
- To remove columns from the mapping, click **Remove**.

Add prefix

For the selected columns, you can apply a specific prefix value to the names.

Old Column Names	Prefix	New Column Names
column1	pre_	pre_column1
column2	pre_	pre_column2
column3	pre_	pre_column3

Transformation:

Transformation Name	Rename columns
Parameter: Option	Add prefix
Parameter: Column	column1, column2, column3

Parameter: Prefix	pre_
--------------------------	------

Add suffix

For the selected columns, you can apply a specific suffix value to the names. Example:

Old Column Names	Suffix	New Column Names
column1	_new	column1_new
column2	_new	column2_new
column3	_new	column3_new

Transformation:

Transformation Name	Rename columns
Parameter: Option	Add suffix
Parameter: Column	column1,column2,column3
Parameter: Suffix	_new

Apply rename to all columns

The following transformation performs the same rename as the previous one. Instead, it uses the All option to apply the rename across all columns of the dataset. If the number of columns changes in the future, then the rename is still applied across all of the columns in the dataset.

Transformation:

Transformation Name	Rename columns
Parameter: Option	Add suffix
Parameter: Columns	All
Parameter: Suffix	_new

Convert to lowercase

For the selected columns, you can convert the columns names to lowercase. Example:

Old Column Names	New Column Names
Daily	daily
POS_Cost	pos_cost
Sales_Type	sales_type

Transformation:

Transformation Name	Rename columns
Parameter: Option	Convert to lowercase
Parameter: Column	Daily,POS_Cost,Sales_Type

For example, if the old column name is `Sales_Type`, then the new column name is renamed to `sales_type`.

Convert to UPPERCASE

For the selected columns, you can convert the columns names to uppercase. Example:

Old Column Names	New Column Names
Daily	DAILY
POS_Cost	POS_COST
Sales_Type	SALES_TYPE

Transformation:

Transformation Name	Rename columns
Parameter: Option	Convert to UPPERCASE
Parameter: Column	Daily,POS_Cost,Sales_Type

For example, if the old column name is `Sales_Type`, then the new column name is renamed to `SALES_TYPE`.

Keep from beginning (left)

For the selected columns, you can specify the number of characters to keep from the beginning (left) of the column names. Based on the number of characters you provide, the column name is updated. Example:

Old Column Names	Number of characters	New Column Names
Daily	3	Dai
POS_Cost	3	POS
Sales_Type	3	Sal

Transformation:

Transformation Name	Rename columns
Parameter: Option	Keep from beginning (left)
Parameter: Column	Daily,POS_Cost,Sales_Type
Parameter: Number of characters	3

For example, if the old column name is `Sales_Type`, then based on the number of characters to keep from the beginning (left) is 3, then new column name is renamed to `Sal`.

Keep from end (right)

For the selected columns, you can specify the number of characters to keep from end (right) of the column names. Based on the number of characters you provide, the column name is updated. Example:

Old Column Names	Number of characters	New Column Names
Daily	4	aily
POS_Cost	4	Cost

Sales_Type	4	Type
------------	---	------

Transformation:

Transformation Name	Rename columns
Parameter: Option	Keep from beginning (right)
Parameter: Column	Daily, POS_Cost, Sales_Type
Parameter: Number of characters	4

For example, if the old column name is `Sales_Type`, then based on the number of characters to keep from the end (right) is 4, then new column name is renamed to `Type`.

NOTE: If the number of characters are more than the length of the column names, then the whole name of the column is retained.

Find and replace

You can apply literals, Patterns, or regular expressions to match patterns of text in the source column names. These matching values can then be replaced by a fixed value.

Tip: The default behavior is to replace the first instance. Use the Match all occurrences checkbox to apply the pattern matching across all columns in your set.

For the selected columns, you can specify the number of characters to keep from end (right) of the column names. Based on the number of characters you provide, the column name is updated. Example:

Old Column Names	New Column Names
column1	Field1
column2	Field2
column3	Field3

Transformation:

Transformation Name	Rename columns
Parameter: Option	Find and replace
Parameter: Column	column1, column2, column3
Parameter: Find	'column'
Parameter: Replace with	'Field'

The above uses literal values for find and replace. For more information on pattern-based matching, see *Text Matching*.

Use row(s) as column names

When this method is applied, all of the values in the specified row or rows are used as the new names for each column.

NOTE: This method applies to all columns in the dataset.

Types:

Type	Description
Use a single row to rename columns	Specify the row number in the sample to use as the source for column names. NOTE: Source row number information must be available. See below.
Use the first row in the sample to rename columns	Use the first row in the sample as the name for all columns.
Combine multiple rows to rename columns	Specify two or more rows to combine into column names. Details are below. NOTE: Source row number information must be available. See below.

Source row number information:

NOTE: If source row number information is no longer available, this method cannot be used for column rename.

- If a value is not applied for the source row number, the next row of data is used.
- Source row numbers apply. Current row numbers may not be the same. In the data grid, mouse over the leftmost column to see available row information.
- Each value in the row or combination of values across rows must be unique within the set of new column names.
- The row is removed from its original position.
- If the product is unable to find unique multi-row headers for the column, the first row of the header set is used.

Combine multiple rows

The following transformation renames the columns in the dataset based on the values in rows 3 and 4 of the data:

Transformation Name	Rename columns
Parameter: Option	Use row(s) as column names
Parameter: Type	Combine multiple rows to name columns
Parameter: Row Numbers - row A	3
Parameter: Row Numbers - row B	4
Parameter: Choose your separator	'_'
Parameter: Fill across?	Selected

In the above:

- The separator is defined as an underscore character (_). This value can be empty.
- When Fill across is selected, if any row value is empty, the last non-empty value for the row in a previous column is used as part of the column header.

Sanitize Column Names

If needed, you can clean the names of the columns in your dataset.

When column names are sanitized:

- alphanumeric characters and underscores (`_`) are permitted
- spacebars are converted to underscores
- all other characters are removed

Although Trifacta® supports a wider range of characters, you may wish to sanitize your column names to simplify publishing to and import into downstream systems.

Sanitize during Import

The above sanitization can be applied to your column names when the dataset is imported.

Tip: If you notice issues with references to your column names in your recipes, you may be able to fix them by re-importing the dataset and choosing to sanitize during import.

Steps:

1. From the menubar, click **Library**.
2. In the Library page, click **Import Data**.
3. In the Import Data page, select the file or table to import.
4. Click **Edit Settings**.
5. In the dialog, select **Remove special characters from column names**.
6. Complete the import of the dataset.

For more information, see *File Import Settings*.

Sanitize via Transformation

Through the Transform Builder, you can add a step to sanitize column names in your recipe.

Transformation Name	Rename by removing special characters
Parameter: Option	Clean current column names

Tip: If you are sanitizing your column names for downstream systems, you should add this step at the end of your recipe.

You can perform more fine-grained column renaming operations. See *Rename Columns*.

Change Column Data Type

Contents:

- *Change Type*
 - *Change from column menus*
 - *Change through Transform Builder*
 - *Lock Data Type*
 - *Via Transform Builder*
 - *Unlock Data Type*
 - *Via Transform Builder*
 - *Via column menus*
 - *Change Datetime Data Type*
 - *Via column menus*
 - *Via Transform Builder*
-

While transforming your data, you may need to change the data type of one or more columns.

For example, data of String type may be the easiest to manipulate. Since there are no mismatched values for String data type, you may wish to change a column's data type to this baseline type.

- Data types that you see in the Transformer page represent types that are understood by the product.
- When data is imported from a separate datastore, Trifacta may apply internal data types to the data. These types may differ from the original data typing in the source. As needed, the inferring of data types can be disabled at the file, connection, or global level. For more information, see *Disable Type Inference*.
- When data is published from the product to a separate datastore, these types may be mapped to different data types in the target. For more information, see *Type Conversions*.

Tip: You can use the Change Column Type transformation to override the data type inferred for a column. However, if a new transformation step is added, the column data type is re-inferred, which may override your specific typing. You should consider applying Change Column Type transformations as late as possible in your recipes.

For more information on the available data types, see *Supported Data Types*.

Change Type

You can change a column's data type in one of the following ways:

Change from column menus

You can change the data type for individual columns through the following column menus:

1. To the left of the column name, you can click the icon and select a new data type from the list.

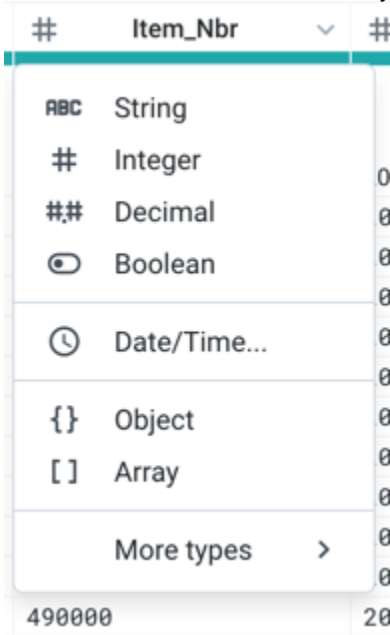


Figure: Column Data Type Menu

2. To the right of the column name, you can click the caret to open the column menu. Select **Change Type** and make a selection from the sub-menu.

Tip: Both of the above methods become individual steps in your recipe.

Change through Transform Builder

You can change data type for a single column or multiple columns through the Transform Builder. You can use a transformation like the following, which changes the columns `LastName`, `FirstName`, and `Address` to `String` data type.

Transformation Name	Change column type
Parameter: Column 1	LastName
Parameter: Column 2	FirstName
Parameter: Column 3	Address
Parameter: New Type	String

NOTE: You can lock the data type for columns to prevent it from being updated when the data is transformed in subsequent steps.

NOTE: When specifying a data type by name, you must use the internal value for the data type. The value in the column menu is the display name for the type.

For more information, see *Valid Data Type Strings*.

Lock Data Type

You can lock a column's data type through the Transform Builder. When a column's data type is locked, the data type is no longer automatically checked and updated by the Trifacta application.

Tip: If you do not wish to have the data types modified, you can add a transformation to lock all of them in a single step. Details are below.

Via Transform Builder

1. In the Search panel, enter `lock column type`.
2. From the **Columns** drop-down, select any one of the following options:
 - a. **Multiple:** Select one or more columns from the drop-down list.
 - b. **Range:** Specify a start column and ending column. All columns inclusive are selected.
 - c. **All:** Select all columns in the dataset.

NOTE: This option locks all the column's data type.

- d. **Advanced:** Specify the columns using a comma-separated list. You can combine multiple and range options under Advanced. Example:

```
c1,c3,c5~c8
```

3. Specify the other parameters.
4. To add the step to your recipe, click **Add**.

Example - lock a column's data type

This transformation locks the column data type:

Transformation Name	lock column to current type
Parameter: Columns	Multiple
Parameter: Column 1	Store_Nbr, Whse_Name

Example - lock the data types for all columns

This transformation locks the data types for all columns:

Tip: Many transformations support the Advanced option for column selection. You can specify column ranges, including all columns using the asterisk (*) wildcard. See the following.

Transformation Name	lock column to current type
Parameter: Columns	Advanced
Parameter: Column 1	*

Unlock Data Type

You can unlock a column's data type by following any one of these methods:

Via Transform Builder

In the Transformer Builder, you can select unlock to the current type option to apply the unlock feature to one or more columns.

This transformation unlocks the column data type:

Transformation Name	unlock column to current type
Parameter: Columns	Multiple
Parameter: Column 1	Store_Nbr, Whse_Name

Via column menus

You can unlock the data type for individual columns through the following column menus:

- To the left of the column name, you can click the icon and select **Automatically update**. The selected column is unlocked.

Change Datetime Data Type

If you are changing a column's data type to Datetime, you must also select a format string to apply to the column.

Via column menus

You can apply a Datetime data type through the column menus. When you choose the Datetime data type, you must apply a format for your Datetime values. For more information, see *Choose Datetime Format Dialog*.

Via Transform Builder

In the Transformer Builder, you can apply a specific transformation to format one or more columns to Datetime data type, using a specific format.

Tip: You can use the following transformation to change the format of a Datetime column.

This transformation looks like the following:

Transformation Name	Change column type
Parameter: Columns	Multiple
Parameter: Column 1	myDate
Parameter: New Type	Date/Time
Parameter: Date/time Type	month*dd*yyyy*hh:MMAx

Copy and Paste Columns

You can cut, copy, and paste columns or column values in your dataset through the Column Browser panel or the column menus in the data grid.

NOTE: You cannot copy and paste columns between datasets.

Steps:

1. In the Column Browser or the data grid, select the column or columns for your source.
2. After you have selected one or more columns, from the column menu, select one of the following options:

Menu option	Description
Cut	Cut the column(s) to the clipboard. Selection is removed from the dataset temporarily. <div>NOTE: Cut operations do not add steps to your dataset. If you choose to do something other than pasting the column or its values, the source column is left untouched.</div>
Copy	Copy the column(s) to the clipboard.
Paste before	Paste the column(s) in the clipboard before the currently selected column in the dataset.
Paste after	Paste the column(s) in the clipboard after the currently selected column in the dataset.
Paste values only	Paste the values from the column(s) in the clipboard into the selected column(s). <div>NOTE: When values are pasted into the column, the column data type may be re-inferred.</div>

3. Select the column where you wish to move the columns or paste the values.

NOTE: Do not select multiple columns for multi-column pasting. You must select only one column. Multi-column operations are applied to the columns to the bottom/right of the selection.

4. From the column menu, select **Paste**:
 - a. **Paste before:** Paste cut or copied columns before the selected one.
 - b. **Paste after:** Paste column(s) after the selected one.
 - c. **Paste values:** Replace values in the selected column(s) with the values from the column(s) in the clipboard. The number of selected columns on the clipboard and in the selected target area must match. Data types do not have to match.

NOTE: When values are pasted into the column, the column data type may be re-inferred.

Create Column by Example

You can create a new column of data from an existing one by providing example values for the new column for values in the source column. With each successive example value, Transformation by Example (TBE) improves the quality of the output values, until you have the desired set of values for your newly generated column.

Limitations:

- Transformation by Example works best for text-based inputs. Non-text inputs are treated as String type by the feature.

NOTE: Multi-value inputs, such as Object or Array data types, must be converted to String data type prior to transformation by example.

- In the Transformer page, TBE is applied across the currently displayed sample. In the entire dataset, there may be outlier values that do not match any of the examples that you have provided.

Tip: If your column data is quite varied, you should collect additional samples to verify that your TBE is properly matching all values in the column.

For more information, see *Overview of TBE*.

Steps:

- In the Transformer page, locate the column to use as your source. From the column menu, select **Create column from examples**.
- In the Transform Builder, enter the new column name.
- In the following example, a new column called `zip` is being created from the `Addresses` column:

The screenshot shows the 'Create column from examples' dialog in the Transform Builder. The 'Source' column is 'Addresses' and the 'New column name' is 'zip'. The 'Preview' column shows the first value '92704-4321' entered into the first empty cell. The dialog also includes a 'Cancel' button and an 'Add to Recipe' button.

Source	Preview
Addresses	zip
1 1881 South Poplar, Santa Ana 92704-4321	92704-4321
2 7772 Chapman, Garden Grove, CA 92841	92841
3 1143 S Nakoma Dr, Santa Ana, CA 92784	92784
4 1398 E. McFadden Avenue, Santa Ana, CA 92785	92785
5 9821 Catherine Ave. Garden Grove, CA 92841	92841
6 9892 Woodbury Rd. Garden Grove, CA 92843	92843
7 698 S Jackson St, Santa Ana, CA 92784	92784
8 13222 Lewis Street, Garden Grove, CA 92843	92843
9 15328 Pickford St. Westminster, CA 92683	92683
10 11383 Sandstone Ave, Fountain Valley, CA 92788	92788
11 15791 Bushard Westminster, CA 92683	92683
12 880 Mustang Way, Calimesa, CA 92320	92320
13 32870 Avenue E, Yucaipa, CA 92399	92399
14 13908 Foster Ave, Baldwin Park, CA 91786	91786
15 811 E Bishop St, Santa Ana, CA 92781	92781
16 12820 Bess Street, Baldwin Park, CA 91786	91786
17 1345 W. 48th Street, San Bernardino, CA 92487	92487
18 5378 North H St., San Bernardino, CA 92487	92487
19 7351 Holder Street Buena, Park, CA 98628	98628
20 12521 Monroe, Garden Grove, CA 92841	92841
21 1488 West 11th Street, 92411-2132	92411-2132
22 13523 2nd Street, Yucaipa, CA 92399	92399
23 489 48th St, San Bernardino, CA 92411	92411

Figure: Selected column and first value is specified

- Double-click an empty cell in the Preview column to populate it with an example. In the above, the zip code from the first value has been entered into the Preview column: 92704-4321.

Tip: You can copy values from the source column and paste them into the Preview column.

- While many of the zip code values from other rows have been accurately populated, there are still some values that need fixing. In the following, you can see that one zip code was not properly extracted. Double-click in the Preview column for the third row and fix the value: 91935:

The screenshot shows the DataTBE interface. On the left, a table with columns 'Source', 'Address', '#', and 'Zip' is displayed. The third row has a zip code of 91935. On the right, a 'Create column from examples' panel is open, showing a grid with 'Address' and 'Zip' columns. The 'Zip' column has a value of 91935. The 'Add to Recipe' button is highlighted.

Figure: Populating multiple example rows improves the overall quality of transformation across all rows

- A quick scroll through the rest of the rows in the sample indicate that you have properly extracted the zip code values for all rows.
- Click **Add to Recipe**.
- The new zip column is added to the dataset.

The screenshot shows the DataTBE interface with a dataset containing 48 rows and 6 columns. The columns are RBC, column2, column3, column4, Address, #, Zip, and column6. The 'Zip' column is highlighted. The dataset is displayed in a table view with a toolbar at the top and a sidebar on the left.

Figure: Transformed example column

Remove Data

Contents:

- *Considerations when removing data*
- *Delete columns*
- *Delete rows*
 - *Delete rows based on selections*
 - *Filter rows based on matching conditions*
 - *Filter rows based on data type mismatches*
 - *Delete rows based on multiple blank cells*
- *Remove values*
 - *Using regular expressions*

Through simple selections, you can identify columns to remove, values on which to base row deletion, or strings to remove from your dataset. As needed, these transformations can be modified for more sophisticated removal transformations.

Considerations when removing data

Please keep in mind:

- When data is removed from your dataset, no actual deletion is performed.
 - Trifacta® does not modify source data. All recipe executions generate new sets of data based on the transformations you define, which are applied to a generated version of the source data.
 - Transformation steps are previewed and can be undone on sampled data in the Transformer page, so you should feel free to experiment with data removal.
- In large volume datasets, be careful applying patterns or regular expressions to your data. You should limit your application of these pattern-based changes to the minimum range of columns, rows, or strings required to complete the task.

Delete columns

To delete a column from your dataset, click the column drop-down and select **Delete**. The data is no longer available in the data grid or subsequent recipe steps.

Tip: To delete multiple columns, select them in the data grid or column browser. Then select **Delete** from the column menu.

Tip: To simply remove columns from display, use the **Hide** command. The hidden column still appears in the output.

Manual transformations:

To delete multiple columns, you can specify comma-separated column names in your Delete Columns transformation:

Transformation Name	Delete columns
Parameter: Columns	ColA,ColC,ColE
Parameter: Action	Delete selected columns

To delete a range of columns, use the tilde (~) character between the start and end column names:

Transformation Name	Delete columns
Parameter: Columns	ColA~ColE
Parameter: Action	Delete selected columns

Delete rows

You can delete rows in your dataset based on conditional patterns that you specify. The easiest method is to select a string in the appropriate column and then choose the Delete suggestion card.

Delete rows based on selections

Steps:

In the following example, each row contains an entry for a different business, and you want to remove all of the business entries from the city of Tempe.

1. In this case, you could use the column histogram to select the value `Tempe` in the `city` column, or you can use the Filters panel to filter for rows containing the value `Tempe`.
2. Then, select the Delete suggestion card.

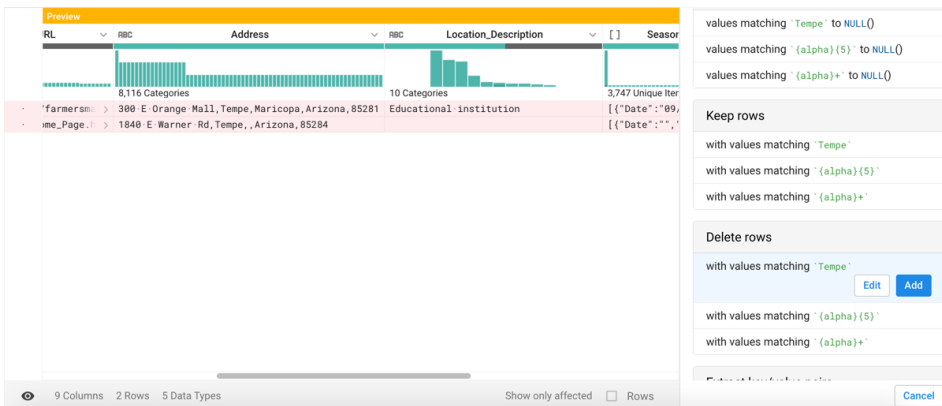


Figure: Select `Tempe` in the `City` column to remove all entries for that city

3. After selecting `Delete`, the application evaluates your selected value and attempts your intention with the selection. Is it a string literal or a pattern? If it's a pattern, what does the pattern represent? You may select one of the variants in the Delete card to find the right match.

NOTE: Be sure to scroll up and down in the data grid to review the values that are affected. In some cases, your selection may turn into a pattern, which could apply to more than just the desired values. In the previous example, selecting `Tempe` may yield a matching pattern of `{alpha}{5}`, which would match any five-letter city name, including `Tempe`. Select other variants in the Delete card to change the matching pattern. Click **Edit** to review the matching string.

4. After defining and modifying your Filter Rows transformation, you can use the preview to see the rows that will be removed, prior to adding the transformation to your recipe.

Tip: You can also use the Filter Rows to retain rows based on a specified condition, effectively deleting the rows that do not match. See *Filter Data*.

Filter rows based on matching conditions

You can delete or keep rows in your dataset based on one or more matching conditions you define.

1. In the Search panel, enter `filter`.
2. Select the type of conditional. You can filter based on:
 - a. Type: missing or mismatched values.
 - b. Matches: literal or pattern matches that are exact matches, partial matches, or matches with the beginning or ending of column values.
 - c. Ranges: Less than (or equal to), greater than (or equal to), or combinations.
 - d. Custom formula: Specify an expression that evaluates to `true` or `false`. If `true`, then the data is filtered.
3. Specify the other parameters, including whether to delete or keep the matching rows.

For more information, see *Filter Data*.

Filter rows based on data type mismatches

You can delete or keep rows based on whether a cell value in the row matches a specified data type. The following example removes rows that do not match the `mm*dd*yy` format for the Datetime data type from the `transactionDate` column.

1. In the Search panel, enter `filter mismatched`.
2. Specify the following transformation:

Transformation Name	Filter mismatched
Parameter: Condition	Is mismatched
Parameter: Column	transactionDate
Parameter: Date/Time type	mm*dd*yy
Parameter: Action	Delete matching rows

3. Review the preview. If it looks good, add it to your recipe.

Delete rows based on multiple blank cells

If you have rows in your dataset that contain no data, you can use the following two steps to remove them. Assuming that you know the starting (`col1`) and ending (`colN`) column names of your dataset, try the following:

NOTE: If at a later time, you reorder or remove the starting or ending columns in a step before this one, these steps are broken.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>MERGE([column1~columnN])</code>
Parameter: New column name	<code>'all_blank_vals'</code>

Transformation Name	Delete rows when value is missing
Parameter: Column	<code>all_blank_vals</code>
Parameter: Action	Delete selected columns

The above merges all values into a single value in the `all_blank_vals` column. The second step removes the row if the value in the merged column is blank.

Remember to delete the `all_blank_vals` column after you are done.

For more information, see *Filter Data*.

Remove values

To delete values from a column, select the values in the data grid. In the suggestion cards, select the `Replace` card. In the following example, the `city` column is removed of all values matching `Tempe`:

Transformation Name	Replace text or patterns
Parameter: Column	city
Parameter: Find	'Tempe'
Parameter: Replace with	' '
Parameter: Match all occurrences	true

The `Replace` transformation applies only to string values. The rest of a matching row is unaffected.

The above transformation matches all values in the column, even partial values, the match string is removed from the column value. For example, an entry `Tempest` would be turned into `st` if the above transformation was added.

To ensure that only full-column value matches are applied, you can add `Patterns` to indicate the start and end of the column value as in the following:

Transformation Name	Replace text or patterns
Parameter: Column	city
Parameter: Find	`{start}Tempe{end}`
Parameter: Replace with	' '
Parameter: Match all occurrences	true

In the above case, only values of `Tempe` that are the entire column value are matched.

Using regular expressions

For more sophisticated matching, you can apply regular expressions to your `replace` command. In the following example, all integers from 0-99 are matched in the `qty` column. Because there is no replacement value, they are deleted.

Regular expressions are very powerful pattern matching tools. You should be careful in your use of them. See *Text Matching*.

Character	Definition
^	Beginning of string. Required to prevent matching on the last digit of any numeric value.
\$	End of string. Required to prevent a 2-digit match on three-digit numbers.

\d	A single digit
	Logical or. In this case, it is used to define separate regexes for 1- and 2-digit values.

Deduplicate Data

Contents:

- *Validate Duplicate Data*
- *Remove duplicate rows transformation*
- *Deduplicate Rows Based on a Primary Key*
- *Deduplicate Columns*

As part of your data cleansing steps, you might need to remove duplicate rows of data from your dataset.

Validate Duplicate Data

In some cases, it might be acceptable to have duplicated data. For example, additional records using the same primary key might be included in a dataset as amendments or detail records.

NOTE: Before you remove duplicates from your dataset, you should verify that the data should not contain duplicates at all. If the data structure supports some duplicate elements including key values, you should exercise care in how you identify what constitutes duplicate information.

Remove duplicate rows transformation

Trifacta® provides a single transformation, which can remove identical rows from your dataset:

Tip: If you are attempting to identify if there are duplicate rows, check the row count in your dataset before and after you have added this transformation.

Transformation Name	Remove duplicate rows
---------------------	-----------------------

Limitations:

- This transformation is case-sensitive. So, if a column has values `Hello` and `HELLO`, the rows containing those values are not considered duplicates and cannot be removed with this transformation.
- Whitespace and the beginning and ending of values is not ignored.

Before applying the `Remove deduplicate rows` transformation, you should attempt to normalize your data. You can use the following techniques to normalize a few columns of data.

NOTE: If you have more than 20 columns of data, you might be better served by trying to identify a primary key method for de-duplicating your dataset. Details are below.

For individual columns, you can use the `trim` function to remove leading and trailing whitespace:

NOTE: To preserve the original column values, use the `New formula` transformation. The `Edit column with formula` transformation replaces the original values.

Transformation Name	New formula
Parameter: Formula type	Single row formula

Parameter: Formula	TRIM(Item)
---------------------------	------------

Since the `Remove deduplicate rows` transformation is case-sensitive, you can use the `LOWER` function to make the case of each entry in a column to be consistent:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	LOWER(Description)

Deduplicate Rows Based on a Primary Key

Another method to deduplicate data might be to delete rows based on one or more columns that you identify as a primary key for the dataset. A **primary key** is an identifier that uniquely identifies a row of data within a dataset. It can be a single field (column) or a combination of columns. For example, in a datasets of restaurant locations, the primary key can be a combination of RestaurantName, Address, and Zip.

NOTE: Before continuing, you must identify a primary key for your dataset. See *Generate Primary Keys*.

When you have identified your primary key, you should identify the appropriate method for your dataset. Please complete the following steps.

Steps:

1. If your primary key spans multiple columns, use the `Merge columns` transformation to bring the values into a single column:

Transformation Name	Merge columns
Parameter: Columns	RestaurantName,Address,Zip
Parameter: Separator	' - '

2. Rename the generated column: `PrimaryKey`.
3. Use the following transformation to generate a new column, comparing each value in the `PrimaryKey` column to the previous one:

Transformation Name	Window
Parameter: Formulas	PREV(PrimaryKey, 1)
Parameter: Order by	PrimaryKey

4. For each row, the value of the new column is the value in the `PrimaryKey` for the previous row. Now, test if this value is the same as the value in the `PrimaryKey` column for the current row:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	IF((window==PrimaryKey),true,false)
Parameter: New column name	IsDupe

-
5. The new column (`IsDupe`) contains `true` for duplicate primary keys. Delete the rows that are duplicates:

Transformation Name	Delete rows
----------------------------	-------------

6. Delete any generated columns that are no longer needed.

Deduplicate Columns

While this form of duplicate data is rarer, you might want to check on the possibility of duplicate data between your columns. To check for duplicate column data, you can use a transformation similar to the following:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	Column1 == Column2
Parameter: New column name	'dupeColVals'

In the generated column, values that are `true` indicate duplicate data. If all values are `true`, then you can remove one of the columns.

Compare Values

Contents:

- *Compare Numeric Values*
 - *Compare Boolean Values*
 - *Compare Date Values*
 - *Compare String Values*
-

Depending on the data type, you can compare values in separate columns or single columns against fixed values.

Compare Numeric Values

You can use basic comparison operators to perform comparisons on your data. In this example, the `compareCol` column is generated as the evaluation of `3 < 6`, which is `true`:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	(3 < 6)
Parameter: New column name	'compareCol '

For more information, see *Comparison Operators*.

Compare Boolean Values

Boolean values can be `true` or `false`, so comparisons like the following can be applied to a Boolean set of values:

Transformation Name	Edit column with formula
Parameter: Columns	Attendance
Parameter: Formula	IF(isSeated == true,true,Attendance)

In the above case, the value in `Attendance` is set to `true` if the value in the `isSeated` column is `true`. Otherwise, the current value in `Attendance` is used.

Compare Date Values

You can use the `DATEDIF` function to compare two date values, as in the following, which compares the number of days between `startCol` and `endCol` values:

NOTE: Both parameters of the `DATEDIF` function must be column references containing valid date values.

Transformation Name	New formula
Parameter: Formula type	Single row formula

Parameter: Formula	DATEDIF(startCol, endCol, 'day')
Parameter: New column name	'DurationInDays'

Compare String Values

See *Compare Strings*.

Replace Cell Values

You can search and replace for specific values in a column.

If the column is known

Steps:

1. Select the column containing the value you wish to replace.
2. In the Selection Details panel on the right side, click the appropriate bar under Unique Values. All matching values are selected within the column.
3. Right-click the bar and select **Replace Values...**
4. A pre-configured transformation appears in the Transform Builder. Example:

Transformation Name	Replace cells
Parameter: Column	myDates
Parameter: Find	' 2013/02/07 '
Parameter: Replace with	' '

Tip: You can search for multiple items within the same column. Add other search values in additional Find textboxes.

5. Add your value in the Replace with textbox.
6. Click **Add**.
7. All matching cell values in the column are replaced with your entered value.

For more information, see *Selection Details Panel*.

If the column is unknown

Steps:

1. If you do not know the column, click the Filter tool in the Transformer bar.
2. Click the Rows tab and enter the value to locate. Only rows where the value appears are displayed in the data grid. Each instance of the matching value is highlighted.
3. Locate the column containing the specific highlighted values to replace. Select the value.
4. In the Suggestions panel, locate the Replace transformation card.
5. Select the variant of the Replace transformation that contains the specific value you selected. Then, click **Edit**.
6. A pre-configured transformation appears in the Transform Builder. Example:

Transformation Name	Replace text or patterns
Parameter: Column	myString
Parameter: Find	'Red'
Parameter: Replace	' '

7. Enter your replacement value in the Replace with textbox.

8. Click **Add**.
9. All matching values in the column are replaced with your entered value.

For more information, see *Filter Panel*.

Replace Values Using Patterns

Contents:

- *Replace Methods*
 - *Replace by selection*
 - *Replace using Transformer toolbar*
 - *Replace using Column Details panel*
- *Find Values in a Column*
- *Examples*
 - *Replace first three characters*
 - *Replace using literal expressions*
 - *Replace string of four digits*
 - *Replace date and time patterns*
 - *Replace based on position*
 - *Replace alpha-numeric and position patterns*
 - *Replace using special patterns*

Trifacta® patterns enable you to identify patterns in cell values and to perform replacements on those found elements of text. This section describes how to use patterns to find text and replace them with preferred values.

Tip: Patterns can also be used to extract values from cell values into a new column. The Trifacta patterns listed on this page can also be applied to the Extract text or pattern transformation. For additional example Trifacta patterns, see *Extract Values*.

- For more information, see *Overview of Pattern Matching*.
- For more information on Pattern syntax, see *Text Matching*.

Replace Methods

You can use the Replace text or patterns transformation to replace values in one or more columns with literal values, Trifacta patterns, or regular expressions through any of the following methods. You can use this transformation to replace missing, mismatched, or bad data using the following methods.

Replace by selection

When you select a piece of text in the data grid, the replace suggestion card displayed in the Selection Details panel on the right side may contain Pattern-based options for finding the selected value and similar values in the column of data. You can use these suggestions to replace column values.

Steps:

1. Select the data you want to replace. The suggestion cards are displayed.
2. In the Selection Details panel on the right side, select the Replace pattern suggestion card and click **Edit**.
3. The Replace text or patterns transformation is specified for you in the Transform Builder, where you can modify the Find value and other parameters as needed. See example below.

Replace using Transformer toolbar

In the Transformer toolbar at the top of the grid, click **Replace > Text or Pattern**. The Replace text or pattern transformation is displayed in the Transform Builder. For more information, see *Transformer Toolbar*.

For more information on procedures, see "Replace using Transform Builder" below.

Replace using Column Details panel

You can review sets of patterns for the selected column in the Column Details panel. When you select a column in the Column Details panel, you are prompted with a set of suggested patterns.

For more information on suggestions, see *Overview of Predictive Transformation*.

Replace using Transform Builder

When a pattern suggestion is selected, it is specified in the Transform Builder for review and addition to your recipe. In the Transform Builder, you can select one or more columns to replace text or patterns.

Steps:

The following steps describe how to build a pattern-based replacement transformation from scratch in the Transform Builder.

Tip: Some selections in the data grid or related tools can lead to suggestions or pre-configured transformations in the Transform Builder.

1. Enter `Replace text or pattern` in the Search panel.
 2. Select an individual column or multiple columns from the following options:
 - **Multiple:** Select one or more columns from the drop-down list.
 - **All:** Select all columns in the dataset. See below for an example.
 - **Range:** Specify a start column and an ending column. All columns in between are selected.
 - **Advanced:** Specify the columns using a comma-separated list. You can combine multiple and range options under Advanced.
 - Ranges of columns can be specified using the tilde (~) character.
 - The following example range selects from the dataset as displayed in the data grid `column1`, `column3`, and the range of columns between `column5` and `column8`, inclusive:
- `column1,column3,column5~column8`
3. In the Find text box, enter the text value or pattern that matches the value you want to replace. For more information, see "Find Values in a Column" below.
 4. In the Replace text box, enter the value to replace the found text.
 5. For additional controls, click **Advanced Options**:
 - a. **Start search after:** Enter a text or pattern that precedes the value you want to replace. See below example.
 - b. **Start search before:** Enter a text or pattern that follows the value you want to replace. See below example.
 - c. **Ignore case:** If selected, case is ignored when matching.
 - d. **Match all occurrences:** If selected, all occurrences of the found text in the column are matched and replaced.
 6. Click **Add**. The transformation is added to your recipe, and the selected columns are replaced with appropriate patterns in the data grid.

Find Values in a Column

The `Replace with text or pattern` transformation enables you to replace values within the specified column or columns based on a string literal or Trifacta patterns. When you specify the transformation in the Transform Builder, the Find textbox can be populated with one of the following types of values:

Find type	Description	Delimiter	Example
Literal	A literal pieces of text	single	<input type="text"/>

		quotes	<code>&apos;My piece of text&apos;</code>
Trifacta pattern	<p>A Trifacta pattern represents zero or more characters that match a pattern. In Trifacta, Patterns are a simplified means of expressing regular expressions. For more information on Trifacta pattern syntax, see <i>Text Matching</i>.</p> <p>Tip: The examples in this section use Trifacta Patterns, which are simpler to use than regular expressions.</p>	back-ticks	<code>`{start}{digit}{3}`</code>
Regular expression	<p>Regular expressions are a standard-based method of describing patterns in values.</p> <p>NOTE: Regular expressions are considered a developer-level skill. For more information on regular expression, see on <i>RE2</i> and <i>PCRE</i> regular expressions.</p>	forward slashes	<code>/^.{0,3}/</code>

Examples

The following examples demonstrate how Trifacta Patterns can be used to find and replace values within a column or set of columns.

Replace first three characters

This example uses Trifacta Pattern to find the first three characters. In this example, the first three characters of the `Customer ID` column are replaced with the value `CustID-` for the selected column in the dataset.

Transformation:

Transformation Name	Replace text or patterns
Parameter: Column	CustomerID
Parameter: Find	<code>`{start}%{3}`</code>
Parameter: Replace with	<code>CustID-</code>

Results:

Before	After
Tri02468	CustID-02468
Mul2239	CustID-2239
Zev5521	CustID-5521

Replace using literal expressions

This example is based on the search and replace content in your dataset using literals. In the following example, the value `##CLT_NAME##` is replaced with `Our Customer, Inc.` across all columns in the dataset.

Transformation:

Transformation Name	Replace text or patterns
Parameter: Column	All

Parameter: Find	'##CLT_NAME##'
Parameter: Replace with	'Our Customer, Inc.'
Parameter: Match all occurrences	true

Replace string of four digits

Tip: For privacy reasons or sensitivity reasons, you can mask the sensitive data with the following replacements.

The following example uses Trifacta Patterns to find a string of four digits. The replacement is based on the structure of the data, not on the type of data. If you have data that are not credit card numbers yet follows the four-digit pattern, those values can also be replaced. In this example, the `myCreditCardNumbers` column is masked with `XXXX`.

Transformation:

Transformation Name	Replace text or patterns
Parameter: Columns	<code>myCreditCardNumbers</code>
Parameter: Find	<code>`{start}{digit}{4}{any}{digit}{4}{any}{digit}{4}{any}({digit}{4}){end}`</code>
Parameter: Replace with	<code>XXXX-XXXX-XXXX-\$1</code>

Results:

Before	After
1234-1234-1234-1234	XXXX-XXXX-XXXX-1234
1111-1111-1111-1111	XXXX-XXXX-XXXX-1111
4321-4321-4321-4321	XXXX-XXXX-XXXX-4321

Using capture groups

The previous example captures aspects of the found pattern for use during replacement. A **capture group** is a mechanism in Trifacta Patterns or regular expressions to capture one or more parts of the matched values into variables.

In the example, the last four-digit segment of the Trifacta Pattern is surrounded by parentheses:

```
((digit){4}){end}
```

This group of digits is captured as the first (and only) capture group. In the replacement string, it is referenced as:

```
$1
```

You can have multiple capture groups in a single pattern. In the replacement, these capture groups can be referenced sequentially left-to-right from the pattern: `$1`, `$2`, and so on.

For more information, see *Capture Group References*.

Tip: You can use both `{digit}` and `{#}` Trifacta patterns for columns containing numeric values.

Replace date and time patterns

The following example is based on replacing the date and time using the pre-configured suggestions displayed in the search context panel. In this example, the date Trifacta Patterns `yy/mm/dd` is replaced with `mm/dd/yy`.

Transformation:

Transformation Name	Replace text or patterns
Parameter: Column	ORDER_DATE
Parameter: Find	`({yy}){delim}({MM}){delim}({dd})`
Parameter: Replace with	\$2-\$1-\$3

Results:

Before	After
20/11/02	11/02/20
20/11/22	11/22/20
20/11/26	11/26/20

Replace based on position

You can specify replacements based on the character position of values in your source column values. This method of finding and replacing values is useful if the source column data is consistently structured.

For example, suppose you have dates in the following format:

Before
2020-05-01
2020-05-02
2020-05-03

Transformation:

Suppose you wanted to replace the value for the month with `Month`, you could add the following transformation step:

Transformation Name	Replace between positions
Parameter: Column	Before
Parameter: Start position	6
Parameter: End position	8
Parameter: Replace with	Month

Results:

After
2020-Month-01
2020-Month-02
2020-Month-03

To replace the four digits of the year, you could perform a basic replace text or pattern transformation with a pattern to find of the following:

```
`{start}{digit}{4}`
```

Replace alpha-numeric and position patterns

You can use alpha-numeric and position Trifacta patterns for replacing the customer's address in the the dataset. In this example, `{alpha-numeric}` pattern is applied to find the customer's addresses and used `{start}` and `{end}` pattern to mention the position of replacement. For more information on Pattern Syntax, see *Text Matching*.

Transformation:

Transformation Name	Replace text or patterns
Parameter: Column	address_street_number
Parameter: Find	`{alpha-numeric}`
Parameter: Replace with	##
Parameter: Start search after	`{start}{digit}{2}`
Parameter: Stop search before	`{any}`

Results:

Before	After
3298, Church Street	32##, Church Street
4132, Park Avenue	41##, Park Avenue
1234, McGrath Road	12##, McGrath Road

Replace using special patterns

You can use the following special Trifacta Pattern tokens to search for matches in your source values. In some cases, these Trifacta Patterns are consistent with the patterns used for specific data types.

Pattern	Description
<pre>`{at-username}`</pre>	Matches values that begin with an at-sign, such as @trifacta. This Trifacta Pattern can be useful if you need to remap or mask username values.
	Matches values that begin with a hashtag, such as #dataprep. For an example of this Trifacta Pattern, see <i>Extract Values</i> .

<pre>` {hashtag} `</pre>	
<pre>`{hex}`</pre>	Matches values that are valid hexadecimal (base-16) numbers. These values contain a string of numerals, letters A-F, and combinations of them, without spaces. Examples: <code>AE00</code> , <code>1F2F</code> , <code>100</code> .
<pre>`{phone} `</pre>	Matches valid phone numbers within a set of values. For more information on this data type pattern, see <i>Phone Number Data Type</i> .
<pre>`{email} `</pre>	Matches valid email addresses within a set of values. For more information on this data type pattern, see <i>Email Address Data Type</i> .
<pre>`{url}`</pre>	Matches valid URL addresses within a set of values. For more information, on this data type pattern, see <i>URL Data Type</i> .

Replace Groups of Values

Contents:

- *Replacement methods*
 - *Replace by selection*
 - *Mask data*
 - *Delete whole column(s)*
 - *Masking all values*
 - *Partial masking of values*
 - *Mask multiple columns based on data type*
 - *Replace with values from another column*
 - *Replace whole column*
 - *Replace partial values from another column*
 - *Replace between positions*
 - *Search and replace text or pattern*
 - *Replace missing values*
 - *Replace missing with zeroes*
 - *Replace missing with average values*
 - *Replace mismatched values*
-

Whether data is missing, mismatched, or simply wrong, you can use a variety of methods in the Trifacta® application to replace values in one or more columns with literal values or pattern-based replacements.

Replacement methods

In the Transformer page, you can use the following methods to replace values:

Method	Description
By selection	Select a value in the data grid to prompt a series of suggestions on what to do with the data. Typically, replacement options are near the top of the suggestions. <div>Tip: You can replace specific values in a column with a preferred value. For more information, see <i>Replace Cell Values</i>.</div>
By column menu	From the column menu, select Replace and a sub-menu item to begin configuring a replacement transformation.
By Transformer toolbar	At the top of the data grid, click the Replace icon in the Transformer toolbar to begin configuring replacements.
By Search panel	In the Search panel, enter <code>replace</code> to build a replacement transformation from scratch.

Replace by selection

When you select data in the data grid, the replacement suggestions are pre-specified for you, including a number of variants available in the suggestion card.

Notes:

- Suggestions are typically conservative in the scope of their changes. Case-sensitive searches and matching of the first occurrence only are the default settings.
- Order of listing of suggestions in a suggestion card:
 - Pattern-based replacements are listed first. These replacements use `Patterns`, instead of regular expressions. Regular expressions can be more difficult to control.

- Literal value replacements are listed below the pattern-based ones.

For more information, see *Overview of Predictive Transformation*.

Mask data

For privacy reasons or for sensitivity reasons, you may wish to mask sensitive data in one or more columns with fixed strings.

Delete whole column(s)

If you need to remove the data in an entire column, the easiest method is to delete a column. Select one or more columns and then select **Delete** from the column drop-down. See *Remove Data*.

Masking all values

You can use a transformation like the following to replace all values in a column with a simple string. In this case, the value #REDACTED# has been inserted in place of all values in the column.

NOTE: This replacement changes the data type of the column to String. If you must retain the original data type, the replacement value should be valid for the data type.

Transformation Name	Edit column with formula
Parameter: Columns	transactionValue
Parameter: Formula	'#REDACTED#'

Partial masking of values

Suppose you wish to partially mask data in a column. In the following example, data for the AcctNum column is masked, except for the last four characters (digits):

Transformation Name	Edit column with formula
Parameter: Columns	AcctNum
Parameter: Formula	value: merge(['XXXX',right(AcctNum, 4)], '')

Mask multiple columns based on data type

You can use the following type of transformation to hide data based on data type. In this example, the values in all columns with Social Security Number (SSN) are replaced with a masking value: XXX-XX-XXXX:

This method performs a simple text replacement of the data in the columns(s). After this transformation has been applied to the data, the source data is no longer available, unless you step back to a step before this one. For these kinds of operations, you may find it more secure to apply these kinds of masking operations to the source data in a single recipe and then make that output available to other users to use as an imported dataset.

Transformation Name	Edit column with formula
Parameter: Columns	All
Parameter: Formula	if(isvalid(\$col, ['SSN']), 'XXX-XX-XXXX', \$col)

Replace with values from another column

Replace whole column

You can do simple replacements of data from one column into another with transformations like the following. In this example, the values of `colB` are replaced with the values of `colA` with `0.15` added to them:

Transformation Name	Edit with formula
Parameter: Columns	colB
Parameter: Formula	colA + 0.15

Replace partial values from another column

You can use the `MERGE` function to blend full or partial sets of columns into a new column. In the following example, the `newBrandId` value is concatenated with the product code in the `ProdId` column to create a new product identifier:

Transformation Name	Edit with formula
Parameter: Columns	ProdId
Parameter: Formula	merge([newBrandId, right(prodId, 5)], '-')

Replace between positions

You can perform replacements based on character positions that you specify as part of the transformation.

- The beginning character value is specified as a number from 0, which starts on the left.
- The ending character value must be equal to or greater than the beginning character value.

In the following example, the `Whse_Name` column values are prepended with the value `old-`.

Transformation Name	Replace by position
Parameter: Column	Whse_name
Parameter: Start position	0
Parameter: End position	0
Parameter: Replace with	old-

Search and replace text or pattern

You can search and replace content in your dataset based on literals or patterns. In the following example, the value `##CLT_NAME##` is replaced with `Our Customer, Inc.` across all columns in the dataset:

Transformation Name	Replace text or patterns
Parameter: Column	All
Parameter: Find	'##CLT_NAME##'
Parameter: Replace with	'Our Customer, Inc.'

Parameter: Match all occurrences	true
---	------

Replace missing values

Replace missing with zeroes

For numeric data, you may choose to replace values that are missing in a column with zeros. The following transformation sets missing values in the `Qty` and `DiscountPct` columns of Decimal data type to 0:

Transformation Name	Edit column with formula
Parameter: Columns	Qty,DiscountPct
Parameter: Formula	if(ismissing([\$col]), '0', \$col)

Replace missing with average values

One of the problems with the above method is that any statistical computations applied to the column are now affected by the zeroing of the missing values. For example, the computation for the `AVERAGE` function does not factor in missing values into the count of rows, which result in skewing of values for your purposes.

The following example creates a new column from the `DiscountPct` column in which empty values are inserted as the average of the values in the source column:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	if(ismissing([DiscountPct]), average(DiscountPct), DiscountPct)
Parameter: New column name	DiscountPct-0toAVG

In this manner, the new column can be used for some statistical modeling, while preserving the original values in the original column.

Replace mismatched values

You can perform replacements based on the values in a column that are mismatched against a specified type.

In the following example, Datetime values that do not match the `yyyy*mm*dd`, where the asterisk (*) is a wildcard value.

Transformation Name	Replace mismatched values
Parameter: Columns	Multiple
Parameter: Column 1	myDate
Parameter: Data type to evaluate	Date/Time
Parameter: Date/Time type	yyyy*mm*dd
Parameter: Replace with	Custom value
Parameter: New value	'##BAD_DATE##'

NOTE: In the above example, the Date/Time type parameter applies only to replacements that are mismatched against the Date/Time data type. This parameter is used to specify the Datetime format against which the source values are validated. The parameter does not appear in Replace mismatched values transformations for other data types.

Normalize Numeric Values

Contents:

- *Numeric precision*
- *Standardize decimal precision*
- *Standardize units*
 - *Example - Fixed conversion factors*
 - *Dynamic conversion factors*
- *Adjust level of precision*
 - *Adjust data granularity by aggregation*

This section describes techniques to normalize numeric values in your datasets.

Ideally, your source systems are configured to capture and deliver data using a consistent set of units in a standardized structure and format. In practice, data from multiple systems can illuminate differences in the level of precision used in numeric data or differences in text entries that reference the same thing. Within Trifacta®, you can use the following techniques to address some of the issues you might encounter in the standardization of units and values for numeric types.

Numeric precision

In Trifacta, mathematical computations are performed using 64-bit floating point operations to 15 decimals of precision. However, due to rounding off, truncation, and other technical factors, small discrepancies in outputs can be expected. Example:

-636074.22

-2465086.34

Suppose you apply the following transformation:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	(-636074.22 + -2465086.34)
Parameter: New column name	MySum

The expected output in the MySum column: -3101160.56

The actual output for in the MySum column: -3101160.5599999996

NOTE: For 64-bit floating point mathematical operations, deviations like the above are intrinsic to the Decimal data type and how the platform performs computations.

Depending on your precision requirements, you can manage precision across your columns using a transformation like the following, which rounds off MySum to three digits:

Transformation Name	Edit column with formula
Parameter: Columns	MySum

Parameter: Formula	ROUND(\$col, 3)
---------------------------	-----------------

For more information on floating point computations, see https://en.wikipedia.org/wiki/Numeric_precision_in_Microsoft_Excel.

Standardize decimal precision

If decimal values in a column are of varying levels of precision, you can standardize to a single level of precision.

Steps:

1. From the column menu, select **Column Details**.
2. In the Column Details panel, select the Patterns tab. Among the patterns, select the following:

```
{digit}.{digit}
```

3. In the Suggestions panel on the right, locate the Edit Column transformation suggestion that uses the ROUND function. Click **Edit**.
4. Change the second parameter of the ROUND function to match the number of digits of precision.

You can generalize this formatting across multiple columns by applying the \$col reference in the transformation's function, as in the following:

Transformation Name	Edit column with formula
Parameter: Columns	colA, colB, colC
Parameter: Formula	IFVALID(\$col, ['Float'], ROUND(\$col, 2))

Standardize units

Tip: Each column that contains numeric values should have an identified unit of measurement. Ideally, this information is embedded in the name of the column data. If the unit of measurement is not included, it can be difficult to properly interpret the data.

Trifacta does not impose any units on imported data. For example, a column of values in floating point format could represent centimeters, ounces, or any other unit of measurement. As long as the data conforms to the specified data type for the column, then Trifacta can work with it.

However, this flexibility can present issues for users of the dataset. If data is not clearly labeled and converted to a standardized set of units, its users are forced to make assumptions about the data, which can lead to misuse of it.

Tip: The meaning of some units of measure can change over time. For example, a US Dollar in 2010 does not have the same value as a dollar in 2015. When you standardize shifting units of measure, you should account for any time-based differences, if possible.

Example - Fixed conversion factors

In many cases, units can be converted to other units by applying a fixed conversion factor to a column of data. For example, your dataset has the following three columns of measured data:

Person	Height_ft	Weight_kg	Arm_Length_in
Jack	5'10"	92 kg	32

Jill	5'2"	56 kg	29
Joe	6'3"	101 kg	35

The above data has the following issues:

1. The Weight and Height columns contain unit identifiers, which forces the values to be treated as strings.
2. Metric data (kg) is mixed with English unit data (ft and in).
3. The Height data is non-numeric.

Problem 1 - remove units

The `Weight_kg` column contains a unit identifier. On import, these values are treated as strings, which limits their use for analysis.

Steps:

1. In the data grid, select an instance of " kg". Note that the space should be selected, too.
2. Among the suggestion cards, select the Replace card.
3. It should automatically choose to replace with nothing, effectively deleting the content. To check, click **Modify**.
4. The transformation should look like the following:

Transformation Name	Replace text or patterns
Parameter: Column	Weight_kg
Parameter: Find	' kg'
Parameter: Replace with	' '
Parameter: Match all occurrences	true

5. Add it to your recipe.
6. Verify that the column's data type has been changed to `Integer` or `Decimal`, depending on the values in it.

Problem 2 - convert English to metric units

To normalize to English units, the first issue is easily corrected by multiplying the Weight values by 2.2, since 1 kg = 2.2 lb:

Transformation Name	Edit column with formula
Parameter: Columns	Weight_kg
Parameter: Formula	(Weight_kg * 2.2)

If you want to round the value to the nearest integer, use the following:

Transformation Name	Edit column with formula
Parameter: Columns	Weight_kg
Parameter: Formula	ROUND((Weight_kg * 2.2))

After the above is added to the recipe, you should rename the column: `Weight_lbs`.

Problem 3 - convert ft/in to in

The final issue involves converting the `Height_ft` values to a single value for inches, so that these values can be used consistently with the other columns in the dataset.

On import, your data for the column might actually look like the following:

Height_ft
"5'10"
"5'2"
"6'3"

Steps:

1. Select the first quote mark in one of the entries.
2. In the suggestion cards, select the Replace card.
3. Select the variant that deletes all quotes in the column.
4. The full command should look like the following:

Transformation Name	Replace text or patterns
Parameter: Column	Height_ft
Parameter: Find	` ``
Parameter: Replace with	' '
Parameter: Match all occurrences	true

5. Add it to your recipe.
6. The remaining steps compute the number of inches. Multiply the feet by 12, and then add the number of inches, using new columns of data.
7. Select the single quote mark, and choose the Split suggestion card. This transformation step should split the column into two columns: `Height_ft1` and `Height_ft2`.
8. Derive the value in inches:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	$((\text{Height_ft1} * 12) + \text{Height_ft2})$
Parameter: New column name	Height_in

9. You can delete the other, interim columns.

Dynamic conversion factors

In some cases, the conversion rate between two different units of measures is dynamic. A common example involves mismatches between currency. For example, one dataset can be using U.S. dollars while another represents values in Euros.

Within a column

If you have inconsistent units within a column, it might be possible to correct these values by applying a multiple. For example, you might be able to determine that some values are in kilometers, instead of meters, based on their much smaller values. Multiplying the kilometer values by 1000 should standardize your units. The following multiplies all values in the column `Distance` that are less than 1000 by 1000.

Transformation Name	Edit column with formula
Parameter: Columns	Distance
Parameter: Formula	IF((Distance < 1000,(Distance * 1000), Distance)

Note the implied assumption that there are no distances in kilometers that are over 1000.

NOTE: Inconsistency in units within a column indicates a problem in either the source data or how the column data was modified after import. Where possible, you should try to fix these issues in the source data first, as they can introduce problems when the data is used.

Adjust level of precision

For numeric values that are used for measurement, you can adjust the level of precision within and across columns of values. For example, you have the following columns of data:

Name	Width_cm	Height_cm
Object 1	23.3	55.5512
Object 2	65.2	102.4024
Object 3	54.2	12.22

In the above, you can see the following precision mismatches:

- The Height column contains one value with only two digits of arithmetic precision in measurement.
- The Width column uses two digits of arithmetic precision, while the Height column contains more digits of precision.

Where precision in measurement is important, you should consider rounding to the lowest level of precision. In this case, within the Height column, that level is to two significant digits after the decimal point (e.g. 12.22). However, across all of the columns of the dataset, the level of precision is to one significant digit after the decimal point, as the Width values are all restricted to this level of precision. While you could choose to round off to four digits across all columns, the extra values of 0 do not accurately reflect measurement and are therefore misleading.

You can use the following transformations to perform rounding functions within these columns:

Transformation Name	Edit column with formula
Parameter: Columns	Width_cm
Parameter: Formula	NUMFORMAT(Width_cm '#.#')

Transformation Name	Edit column with formula
Parameter: Columns	Height_cm
Parameter: Formula	NUMFORMAT(Height_cm '#.#')

NOTE: The above assumes that the number of significant digits remains fixed in the source data. If this varies over times or uses in your recipe, you might need to revisit these specific transformation steps.

NOTE: The above formatting option drops the zero for values like 4 . 0. As an alternative, you can use a format of ' # . 0 ', which always inserts a zero, even in cases where the zero is not present.

Results:

Name	Width_cm	Height_cm
Object 1	23.3	55.5
Object 2	65.2	102.4
Object 3	54.2	12.2

Adjust data granularity by aggregation

For data hierarchies, you can use aggregations to adjust the granularity of your data to the appropriate grouping level. For example, you want to join a dataset that is organized by individual products with a dataset that is organized by brand. In most cases, you should aggregate the product-level data in the first dataset to the brand level.

NOTE: When aggregation is applied, a new table of data is generated with the columns that you specifically select for inclusion.

For more information, see *Pivot Data*.

Standardize Using Patterns

Contents:

- *Example - Phone number patterns*
- *Generic Conversions*
- *Datetime Patterns*

This section describes techniques to standardize values in your datasets using patterns. From the Column Details panel in the Trifacta® application, you can review and select patterns in the column's data. These selections can be used as the basis for converting all applicable values to the selected format.

NOTE: Pattern-based conversions can be applied to any data type.

In the Patterns tab, click the whitespace around a pattern and then review the Convert suggestion to define how the pattern matches can be converted to a single standardized format.

Tip: To select, click the whitespace around the pattern and example values.

NOTE: The application does not suggest pattern-based conversions that add or remove alphanumeric characters.

The screenshot displays the Trifacta application interface. On the left, the 'Patterns' tab is active, showing a list of patterns for the 'contractDate' column. The patterns are categorized into three groups: 'All patterns' (12 total), 'dd / mm / yyyy' (4 total), and 'm / d / yyyy' (4 total). The 'dd / mm / yyyy' group includes examples like '14/03/2018', '15/04/2017', '15/04/2016', and '28/04/2018'. The 'm / d / yyyy' group includes examples like '3/14/2018', '4/6/2017', '12/2/2016', and '5/4/2015'. On the right, the 'Suggestions' panel is open, showing a 'Convert' task. It lists values like '3/14/2018' and '2014-3-14' and suggests converting them to the pattern format '14/03/2018'. There are 'Edit' and 'Add' buttons for this suggestion. Below the 'Convert' section, there is a 'Delete rows' section with a suggestion to delete rows with values matching the pattern '(start)((dd)/(mm)/(yyyy))(end)'. At the bottom, there is a 'Set' section.

Figure: Selecting Datetime patterns in the Patterns tab

In the above, the pattern block prompts suggestions for Convert tasks based on the selected patterns.

- Click **Edit** to modify the task.
- Click **Add** to add the task as a step to your recipe.

Example - Phone number patterns

For columns containing phone number data, you can use the Patterns tab to standardize formatting options. Consider the following values, which are valid phone numbers. Next to each value is a pattern representing the value:

PhoneNum	Pattern
(415) 555-1212	<code>\((\{digit\}{3})\) (\{digit\}{3})\-(\{digit\}{4})</code>
415-555-1212	<code>(\{digit\}{3})\-(\{digit\}{3})\-(\{digit\}{4})</code>
415.555.1212	<code>(\{digit\}{3})\.\{digit\}{3}\.\{digit\}{4}</code>
415 555-1212	<code>(\{digit\}{3}) (\{digit\}{3})\-(\{digit\}{4})</code>
1+415-555-1212	<code>1\+(\{digit\}{3})\-(\{digit\}{3})\-(\{digit\}{4})</code>

In the Patterns tab, you can select the patterns to which you would like the other patterns in the same pattern group to be converted. Below, the selected **target pattern** becomes the pattern to which other patterns in the column values are converted:

The screenshot shows the Alteryx Patterns tab for a column named 'PHONE'. The interface displays a list of patterns and their corresponding values. The 'All patterns' section shows a list of patterns and their values, with a bar chart indicating the frequency of each pattern. The 'Convert' section shows a list of values and a bar chart indicating the frequency of each value. The 'Delete rows' section shows a list of values and a bar chart indicating the frequency of each value. The 'Set' section shows a list of values and a bar chart indicating the frequency of each value. The 'Create a new column' section shows a list of values and a bar chart indicating the frequency of each value.

Source: PHONE

Overview | **Patterns**

All patterns

- Pattern: `(\{digit\}{3}) \{digit\}{3} - \{digit\}{4}` (16.07k)
- Pattern: `\{digit\}{3} \{digit\}{3} - \{digit\}{4}` (2.69k)
- Pattern: `\{digit\}{3} \{digit\}{3} - \{digit\}{4}` (1.24k)

Convert

values like
'443 871 4409'
to pattern format
'(443)871-4409'

Delete rows

with values matching `'(start)\{digit\}{3}\{digit\}{3}-(digit){4}(end)'`

Set

values matching `'(start)\{digit\}{3}\{digit\}{3}-(digit){4}(end)'` to NULL()

Create a new column

flag rows matching `'(start)\{digit\}{3}\{digit\}{3}-(digit){4}(end)'`

20 Columns | 20,000 Rows | 8 Data Types | Show only affected | Columns | Rows

NOTE: You may have to modify the phone number values before attempting the conversion, as they may contain extra alphanumeric values. For example, international country codes (such as 044) or a preceding 1+ required in long-distance numbers, may need to be extracted or removed from the column values prior to conversion.

Generic Conversions

Below are types of conversions that are supported and not supported.

Supported:

Example Source Value	Example Target Value	Notes
123.456.7890	123-456-7890	Changing symbolic characters
(123) 456-7890	123 456-7890	Removing symbolic characters
(123)456-7890	(123)-456-7890	Adding symbolic characters
1234567890	123-456-7890	Splitting a long character group and adding symbolic characters
123-456-7890	1234567890	Merging multiple character groups and removing symbolic characters

Not supported:

Example Source Value	Example Target Value	Notes
123.456.7890	+1.123.456.7890	Adding a new character group
+1.123.456.7890	123.456.7890	Deleting a character group (alphanumeric characters cannot be deleted through pattern standardization)
Adam Wilson	A Wilson	Partial deletion of data from a character group
+1 (123) 456-7890	+001 (123) 456-7890	Prepending or appending a character group with specified characters

Datetime Patterns

For columns of Datetime type, the available Convert mappings are based upon the supported date formats in the platform. Standardization of Datetime patterns is a specific implementation.

Notes on Datetime patterns:

Two-digit years (YY) do not yield four-digit year (YYYY) suggestions due to ambiguity. For example, it is unclear if 50 should map to 1950 or 2050.

For performance reasons, a maximum of two semantic standardizations can be applied at once. Examples:

Source Value	Possible Standardization	Semantic Mappings	Status
Jan 1, 1981	01/01/1981	<ul style="list-style-type: none">Jan 011 01	ok (2 mappings)
Jan 1, 1981	01/01/81	<ul style="list-style-type: none">Jan 011 011981 81	Not suggested (3 mappings)

Tip: Use the DATEFORMAT function to convert Datetime values to different date formats.

For more information on supported formats, see *Datetime Data Type*.

Patterns by Example

You can generate a new column of values based on pattern matches from a source column. When you enter example values to match with source values, other values with similar patterns may also be matched based on your entered example value.

Tip: This method provides an easy way to build pattern-based matching for values in a source column.

For more information on transformation by example, see *Overview of TBE*.

Modify String Values

Contents:

- *Convert Columns to String*
 - *Available string functions*
 - *Example - Clean up Strings*
 - *Trim strings*
 - *Use missing or mismatched value presets*
 - *Remove a specific sub-string*
 - *Replace double spaces*
 - *Break out CamelCase*
 - *Reduce strings by words*
 - *Other String Cleanup Transformations*
 - *Trim whitespace from text*
 - *Remove whitespace*
 - *Remove symbols*
 - *Remove accents*
 - *Trim quotes*
 - *Pad Values*
 - *Add prefix or suffix to strings*
 - *Standardize String Values*
 - *Standardize case*
 - *Standardize String Lengths*
 - *Pad string values*
 - *Fixed length strings*
 - *Manage Sub-Strings*
 - *Reset Types*
-

This section describes techniques to standardize text values in your datasets. You can use the following techniques to address some common issues you might encounter in the standardization of text and other non-numeric values.

Convert Columns to String

For manipulation of individual values, it is often easiest to work with the String data type, which is the most flexible. Depending on your approach, you may choose to convert some of your columns into String type:

Transformation Name	Change column type
Parameter: Columns	col1,col2, col3
Parameter: New type	'String'

For more information, *Valid Data Type Strings*.

Available string functions

You can edit values in a column by applying one of the available string functions. The following transformation can be modified for any of the available string functions:

Transformation Name	Edit column with formula
Parameter: Columns	myCol

Figure: Example data after import

Notes:

- You can see that white space is demarcated in the imported data. In particular, the line item with two spaces between the words is accurately represented in the data grid.
- Newlines, carriage returns, tabs, and other non-visible characters are represented with icons.

To normalize these text values, you can use some of the techniques listed on this page to match the problematic string values in this dataset and correct them, as needed. The sections below outline a number of techniques for identifying matches and cleaning up your data.

Trim strings

NOTE: Before you begin matching data, you should perform a `TRIM` transform to remove whitespace at the beginning and end of the string, unless the whitespace is significant to the meaning and usage of the string data.

When transforming strings, a key step is to trim off the whitespace at the beginning and ending of the string. For the above dataset, you can use the following command to remove these whitespaces:

Transformation Name	Edit column with formula
Parameter: Columns	All
Parameter: Formula	TRIM(\$col)

The above transform uses the following special values, which are available for some transforms like `set`:

Special Value	Description
*	For the Columns textbox under Advanced, you can use this wildcard to reference all columns in the dataset. <div>Tip: You can also select <code>All</code> from the Columns drop-down.</div>
\$col	When multiple columns are referenced in a transform, this special value allows you to reference the source column in a replacement value.

The previewed data looks like the following, in which five strings are modified and now match the base string:

The screenshot shows the Trifacta Wrangle interface. The main workspace displays a data table with columns 'Source', 'to be dropped', and 'Preview'. The 'Preview' column shows the result of the TRIM(\$col) formula, which removes leading and trailing whitespace from the 'Source' column. The right sidebar shows the recipe configuration with the formula 'TRIM(\$col)' and options to group and sort rows by column.

Figure: Trim data to improve matches

Tip: To remove all whitespace, including spaces in between, you can use the REMOVEWHITESPACE function.

Use missing or mismatched value presets

The platform language, Wrangle, provides presets to identify missing or mismatched values in a selection of data.

Tip: In a column's histogram, click the missing or mismatched categories to trigger a set of suggestions.

Missing values preset: The following transform replaces missing URL values with the text string `http://www.example.com`. The preset `ISMISSING([Primary_WebSite_or_URL])` identifies the rows missing data in the specified column:

Transformation Name	Edit column with formula
Parameter: Columns	Primary_Website_or_URL
Parameter: Formula	<code>IF(ISMISSING([Primary_Website_or_URL]), 'http://www.example.com', \$col)</code>

For more information, see *Find Missing Data*.

NOTE: If the data type for the column is URL, then the replacement text string must be a valid URL, or the new data is registered as mismatched with the data type.

Mismatched values preset: This transform converts to 00000 all values in the Zip column that are mismatched against the Zipcode data type. In this case, the preset `ISMISMATCHED(Zip, ['Zipcode'])` identifies the mismatched values in the column, as compared to the Zipcode data type:

--	--

Transformation Name	Edit column with formula
Parameter: Columns	Zip
Parameter: Formula	IF(ISMISMATCHED(Zip, ['Zipcode']), '00000', \$col)

For more information, see *Find Bad Data*.

Remove a specific sub-string

An entry in the example data contains an additional word: `My String extra`. You can use a simple replace command to remove it:

Transformation Name	Replace text or patterns
Parameter: Column	String
Parameter: Find	' extra'
Parameter: Replace with	' '
Parameter: Match all occurrences	true

The `global` parameter causes the replacement to be applied to all instances found within a cell value. Otherwise, the replacement occurs only on the first instance.

Replace double spaces

There are multiple ways of removing double spaces, or any pattern, from text values. For best results, you should limit this change to individual columns.

NOTE: For matching string patterns that are short in length, you should be careful to define the scope of match. For example, to remove double spaces from your dataset, you should limit the columns to just the ones containing string values. If you applied the change to all columns in the dataset, meaningful uses of double spacing could be corrupted, such as in JSON data fields.

Transformation Name	Replace text or patterns
Parameter: Column	String
Parameter: Find	' '
Parameter: Replace with	' '
Parameter: Match all occurrences	true

- In the above, the Find term contains a string with two spaces in it.

Tip: If you wish to find two or more spaces, you can use the following Pattern in the Find parameter:

```
`( )+`
```

- The Replace term contains no spaces.

Break out CamelCase

CamelCase refers to text in which multiple words are joined together by removing the spaces between them. In the example data, the entry `MyString` is an example of CamelCase.

NOTE: Regular expressions are very powerful pattern-matching tools. If they are poorly specified in a transform, they can have unexpected results. Please use them with caution.

You can use `Patterns` to break up CamelCase entries in a column of values. The following transforms use regular expressions to identify patterns in a set of values:

Transformation Name	Replace text or patterns
Parameter: Column	String
Parameter: Find	`({alpha})({upper})`
Parameter: Replace with	'\$1 \$2'
Parameter: Match all occurrences	true

The first transform locates all instances of uppercase letters followed by lower-case letters. Each instance is replaced by a space, followed by the found string (\$2). For more information, see *Text Matching*.

Reduce strings by words

Remove last word:

For example, you need to remove the last word of a string and the space before it. You can use the following `replace` transform to do that:

Transformation Name	Replace text or patterns
Parameter: Column	String
Parameter: Find	` {alpha}+{end}`
Parameter: Replace with	' '
Parameter: Match all occurrences	true

When the above is previewed, however, you might notice that ending punctuation is not captured. For example, periods, exclamation points, and question marks at the end of your values are not captured in the `Pattern`. To capture those values, the `Find` parameter must be expanded:

Transformation Name	Replace text or patterns
Parameter: Column	String
Parameter: Find	` {alpha}+([?!.;\]) {end}`
Parameter: Replace with	' '
Parameter: Match all occurrences	true

In the second version, a capture group has been inserted in the middle of the `on` parameter value, as specified by the contents of the parentheses:

- The bracket-colon notation denotes a set of possible individual characters that could appear at this point in the pattern.
 - Note the backward slash before the right parenthesis in the capture group. This value is used to escape a value, so that this parenthesis is interpreted as another character, instead of the end of the capture group.
- The vertical pipe (`|`) denotes a logical OR, meaning that the specified individual characters could appear or the value after the vertical pipe.
- Since the value after the vertical pipe is missing, this capture group finds values with or without punctuation at the end of the line.
- A **capture group** is a method of grouping together sequences of characters as part of a matching pattern and then referencing them programmatically in any replacement value. For more information, see *Capture Group References*.

Reduce total number of words:

You need to cut each value in a column down to a maximum of two words. You can use the following to identify the first two words using capture groups in a `Pattern` and then write that pattern back out, dropping the remainder of the column value:

Transformation Name	Replace text or patterns
Parameter: Column	String
Parameter: Find	<code>`{start}({alpha}*)({alpha}*) ({any}*{end})`</code>
Parameter: Replace with	<code>'\$1\$2'</code>
Parameter: Match all occurrences	true

For the Find pattern:

- The `start` pattern identifies the start of each value in the `String` column.
- The two `alpha` capture groups identify the first two words in the string. Note that the space after the second capture group is specified outside of the capture group; if it was part of the capture group, a trailing space is written in the replacement value.
- The final capture group identifies the remainder of the value in the cell.
 - `any` captures any single character.
 - The wildcard asterisk captures all values between the `any` character and the end of the value.

Other String Cleanup Transformations

Trim whitespace from text

You can trim out whitespace from an individual column via transformation. The `TRIM` function applied to string values removes the leading and trailing whitespace:

Transformation Name	Edit column with formula
Parameter: Columns	myCol
Parameter: Formula	<code>TRIM(myCol)</code>

To apply this function across all columns in the dataset, you can use the following:

Transformation Name	Edit column with formula
Parameter: Columns	All
Parameter: Formula	TRIM(\$col)

Notes:

- Instead of All above, you can use the asterisk (*) **wildcard**, which represents all possible value. In this case, both values for Columns matches with all column names in the dataset.
- You may need to move columns or use range values to apply this transformation to only non-numeric column types.
- The \$col entry denotes a reference to the current column. So for any column to which this transformation is applied, the source values are pulled from the column itself and then trimmed.

In some cases, you may wish to remove all spaces, including those in between words or digits, in your strings:

Transformation Name	Edit column with formula
Parameter: Columns	All
Parameter: Formula	REMOVEWHITESPACE(\$col)

Remove whitespace

If needed, you can remove all whitespace from a column of values.

NOTE: This transformation differs from the TRIM function, which removes only the whitespace at the beginning and end of the string. This transformation removes all whitespace, including space in the middle of the string.

Tip: For some of the string comparison functions, you may achieve better results by comparing strings without whitespace.

Transformation Name	Remove whitespace
Parameter: Columns	name
Parameter: Format	Remove all whitespace

Remove symbols

The following transformation removes all non-alphanumeric symbols from your string values, including:

- Punctuation
- Numeric value indicators (\$, %, etc.)

NOTE: Accented characters may not be removed. If this function fails to remove specific symbols, you may need to remove these symbols manually or change the input encoding on the dataset through the Import Data page.

Transformation Name	Remove symbols
Parameter: Columns	All

Parameter: Format	Remove symbols
--------------------------	----------------

Remove accents

The following transformation converts all accented characters (e.g."ä") to unaccented characters (e.g "a").

Transformation Name	Remove accents from text
Parameter: Columns	All
Parameter: Format	Remove accents

Trim quotes

When some files are imported into the application, leading and trailing quotes may remain for some or all columns. You can use the following transformation to remove these quotes from all columns:

NOTE: Quotes that appear in the middle of the string value are not removed. Single quotes, such as apostrophes, are not removed.

Transformation Name	Trim quotes
Parameter: Columns	All
Parameter: Format	Trim leading and trailing quotes

Pad Values

Add prefix or suffix to strings

You can add fixed-string prefixes or suffixes to your string values. The following adds -0000 to a text version of the Zipcode column:

Transformation Name	Add suffix to text
Parameter: Columns	txtZipCode
Parameter: Format	Add suffix
Parameter: Text to add	'-0000'

Standardize String Values

Standardize case

You can use the following steps to set all text values in a column to be the same case.

Lower case:

Transformation Name	Edit column with formula
Parameter: Columns	myStrings
Parameter: Formula	LOWER(myStrings)

Upper case:

Transformation Name	Edit column with formula
Parameter: Columns	myStrings
Parameter: Formula	UPPER(myStrings)

Proper (sentence) case:

Transformation Name	Edit column with formula
Parameter: Columns	myStrings
Parameter: Formula	PROPER(myStrings)

Standardize String Lengths

Pad string values

If you need all of your column values to be of the same length, one technique is to pad each string value at the front sufficiently, such that all string lengths in the column are identical.

This transformation results in adding enough spaces to each row value until the length of each value is 50 characters.

NOTE: Strings that are longer than the prescribed maximum are unchanged. You can use the `LEFT` or `RIGHT` functions to change the size of the oversized ones. See below.

Transformation Name	Pad text with leading characters
Parameter: Columns	MyStrings
Parameter: Format	Pad with leading characters
Parameter: Character to pad with	' '
Parameter: Length	50

Fixed length strings

You can limit the maximum size of a column or set of columns to a fixed string length. For example:

Transformation Name	Edit column with formula
Parameter: Columns	col1,col2
Parameter: Formula	IF(LENGTH(\$col)>32,LEFT(\$col,32),\$col)

In the above, if the length of either column is longer than 32 characters, then the column value is set to the leftmost 32 characters. For shorter strings, the entire string is used.

For more information, see *Manage String Lengths*.

Manage Sub-Strings

You can use the following functions to locate values within your strings. These functions can be used as part of New Formula or Edit Formula transformations to create or edit column content:

Function Name	Description
<i>LEN Function</i>	Returns the number of characters in a specified string. String value can be a column reference or string literal.
<i>FIND Function</i>	Returns the index value in the input string where a specified matching string is located in provided column, string literal, or function returning a string. Search is conducted left-to-right.
<i>RIGHTFIND Function</i>	Returns the index value in the input string where the last instance of a matching string is located. Search is conducted right-to-left.
<i>LEFT Function</i>	Matches the leftmost set of characters in a string, as specified by parameter. The string can be specified as a column reference or a string literal.
<i>RIGHT Function</i>	Matches the right set of characters in a string, as specified by parameter. The string can be specified as a column reference or a string literal.
<i>SUBSTRING Function</i>	Matches some or all of a string, based on the user-defined starting and ending index values within the string.
<i>SUBSTITUTE Function</i>	Replaces found string literal or pattern or column with a string, column, or function returning strings.

Reset Types

After modifying non-text values as strings, remember to convert them back to their original types.

Manage String Lengths

Contents:

- *Test String Length*
- *Truncate Strings*
- *Specialized String Lengths*
- *Use Rightmost Values*
- *Substring Values*
- *Additional String Functions*

In this example, your target system has a limit on the maximum length for the First Name and Last Name fields. You can use the following transforms to evaluate and truncate your strings based on their length.

Test String Length

You can use the following command to write a `TOO LONG` message when the length of the `first_name` field exceeds 32 characters:

Transformation Name	Edit column with formula
Parameter: Columns	String_test
Parameter: Formula	IF(LEN(first_name) > 32, 'TOO LONG',String_test)

Truncate Strings

The above test allows you to evaluate individual strings that are too long to see if they are errors or can somehow be shortened. For a large dataset in which you cannot easily solve these problems, you can simply choose to cut off the length of a string at 32 characters:

Transformation Name	Edit column with formula
Parameter: Columns	*
Parameter: Formula	LEFT(\$col,32)

In the above, you can use a wildcard to match all columns in the dataset. The replacement value is defined to be the first 32 characters of the source column (`$col`). By definition of the `LEFT` function, columns that are shorter than 32 characters in length are untouched.

Tip: If the field you are truncating is used as a key to your dataset, you should verify that your key still contains unique values after you have applied the truncation. For example, if the combination of `first_name` and `last_name` is a unique identifier in your dataset, you should verify that the column containing these identifiers contains unique values.

Specialized String Lengths

In some cases, you might want to limit the lengths of text strings. In this example, your dataset contains a column of zip code values, some of which are in Zip+4 format. Your source data might look like the following:

zip_code

94104
94104-2218
94105

For consistency, you might want to limit the column to use just the first five digits of the zip code.

Steps:

1. Select the first five digits of one of the nine-digit zip codes.
2. In the suggestion cards, select the Extract card.
3. Select the following variation:

Transformation Name	Extract text or pattern
Parameter: Column to extract from	zipcode
Parameter: Option	Custom text or pattern
Parameter: Text to extract	`{zip}`
Parameter: Start extracting after	`{start}`

4. Click **Add**.

The above solution references two Patterns to identify elements of the cell value. For more information, see *Text Matching*.

For a more generalized approach, you can use some of the following string functions to limit your data length. Values that are shorter than the designated string length are left untouched.

NOTE: Transforms that cut down the size of a value might generate mismatched or missing values based on the column's data type. You should verify that you are not creating new missing or mismatched values.

Use Rightmost Values

Use the following transform to reduce a string to the rightmost 6 characters in any value:

Transformation Name	Edit column with formula
Parameter: Columns	prodID
Parameter: Formula	RIGHT(prodID, 6)

Substring Values

The `SUBSTRING` function enables you to designate a specific subset of the string's characters to use. You specify the index of the first character in the values and the number of subsequent characters to include. For example, when applied to the value `United States of America` in the `countries` column, the following transform sets the new value to be `States`.

Transformation Name	Edit column with formula
Parameter: Columns	countries

Parameter: Formula	SUBSTRING(<i>countries</i> , 7, 6)
---------------------------	-------------------------------------

Note that the index value begins at zero; to extract from the beginning of the value, replace 7 above with 0.

Additional String Functions

Wrangle supports other functions, which can be used to transform string values. See *String Functions*.

Extract Values

Contents:

- *Extract vs. Split*
- *Extract methods*
- *Extract text or patterns*
 - *Extract single values*
 - *Extract values by example*
 - *Constrain matching*
 - *Extract single patterns*
 - *Extract multiple values*
 - *Extract first or last characters*
 - *Extract by positions*
- *Extract by Data Type*
 - *Extract date values*
 - *Extract numeric values*
 - *Extract components of a URL*
 - *Extract object values*
 - *Extract array values*
- *Extract Values into a List*
 - *Extract matches into array*
 - *Extract hashtags*

Extracting one or more values from within a column of values can turn data into meaningful and discrete information. This section describes how to extract column data, the methods for which may vary depending on the data type.

Extract vs. Split

Extract and split transformations do not do the same thing:

- A **split** transformation separates a single column into one or more separate columns based on one or more values in the source column that identify where the data should be split. These delimiters can be determined by the application or specified by the user when defining the transformation.
- An **extract** transformation matches literal or pattern values from a source column and stores it in a separate column.

NOTE: The source column is untouched by extract transformations.

Extract methods

In the Transformer page, you can use the following methods to extract values:

Method	Description
By selection	Select part of a value in the data grid to prompt a series of suggestions on what to do with the data. Typically, extract options are near the top of the suggestions when you select part of a value.
By column menu	From the menu to the right of the column, select Extract and a sub-menu item to begin configuring a transformation.
By Transformer toolbar	At the top of the data grid, click the Extract icon in the Transformer toolbar to begin configuring extract transformations.

Extract text or patterns

A primary use of extraction is to remove literal or patterned values of text from a column of values. Suppose your dataset included a column of LinkedIn updates. You can use one of the following methods to extract keywords from these values.

Extract single values

The following example transformation extracts the word `#bigdata` from the column `msg_LinkedIn`:

Transformation Name	Extract text or pattern
Parameter: Column to extract from	<code>msg_LinkedIn</code>
Parameter: Option	Custom text or pattern
Parameter: Text to extract	<code>'#bigdata'</code>
Parameter: Number of matches to extract	1

Notes:

- The `option` parameter identifies that the pattern to match is a custom one specified by the user.
- The `Number of matches to extract` parameter defaults to 1, meaning that the transformation extracts a maximum of one value from each cell. This value can be set from 1-50.

Extract values by example

You can generate a new column of values extracted from a source column by entering example values to match with source values. Values with similar patterns may also be matched based on your entered example value.

Tip: This method provides an easy way to build pattern-based matching for values in a source column.

For more information on transformation by example, see *Overview of TBE*.

Constrain matching

Within the extract transformation, you can specify literals or patterns before or after which the match is found. This method can be used to remove parts of each cell value from erroneously matching on the literal or pattern that is desired.

The following example extracts the second three-digit element of a phone number, skipping the area code:

Transformation Name	Extract text or pattern
Parameter: Column to extract from	<code>phone_num</code>
Parameter: Option	Custom text or pattern
Parameter: Text to extract	<code>`{digit}`</code>
Parameter: Number of matches to extract	1

Parameter: Ignore matches between	<code>`{start}{digit}{3}\-`</code>
-----------------------------------	------------------------------------

Extract single patterns

You can also do pattern-based extractions using Trifacta patterns or regular expressions.

- **Regular expressions** are a standards-based method of describing patterns of characters for matching purposes. Regular expressions are very powerful but can be difficult to use.
- A **Trifacta pattern** is a proprietary method of describing patterns, which is much simpler to use than regular expressions.
- For more information on both types of patterns, see *Text Matching*.

The following example extracts all words that begin with # in the `msg_LinkedIn` column:

Transformation Name	Extract text or pattern
Parameter: Column to extract from	<code>msg_LinkedIn</code>
Parameter: Option	Custom text or pattern
Parameter: Text to extract	<code>`\#{alphanum-underscore}+`</code>
Parameter: Number of matches to extract	50

Notes:

- The `Text to extract` parameter has changed:

Element	Description
Two back-ticks (`)	Indicate that the expression between them represents a Trifacta pattern.
<code>\#</code>	The slash indicates that the character right after it should be interpreted as a character only; it should not be interpreted as any special character in the pattern.
<code>{alphanum-underscore}</code>	This Trifacta pattern element is used to indicate a single alphanumeric or underscore character.
<code>+</code>	Adding the plus sign after the above character signifies that the pattern can match on a sequence of alphanumeric or underscore characters of one or more length.

- The `Number of matches to extract` parameter has been increased to grab up to 50 hashtags.

Advanced options

Option	Description
Number of patterns to extract	<p>Set this value to the total number of patterns you wish to extract.</p> <div> <p>NOTE: This value determines the number of columns that are generated by the extraction. If no value is available, an empty value is written into the corresponding column.</p> </div> <p>The default is 1.</p>
Ignore case	By default, pattern matching is case-sensitive. Select this checkbox to ignore case when matching.
Ignore matches between	You can enter a pattern here to describe any patterns that should not be part of any match. This option is useful if you have multiple instances of text but want to ignore the first one, for example.

Extract multiple values

In your pattern expressions, you can use the vertical pipe character (|) to define multiple patterns to find. The following example extracts any value from the `myDate` column that ends in 7 or 8:

Transformation Name	Extract text or pattern
Parameter: Column to extract from	myDate
Parameter: Text to extract	`{any}+7 {any}+8`
Parameter: End extracting before	`{end}`

You can use the vertical pipe in both Trifacta patterns and regular expressions.

Extract first or last characters

You can extract the first or last set of characters from a column into a new column. In the following example, the first five characters from the `ProductName` column are extracted into a new product identifier column:

Transformation Name	Extract by positions
Parameter: Column to extract from	ProductName
Parameter: Option	First characters
Parameter: Number of characters to extract	5

You can change the Option value to `Last characters` to extract from the right side of the column value.

Extract and remove

If you need to remove the characters that you extracted, you can use the following transformation. In this case, the first five characters, which were extracted in the previous transformation, are removed:

Transformation Name	Edit column with formula
Parameter: Columns	ProductName
Parameter: Formula	<code>RIGHT(ProductName, LEN(ProductName)-5)</code>

Extract by positions

You can extract values between specified index positions within a set of column values. In the following example, the text between the fifth and tenth characters in a column are extracted to a new column.

Tip: This extraction method is useful if the content before and after the match area is inconsistent and cannot be described using patterns. If it is consistent, you should use the Extract text or pattern transformation.

Transformation Name	Extract by positions
Parameter: Column to extract from	ProductName
Parameter: Option	Between two positions

Parameter: Starting position	5
Parameter: Ending position	10

Extract by Data Type

You can perform extractions that are specific to a data type or based on failures of the data to match a specified data type.

Extract date values

You can use functions to extract values from Datetime columns. The example below extracts the year value from the `myDate` column:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>YEAR(myDate)</code>
Parameter: New column name	<code>myYear</code>

The following functions can be used to extract values from a Datetime column, as long as the values are present in the formatted date:

Function Name	Description
<i>DAY Function</i>	Derives the numeric day value from a Datetime value. Source value can be a reference to a column containing Datetime values or a literal.
<i>MONTH Function</i>	Derives the month integer value from a Datetime value. Source value can be a reference to a column containing Datetime values or a literal.
<i>YEAR Function</i>	Derives the four-digit year value from a Datetime value. Source value can be a reference to a column containing Datetime values or a literal.
<i>HOURL Function</i>	Derives the hour value from a Datetime value. Generated hours are expressed according to the 24-hour clock.
<i>MINUTE Function</i>	Derives the minutes value from a Datetime value. Minutes are expressed as integers from 0 to 59.
<i>SECOND Function</i>	Derives the seconds value from a Datetime value. Source value can be a reference to a column containing Datetime values or a literal.

You can also reformat the whole Datetime column using the `DATEFORMAT` function. The following reformats the column to show only the two-digit year:

Transformation Name	Edit column with formula
Parameter: Columns	<code>myDate</code>
Parameter: Formula	<code>DATEFORMAT(myDate, "yy")</code>

Extract numeric values

You can extract numerical data from text values. In the following example, the first number is extracted from the `address` column, which would correspond to extracting the street number for the address:

Transformation Name	Extract patterns
----------------------------	------------------

Parameter: Column to extract from	address
Parameter: Option	Numbers
Parameter: Number of matches to extract	1

Empty values in this new column might indicate a formatting problem with the address.

Tip: If you set the number of patterns to extract to 2 for the `address` column, you might extract apartment or suite information.

Extract components of a URL

URL components

Using functions, you can extract specific elements of a valid URL. The following transformation pulls the domain values from the `myURL` column:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>DOMAIN(myURL)</code>
Parameter: New column name	<code>myDomain</code>

In some cases, the function may not return values. For example, the `SUBDOMAIN` function returns empty values if there is no sub-domain part of the URL.

The following functions can be used to extract values from a set of URLs:

Function Name	Description
<i>HOST Function</i>	Finds the host value from a valid URL. Input values must be of URL or String type and can be literals or column references.
<i>DOMAIN Function</i>	Finds the value for the domain from a valid URL. Input values must be of URL or String type.
<i>SUBDOMAIN Function</i>	Finds the value a subdomain value from a valid URL. Input values must be of URL or String type.
<i>SUFFIX Function</i>	Finds the suffix value after the domain from a valid URL. Input values must be of URL or String type.
<i>URLPARAMS Function</i>	Extracts the query parameters of a URL into an Object. The Object keys are the parameter's names, and its values are the parameter's values. Input values must be of URL or String type.

Query parameters

You can extract query parameter values from an URL. The following example extracts the `store_id` value from the `storeURL` field value:

Transformation Name	Extract patterns
Parameter: Column to extract from	<code>storeURL</code>

Parameter: Option	HTTP Query strings
Parameter: Fields to extract	store_id

Extract object values

If your data includes sets of arrays, you can extract array elements into columns for each key, with the values written to each key column.

Suppose your restaurant dataset includes a set of characteristics in the `restFeatures` column in the following JSON format:

```
{
  "Credit": "Y",
  "Accessible": "Y",
  "Restrooms": "Y",
  "EatIn": "Y",
  "ToGo": "N",
  "AlcoholBeer": "Y",
  "AlcoholHard": "N",
  "TotalTables": "10",
  "TotalTableSeats": "36",
  "Counter": "Y",
  "CounterSeats": "8"
}
```

You can use the following transformation to extract the values from `TotalTableSeats` and `CounterSeats` into separate columns:

Transformation Name	Unnest Objects into columns
Parameter: Column	restFeatures
Parameter: Paths to elements - 1	TotalTableSeats
Parameter: Paths to elements - 2	CounterSeats
Parameter: Include original column name	Selected

After the above is executed, you can perform a simple sum of the `TotalTableSeats` and `CounterSeats` columns to determine the total number of seats in the restaurant.

Extract array values

In some cases, your data may contain arrays of repeated key-value pairs, where each pair would exist on a separate line. Suppose you have a column called, `Events`, which contains date and time information about the musician described in the same row of data. The `Events` column might look like the following:

```
[{"Date": "2018-06-15", "Time": "19:00"}, {"Date": "2018-06-17", "Time": "19:00"}, {"Date": "2018-06-19", "Time": "20:00"}, {"Date": "2018-06-20", "Time": "20:00"}]
```

The following transformation creates a separate row for each entry in the `Events` column, populating the other fields in the new rows with the data from the original row:

NOTE: This type of transformation can significantly increase the size of your dataset.

Transformation Name	Expand arrays into rows
Parameter: Column	Events

Extract Values into a List

You can also extract sets of values into an array list of values.

Tip: This transformation is useful for extracting types or patterns of information from a single column.

Extract matches into array

Using Trifacta patterns, you can extract the values of the column to form a new column of arrays. The following example shows the usage of {any} pattern to extract the cell values and form a new array column.

Transformation:

Transformation Name	Extract matches into Array
Parameter: Column	product
Parameter: Pattern matching elements in the list	`{any}`
Parameter: Delimiter separating each element	`,`

Results:

Before	After
socks, socks, socks	["socks", "socks", "socks"]
pants, pants	["pants", "pants"]

Extract hashtags

In this example, you extract one or more values from a source column and assemble them in an Array column.

Suppose you need to extract the hashtags from customer tweets to another column. In such cases, you can use the {hashtag} Trifacta pattern to extract all hashtag values from a customer's tweets into a new column.

Source:

The following dataset contains customer tweets across different locations.

User Name	Location	Customer tweets
James	U.K	Excited to announce that we've transitioned Wrangler from a hybrid desktop application to a completely cloud-based service! #dataprep #businessintelligence #CommitToCleanData # London
Mark	Berlin	Learnt more about the importance of identifying issues in your data—early and often #CommitToCleanData #predictivetransformations #realbusinessintelligence
Catherine	Paris	Clean data is the foundation of your analysis. Learn more about what we consider the five tenets of sound #dataprep, starting with #1a prioritizing and setting targets. #startwiththeuser #realbusinessintelligence #Paris
Dave	New York	Learn how #NewYorklife

		onboarded as part of their #bigdata #dataprep initiative to unlock hidden insights and make them accessible across departments.
Christy	San Francisco	How can you quickly determine the number of times a user ID appears in your data?#dataprep #pivot #aggregation#machinelearning initiatives #SFO

Transformation:

The following transformation extracts the hashtag messages from customer tweets.

Transformation Name	Extract matches into Array
Parameter: Column	customer_tweets
Parameter: Pattern matching elements in the list	`{hashtag}`
Parameter: New column name	Hashtag tweets

Then, the source column can be deleted.

Results:

User Name	Location	Hashtag tweets
James	U.K	["#dataprep", "#businessintelligence", "#CommitToCleanData", " # London"]
Mark	Berlin	["#CommitToCleanData", "#predictivetransformations", "#realbusinessintelligence", "0"]
Catherine	Paris	["#dataprep", "#startwiththeuser", "#realbusinessintelligence", "# Paris"]
Dave	New York	["#NewYorklife", "dataprep", "bigdata", "0"]
Christy	SanFrancisco	["dataprep", "#pivot", "#aggregation", "#machinelearning"]

Format Dates

Contents:

- *Recommended Approaches*
 - *Option 1 - Patterns in the Column Details panel*
 - *Option 2 - Patterns based on date format*
 - *Option 3 - Transformation by Example*
 - *Option 4 - Manual fixups*
 - *Custom Datetime Formats*
 - *Normalize Regional Differences*
-

Datetime values can be imported into Trifacta® in a variety of formats.

Below are just a few examples of one date in different acceptable formats:

myDate
Mar-14-2018
03/14/2018
2018-Mar-03
3/14/18
03/14/2018 00:00:00
March 14, 2018

This section describes the tools and approaches for standardizing and formatting your date values.

Recommended Approaches

When you are formatting a column of date values, you can attempt to standardize the values in the following order.

Option 1 - Patterns in the Column Details panel

Through the Column Details panel, you can review the set of patterns that match the values in your date column and select the ones to apply to standardize the values.

Steps:

1. From the column menu for your date column, select **Column Details**.
2. In the Column Details panel, click the Patterns tab.
3. In the Patterns tab, you can review the set of patterns that describe all values that appear in the column. Select one that needs to be corrected.
4. In the right panel, select the Convert card.

Tip: If you do not see the Convert card, you might try to generate a new random sample, in which example patterns are more evenly distributed throughout the sample.

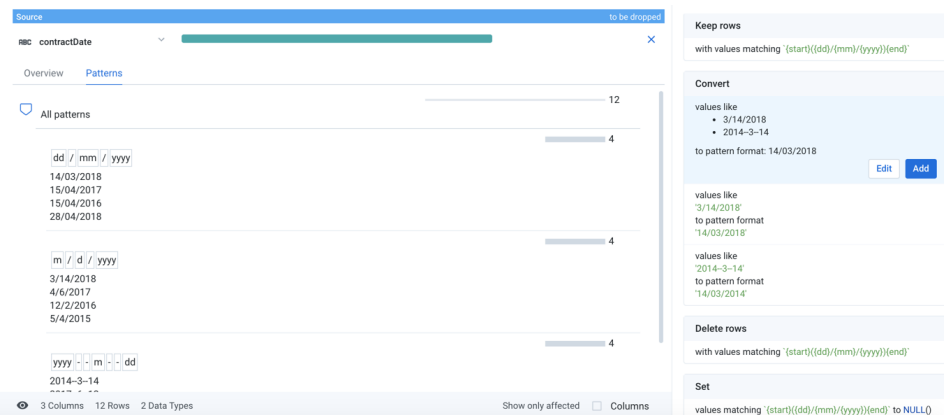


Figure: Select the Convert card in the Patterns tab

5. Click **Add**.
6. The number of patterns displayed in the Patterns tab is reduced. You can continue to select patterns to standardize values.
7. Iterate until there is only one pattern displayed in the panel.

For more information on Datetime patterns, see *Standardize Using Patterns*.

Option 2 - Patterns based on date format

In some cases, you may not be able to simply select patterns, which generates sufficient suggestions to standardize your date values. A second approach involves keying on mismatched values in the column.

Tip: This technique works for columns in which all values are valid Datetime values but are in different date formats. If you have values that are invalid for any date format, you must use Option 3 to correct the syntax errors using patterns first. See below.

In this case, you set the data type for the column to Datetime and use the DATEFORMAT function to match the format of the values that you want to change. Next to the values from the preceding table, you can see the corresponding date format token:

myDate	DATEFORMAT value
Mar-14-2018	MMM-dd-yyyy
03/14/2018	MM/dd/yyyy
2018-Mar-03	yyyy-MMM-dd
3/14/18	M/d/yy
03/14/2018 00:00:00	MM/dd/yyyy HH:mm:ss
March 14, 2018	MMMM dd, yyyy

For purposes of this example, suppose your `myDate` column contains values in `MM/dd/yyyy` and `M/d/yy` format. You wish to standardize on `MMMM dd, yyyy` format.

Steps:

1. From the Data Type menu at the top of the `myDate` column, select **Date/Time**.

2. In the dialog, select the Date format that matches values you wish to fix:

Date / Time Type ✕

☐ mm

☐ yy

☐ mm-yy

☐ mm-dd

☐ dd-mm

☒ mm-dd-yy

☐ dd-mm-yy

☐ yy-mm-dd

☐ yy-dd-mm

☐ mm-dd-yy hh:mm:ss

☐ dd-mm hh:mm:ss

☐ mm-dd hh:mm:ss

☐ dd-mm-yy hh:mm:ss

☐ yy-mm-dd hh:mm:ss

☐ yy-dd-mm hh:mm:ss

☐ hh:mm:ss

month*dd*yyyy

⌵

Cancel

Save

Figure: Date/Time format selector

3. Click **Save**.
4. Now, you need to modify the values that match this format to match the target format (MMMM dd, yyyy). Click the green bar in the column, which matches the values for the currently valid Datetime format., Then click the Set suggestion. Click **Modify**.
5. In the Transform Builder, you have a predefined transformation that sets values based on whether the column values are valid for the currently specified data type and format. You must replace the `NULL()` entry with the `DATEFORMAT` function which changes these values to the proper format:

Transformation Name	Edit with formula
Parameter: Columns	myDate
Parameter: Formula	<code>ifvalid(\$col, ['Datetime','yy','yyyy'], dateformat(\$col, 'MMMM dd, yyyy'))</code>

6. Click **Add**. All values that matched the `MM/dd/yyyy` format are converted to the `MMMM dd, yyyy` format.
7. Repeat the previous steps:
 - a. Set the column's Datetime format to: `M/d/yyyy`.
 - b. Select the green bar in the column data quality bar.
 - c. Select the Set suggestion and modify it.
 - d. For the value in the transformation, insert the following function:

```
ifvalid($col, ['Datetime','M/d/yyyy'], dateformat(myDate, 'MMMM dd, yyyy'))
```

- e. Add the transformation to you recipe.
8. Repeat Step 7 for any other mismatched formats.
 9. You may have some manual fixups to complete at the end. See below.

Option 3 - Transformation by Example

You can reformat dates by providing example output values for a listed source value. For a column of date values, you can begin providing example outputs for individual values, and Trifacta can perform pattern-based transformations to similarly formatted values. For more information, see *Overview of TBE*.

Option 4 - Manual fixups

Steps:

1. Now that you have selected a specific format for your Datetime values, the rows that do not match this format are now identified as mismatched in the column. Click the red bar at the top of the column.
2. In the Status bar at the bottom of the screen, click **Show only affected rows**.
3. You can now see only the rows that remain mismatched with respect to the preferred Datetime format.
4. Select one of these values. For example, suppose you have quite a few values that are only four-digit year values (YYYY). Select one of the values. Then, select the Replace card. Click **Edit**.
5. Your transformation should look like the following:

Transformation Name	Replace text or patterns
Parameter: Column	UpdateTime
Parameter: Find	`{start}{digit}{4}{end}`
Parameter: Replace with	' '

6. You can modify the search and replace patterns to capture and write back the year value:
 - a. In the Find value, put parentheses around the pattern that captures the four digits in a row. Adding parentheses around a matching pattern identifies that sub-pattern as a **capture group**, which can be referenced in any replacement.
 - b. The capture group should look like the following:

```
((digit){4})
```

- c. For the Replace with value, you must insert a month and day value according to the format selected for the column (MM/DD/YYYY), followed by a reference back to the capture group.
- d. Capture groups from the matching pattern can be referenced in the replacement value using references such as \$1, \$2, \$3, and so on. These tokens refer to the first, second, and third capture groups in the Find value.
- e. The Replace value should look like the following:

```
01/01/$1
```

- f. Your transformation should look like the following when done:

Transformation Name	Replace text or patterns
Parameter: Column	UpdateTime
Parameter: Find	`{start}((digit){4}){end}`
Parameter: Replace with	01/01/\$1

7. Click **Add**.
8. You can repeat these steps for the remaining mismatched values.

Custom Datetime Formats

You can create your own customized Datetime formats using the `DATEFORMAT` function. For example, the following changes the format of the `lastDate` function to use the `yyyy:MM:dd` format:

Transformation Name	Edit with formula
Parameter: Columns	lastDate
Parameter: Formula	<code>DATEFORMAT(lastDate, 'yyyy:MM:dd')</code>

Normalize Regional Differences

The following date values correspond to the same date but vary in format in different regions of the world:

Date Value	Region
03/14/2018	U.S.
14/03/2018	E.U.
2014-03-14	China

In the above examples, the delimiters for the U.S. and E.U. values are identical, which makes parsing these values more challenging.

Tip: If your dataset contains date values from different regions of the world, you should find or create a separate column to identify the applicable region.

Suppose the previous set of dates was represented in your dataset with the following values:

contractDate	region
03/14/2018	USA
14/03/2018	EU
2014-03-14	CHN

In this case, you might try the following generalized solution. You can use conditional transformations to extract the day, month, and year values from the `contractDate` column based on the value in the `region` column.

NOTE: This solution assumes that all date values within for a specific region (e.g. USA) are consistently formatted. You should perform those formatting actions first.

Steps:

1. First, you must split the column based on the cell value's delimiter. Note that the following transformation uses the Pattern `{delim}` to locate the delimiter in the cell value. This delimiter is either a dash or a slash.

Transformation Name	Split by delimiter
Parameter: Column	contractDate
Parameter: Option	by Delimiter
Parameter: Delimiter	<code>`{delim}`</code>

2. Create the following three conditional transformations for extracting the day, month, or year values based on the value in the Region column. Here is the transformation to acquire the year values:

Transformation Name	conditions
Parameter: Condition type	Case on single column
Parameter: Column to evaluate	Region
Parameter: Case 1	'EU'
Parameter: Value 1	contractDate3
Parameter: Case 2	'USA'
Parameter: Value 2	contractDate3
Parameter: Case 3	'CHN'
Parameter: Value 1	contractDate1

3. For month:

Transformation Name	conditions
Parameter: Condition type	Case on single column
Parameter: Column to evaluate	Region
Parameter: Case 1	'EU'
Parameter: Value 1	contractDate2
Parameter: Case 2	'USA'
Parameter: Value 2	contractDate1
Parameter: Case 3	'CHN'
Parameter: Value 1	contractDate2

4. For day:

Transformation Name	conditions
Parameter: Condition type	Case on single column
Parameter: Column to evaluate	Region
Parameter: Case 1	'EU'
Parameter: Value 1	contractDate1
Parameter: Case 2	'USA'
Parameter: Value 2	contractDate2
Parameter: Case 3	'CHN'
Parameter: Value 1	contractDate3

5. You can now bring together these three columns:

Transformation Name	Merge columns
Parameter: Columns	day, month, year
Parameter: Separator	' / '
Parameter: New column name	newDate

6. You now have your new date column. You may need to reformat it into a preferred format.

7. Delete the columns that were created during this process.

Apply Conditional Transformations

Contents:

- *Single- and Multi-Case Transformations*
 - *Conditional Functions*
 - *IF function*
 - *CASE function*
 - *Logical Operators*
-

In your recipe steps, you can apply conditional logic to determine if transformational changes should occur.

You can build logical tests into your transformations in multiple levels:

- **Single- and multi-case transformations:** Use case-based transformations to test if-then or case logic against your dataset and to apply the specified results.
- **Conditional functions:** IF and CASE functions can be applied to any transformation that accepts functional expressions.
- **Logical operators:** You can use AND or OR logic to build your conditional expressions.

NOTE: If you are running your job on Spark, avoid creating single conditional transformations with deeply nested sets of conditions. On Spark, these jobs can time out, and deeply nested steps can be difficult to debug. Instead, break up your nesting into smaller conditional transformations of multiple steps.

Single- and Multi-Case Transformations

Through the Transform Builder, you can build conditional tests using if/then/else or case logic to manipulate on the data.

1. In the Search panel in the Transformer page, enter `case`.
2. You can choose one of three different logical transformations:
 - a. **If-then-else:** Specify any logical test that evaluates to `true` or `false` and specify values if `true` (then) or if `false` (else).
 - b. **Single-column case:** Test for explicit values in a column and, if true, write specific values to the new column.
 - c. **Custom conditions:** Specify any number of case statements, which can have completely independent expressions:
 - i. Case 1 is tested, and a value is written if `true`.
 - ii. If Case 1 is false, then Case 2 is tested. If `true`, a different value can be written.
 - iii. Supports an arbitrary number of independent conditional cases.
3. Specify the other parameters, including the name of the new column.

After the transformation is added to the recipe, actions can then be taken based on the values in this new column.

Conditional Functions

You can also apply conditional logical as part of your function definitions for other transformations.

IF function

For example, the following replaces values in the same column with `IN` if they are greater than 0.5 or `OUT` otherwise:

Transformation Name	Edit column with formula
Parameter: Columns	testCol
Parameter: Formula	IF(\$col >= 0.5, 'IN','OUT')

In the above, the token \$col is a reference back to the value defined for the column (testCol in this case). However, you can replace it with a reference to any column in the dataset.

You can use the IF function in any transformation that accepts functional inputs.

CASE function

You can chain together IF functions in the following manner:

Transformation Name	Edit column with formula
Parameter: Columns	testCol
Parameter: Formula	IF(\$col >= 0.5, 'IN',(IF(\$col >= 0.35, 'MAYBE IN','OUT')))

However, these can become problematic to debug. Instead, you can use the CASE function to assist in building more complex logical trees. The following is more legible and easier to manage:

Transformation Name	Edit column with formula
Parameter: Columns	testCol
Parameter: Formula	CASE([\$col >= 0.75, 'IN', \$col >= 0.35, 'MAYBE IN', 'OUT'])

If test	Test	Output if true
If:	\$col >= 0.75	IN
If above is false:	\$col >= 0.35	MAYBE IN
If above is false:	default	OUT

Logical Operators

Logical operators can be applied to your function expressions to expand the range of your logical tests.

In the above example, suppose you have a second column called, Paid, which contains Boolean values. You could expand the previous statement to include a test to see if Paid=true:

Transformation Name	Edit column with formula
Parameter: Columns	testCol
Parameter: Formula	CASE([(\$col >= 0.75 && Paid == true), 'IN', (\$col >= 0.35 && Paid == true), 'MAYBE IN', 'OUT'])

The above performs a logical AND operation on the two expressions in each tested case. The logical operator is &&.

You can also reference explicit functions to perform logical tests. The above might be replaced with the following:

--	--

Transformation Name	Edit column with formula
Parameter: Columns	testCol
Parameter: Formula	CASE([AND(\$col >= 0.75, Paid == true), 'IN', AND(\$col >= 0.35, Paid == true), 'MAYBE IN', 'OUT'])

Logic	Logical Operator	Logical Function
Logical AND	(exp1 && exp2)	AND(exp1,exp2)
Logical OR	(exp1 exp2)	OR(exp1,exp2)
Logical NOT	!(exp1 == exp2)	NOT(exp1,exp2)

Depending on the structure of your transformation and your preferences, either form may be used.

Prepare Data for Machine Processing

Contents:

- *Scaling*
 - *Scale to zero mean and unit variance*
 - *Scale to min-max range*
 - *Outliers*
 - *Identify outliers*
 - *Remove outliers*
 - *Change outliers to mean values*
 - *Binning*
 - *Bins of equal size*
 - *Bins of custom size*
 - *One-Hot Encoding*
-

Depending on your downstream system, you may need to convert your data into numeric values of the expected form or to standardize the distribution of numeric values. This section summarizes some common statistical transformations that can be applied to columnar data to prepare it for use in downstream analytic systems.

Scaling

You can scale the values within a column using either of the following techniques.

Scale to zero mean and unit variance

Zero mean and unit variance scaling renders the values in the set to fit a normal distribution with a mean of 0 and a variance of 1. This technique is a common standard for normalizing values into a normal distribution for statistical purposes.

In the following example, the values in the `POS_Sales` column have been normalized to average 0, variance 1.

- **Remove mean:** When selected, the existing mean (average) of the values is used as the center of the distribution curve.

NOTE: Re-centering sparse data by removing the mean may remove sparseness.

- **Scale to unit variance:** When selected, the range of values are scaled such that their variance is 1. When deselected, the existing variance is maintained.

NOTE: Scaling to unit variance may not work well for managing outliers. Some additional techniques for managing outliers are outlined below.

Transformation Name	Scale column
Parameter: Column	POS_Sales
Parameter: Scaling method	Scale to zero mean and unit variance
Parameter: Remove mean	false
Parameter: Scale to unit	true

variance	
Parameter: Output options	Create new column
Parameter: New column name	scale_POS_Sales

Scale to min-max range

You can scale column values fitting between a specified minimum and maximum value. This technique is useful for distributions with very small standard deviation values and for preserving 0 values in sparse data.

The following example scales the `TestScores` column to a range of 0 and 1, inclusive.

Transformation Name	Scale column
Parameter: Column	TestScores
Parameter: Scaling method	Scale to a given min-max range
Parameter: Minimum	0
Parameter: Maximum	1
Parameter: Output options	Replace current column

Outliers

You can use several techniques for identifying statistical outliers in your dataset and managing them as needed.

Identify outliers

Suppose you need to remove the outliers from a column. Assuming a normal bell distribution of values, you can use the following formula to calculate the number of standard deviations a column value is from the column mean (average). In this case, the source column is `POS_Sales`.

Transformation Name	New formula
Parameter: Formula type	Multiple row formula
Parameter: Formula	$(\text{ABS}(\text{POS_Sales} - \text{AVERAGE}(\text{POS_Sales}))) / \text{STDEV}(\text{POS_Sales})$
Parameter: New column name	stdevs_POS_Sales

Remove outliers

The new `stdevs_POS_Sales` column now contains the number of standard deviations from the mean for the corresponding value in `POS_Sales`. You can use the following transformation to remove the rows that contain outlier values for this column.

Tip: An easier way to select these outlier values is to select the range of values in the `stdevs_POS_Sales` column histogram. Then, select the suggestion to delete these rows. You may want to edit the actual formula before you add it to your recipe.

In the following transformation, all rows that contain a value in `POS_Sales` that is greater than four standard deviations from the mean are deleted:

--	--

Transformation Name	Filter rows
Parameter: Condition	Custom formula
Parameter: Type of formula	Custom single
Parameter: Condition	4 <= stdevs_POS_Sales
Parameter: Action	Delete matching rows

Change outliers to mean values

You can also remove the effects of outliers by setting their value to the mean (average), which preserves the data in other columns in the row.

Transformation Name	Edit with formula
Parameter: Columns	POS_Sales
Parameter: Formula	IF(stdevs_POS_Sales > 4, AVERAGE(POS_Sales), POS_Sales)

Binning

You can modify your data to fit into bins of equal or custom size. For example, the lowest values in your range would be marked in the 0 bin, with larger values being marked with larger bin numbers.

Bins of equal size

You can bin numeric values into bins of equal size. Suppose your column contains numeric values 0–1000. You can bin values into equal ranges of 100 by creating 10 bins.

Transformation Name	Bin column
Parameter: Column	MilleBornes
Parameter: Select Option	Equal Sized Bins
Parameter: Number of Bins	10
Parameter: New column name	MilleBornesRating

Bins of custom size

You can also create custom bins. In the following example, the TestScores column is binned into the following bins. In a later step, these bins are mapped to grades:

Bins	Bin Range	Bin Number	Grade
59	0-59	0	F
69	60-69	1	D
79	70-79	2	C
89	80-89	3	B
	90+	4	A
(no value)			I

First, you bin values into the bin numbers listed above:

Transformation Name	Bin column
Parameter: Column	TestScores
Parameter: Select option	Custom bin size
Parameter: Bins	59,69,79,89
Parameter: New column name	Grades

You can then use the following transformation to assign letters in the Grades column:

Transformation Name	Conditions
Parameter: Condition type	Case on single column
Parameter: Column to evaluate	Grades
Parameter: Case - 0	'F'
Parameter: Case - 1	'D'
Parameter: Case - 2	'C'
Parameter: Case - 3	'B'
Parameter: Case - 4	'A'
Parameter: Default value	'I'
Parameter: New column name	Grades_letters

One-Hot Encoding

One-hot encoding refers to distributing the listed values in a column into individual columns. Within each row of each individual column is a 0 or a 1, depending on whether the value represented by the column appears in the corresponding source column. The source column is untouched. This method of encoding allows for easier consumption of data in target systems.

Tip: This transformation is particularly useful for columns containing a limited set of enumerated values.

In the following example, the values in the `BrandName` column are distributed into separate columns of binary values, with a maximum limit of 50 new columns.

NOTE: Be careful applying this to a column containing a wide variety of values, such as Decimal values. Your dataset can expand significantly in size. Use the max columns setting to constrain the upper limit on dataset expansion.

Transformation Name	One-hot encoding of values to columns
Parameter: Column	BrandName
Parameter: Max number of columns to create	50

Tip: If needed, you can rename the columns to prepend the names with a reference to the source column.

Enrichment Tasks

Contents:

- *Add New Columns*
- *Insert Metadata*
- *Combine Datasets*
 - *Union*
 - *Join*
 - *Lookup*
- *Reshape Datasets*
 - *Aggregation*
 - *Pivot tables*

These topics cover various approaches to augmenting your data with fixed values, generated values, or data from other datasets.

Add New Columns

You can add new columns of data that you specify within the application. See *Create New Column*.

This new column can be created by combining multiple columns of existing data. See *Add Two Columns*.

Insert Metadata

You can insert data about your dataset into the dataset itself. See *Insert Metadata*.

Tip: A common use of available metadata is to create primary keys for each record in your dataset. See *Generate Primary Keys*.

Combine Datasets

Union

A **union** operation concatenates multiple datasets together. An example is below.

Tip: The following example unions two datasets based on the position of the columns. Unions may also be performed based on the column names.

Dataset 1:

CName1	CName2	CName3
C1.1	C2.1	C3.1
C1.2	C2.2	C3.2
C1.3	C2.3	C3.3

Dataset 2:

CName1	CName2	CName4

C4.1	C5.1	C6.1
C4.2	C5.2	C6.2
C4.3	C5.3	C6.3

When a union is performed based on the position of the columns in each dataset, all of the rows of Dataset 1 are included, followed by all of the rows of Dataset 2. You can choose which columns to include from each of the source datasets.

Output:

In the above, note that the name of the third column in each dataset is different (CName3 and CName4).

CName1	CName2	CName3	CName4
C1.1	C2.1	C3.1	
C1.2	C2.2	C3.2	
C1.3	C2.3	C3.3	
C4.1	C5.1		C6.1
C4.2	C5.2		C6.2
C4.3	C5.3		C6.3

When to use:

Tip: You should perform union operations as early as possible in your recipes.

- If your datasets include event or log information, you can use the union operation to create a longer sequence of those transactions. For example, you might union together all of your log data for a week from daily log files.

To union your dataset to another, enter `Union datasets` the Transformation textbox in the recipe panel.

See *Append Datasets*.

Join

A join operation brings together two datasets based on a column that appears in both datasets and contains the same unique values used to identify records. Based on the values in this column, called the **primary key**, records in the second dataset are joined to records in the first dataset. As part of the join definition, you may select the fields from both datasets to include, filtering out any duplicated or unnecessary fields in the combined dataset.

The way in which the two datasets are joined is defined by the type of join:

- inner join - include only the records in which key (**primary key**) values in the first dataset appear as key (**foreign key**) values in the second dataset.
- left join - include only the records that contain a primary key value that appears in the first (left) dataset.
 - If a primary key value from the first dataset does not appear as a foreign key in the second dataset, any columns brought in from the second dataset contain missing values.
 - Foreign key values that appear in the second dataset and not the first one do not generate rows in the output dataset.
- right join - include only the records that contain a foreign key value that appears in the second (right) dataset. The other conditions above apply in reverse.
- outer join - include all records from both datasets. If a key value is missing from either dataset, the column values included from that dataset are missing.
- For more information, see *Join Types*.

When to use:

Tip: Generally, you should perform join operations as late as possible in your recipes.

- A join is useful for pulling in **selected** fields from a second dataset based on matches of key values. These operations can be expensive to execute but can generate a much wider range of output datasets.

To join your dataset with another, enter `join datasets` in the Search panel. See *Join Data*.

Lookup

A **lookup** operation is used to pull in reference fields from another dataset based on the values contained in a selected column of the first dataset. These second datasets are typically static or changing infrequently.

NOTE: A lookup is similar to a left join. However, with a lookup, all fields from the reference dataset are brought into the generated dataset and all fields from the original dataset are included automatically. When you create a join, you may specify the fields to include in the output dataset.

For example, you might create a dataset like the following:

State-2letters	State-full
AL	Alabama
AK	Alaska
AZ	Arizona
...	...
WI	Wisconsin
WY	Wyoming

If you have a dataset containing the two-letter abbreviations, you can perform a lookup into the above dataset to retrieve the corresponding full names, which are inserted as an adjacent column called `State-full` in the original dataset.

NOTE: If a value in the column from the first dataset does not appear in the second dataset, there is no corresponding value in the generated `State-full` column.

When to use:

- Lookups are useful for referencing shared datasets whose meaning must be consistent across multiple datasets. You can use lookups to pull in customer or product master data (Customer name, address, etc.) based on CustomerId or ProductId values.

To perform a lookup from a column in your dataset, open the column drop-down and select **Lookup....** See *Lookup Wizard*.

Reshape Datasets

Aggregation

A single-dataset operation, an aggregation is used to perform summary calculations on columns in your dataset, optionally grouping your data by the values in one or more columns.

For example, your dataset contains point-of-sale transactions from all of the stores in your organization. You can use an aggregation to summarize total sales by performing a sum operation on your `Total_Sale` column. If you group this calculation by month and by `StoreId`, you can acquire monthly sales per month per store.

When to use:

- An aggregation is useful for performing exploratory calculations on your entire dataset or segments of your dataset.
- You can perform aggregations and run jobs to generate the results. After you have these summary reports, you can return to the Transformer page and remove the aggregation to continue wrangling your data.

For more information on in-column aggregations, see *Create Aggregations*.

Pivot tables

For more information on building aggregated pivot tables, see *Pivot Data*.

Create New Column

Contents:

- *New Formula*
 - *Add a column of text values*
 - *Add a column that uses a function*
 - *Add a column that references another column*
 - *Add a column using constants, functions, and column references*
 - *Merge Columns*
 - *Extract Values from a Column*
 - *Split Column Values*
 - *Convert a Column into Multiple Columns*
 - *Unnest*
-

You can create a new column by adding or editing a formula on any existing column.

New Formula

The New Formula transformation allows you to create a new column based upon a formula that you provide to the transformation. Below are some examples.

Add a column of text values

You can insert a new column containing a string value that you specify as part of the transformation. In the following example, the `status` column is created, and all values in it are set to `ok`.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	'ok'
Parameter: New column name	status

Add a column that uses a function

You can insert a new column by using a function. In the following example, the `currentyear` column is extracted as a new column from the `TransactionDate` column using `YEAR` function.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	YEAR (TransactionDate)
Parameter: New column name	currentyear

For more information on extracting date information, see *Extract Values*.

Add a column that references another column

You can also insert columns containing references to other columns. In the following example, the `totalCost` column is created called `totalCost`, which is based on the formula using three separate columns: `baseCost + totalTax - totalDiscount`:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>baseCost + totalTax - totalDiscount</code>
Parameter: New column name	<code>totalCost</code>

Add a column using constants, functions, and column references

You can insert a column by using nested expressions by using constants, functions, and column references. In the following example, the `Three` column is created, which is based on nested functions `ROUND` and `DIVIDE`.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>ROUND(DIVIDE(10,3),0)</code>
Parameter: New column name	<code>Three</code>

Merge Columns

You can merge two or more columns together to create a new column containing the merged values. For more information, see *Add Two Columns*.

Extract Values from a Column

You can extract values based on patterns or literal values from one column and insert them into a new column. See *Extract Values*.

Split Column Values

You can split the values in a column into separate columns based on delimiters and other conditions that you define. See *Split Column*.

Convert a Column into Multiple Columns

Unnest

You can extract values stored in an array into separate columns in your dataset. This type of transformation can be useful for unpacking nested data such as JSON into tabular format.

- For more information, see *Working with JSON v2*.
- For more information, see *Working with Arrays*.

Add Two Columns

Contents:

- *Check Data Types*
 - *Check Values*
 - *Syntax of Math Functions*
 - *Add One Column into Another*
 - *Add Selective Values from One Column into Another*
 - *Add Two Columns into a New Third Column*
 - *Working with More than Two Columns*
 - *Concatenating Columns*
 - *Summing Rows*
-

This section provides an overview of how to perform mathematical operations between columns.

Check Data Types

Before you begin, you should verify that the data types of the two columns match. Check the icon in the upper left of each column to verify that they match.

To change the data type, you can:

- Click the data type icon.
- Select **Edit data type** from the column menu.

Check Values

After setting data types, you should address any missing or mismatched values in the column. For example, if you change a column's data type from Decimal to Integer, values that contain decimal points may be reported as mismatched values. Use the `ROUND` function to round them to the nearest integer.

Transformation Name	Edit column with formula
Parameter: Columns	myColumn
Parameter: Formula	<code>ROUND(myColumn)</code>

Tip: You can use the `FLOOR` or `CEILING` functions to force rounding down or up to the nearest integer.

Syntax of Math Functions

You can express mathematical operations using numeric operators or function references. The following two examples perform the same operation, creating a third column that sums the first two.

Numeric Operators:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>(colA + colB + colC)</code>

Parameter: New column name	'colD'
----------------------------	--------

Math Functions:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	ADD(colA,colB)
Parameter: New column name	'colD'

NOTE: Expressions containing numeric operators can contain more than two column references or values, as well as nested expressions. Math functions support two references only.

Add One Column into Another

To perform math operations, you can use the Edit column with formula transformation to update values in a column based on a math operation. The following transformation multiplies the column by 10 and adds the value of colB:

Transformation Name	Edit column with formula
Parameter: Columns	colA
Parameter: Formula	((colA * 10) + colB)

All values in colA are modified based on this operation.

Add Selective Values from One Column into Another

You can use the Edit column with formula transformation to perform math operations based on a condition you define. In the following step, the Cost column is replaced reduced by 10% if the Qty column is more than 100. The expression is rounded down to the nearest integer, so that the type of the column (Integer) is not changed:

Transformation Name	Edit column with formula
Parameter: Columns	Cost
Parameter: Formula	IF(Qty > 100, ROUND(Cost * 0.9), Cost)

For rows in which Qty is less than 100, the value of Cost is written back to the column (no change).

Add Two Columns into a New Third Column

To create a new column in which a math operation is performed on two other columns, use the New Formula transformation. The following multiplies Qty and UnitPrice to yield Cost:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	MULTIPLY(Qty,UnitPrice)

Parameter: New column name	'Cost '
----------------------------	---------

Working with More than Two Columns

If you need to work with more than two columns, numeric operators allow you to reference any number of columns and static values in a single expression.

However, you should be careful to avoid making expressions that are too complex, as they can be difficult to parse and debug.

Tip: When performing complex mathematic operations, you may want to create a new column to contain the innermost computations of your expression. Then, you can reference this column in the subsequent step, which generates the full expression. In this manner, you can build complex equations in a way that is easier to understand for other users of the recipe. The final step is to delete the generated column.

Concatenating Columns

If you are concatenating string-based content between multiple columns, use the Merge Columns transformation. In the following example, the Merge Columns transformation is used to bring together the order ID (`ordId`) and product ID (`prodId`) columns, with the dash character used as the delimiter between the two column values:

Transformation Name	Merge columns
Parameter: Columns	ordId, prodId
Parameter: Separator	' - '
Parameter: New column name	primaryKey

Tip: This method can be used for columns of virtually any type. Change the data type of each column to String and then perform the merge operation.

Array column types can be concatenated with the ARRAYCONCAT function.

Tip: You can also use the MERGE function to accomplish the above actions. The function method is useful if you are performing a separate transformation action on the data involved. For example, you could use the function if you are using the Edit formula column to modify a column in place.

Summing Rows

You can use aggregate functions to perform mathematic operations on sets of rows. Aggregated rows are collapsed and grouped based on the functions that you apply to them.

Generate Primary Keys

Contents:

- *The unique row identifier method*
 - *Standardize formatting*
 - *Combine across datasets*
- *The combined field method*

This section describes how you can create primary keys for each row in your dataset.

In database terms, a **primary key** is a column or set of columns that uniquely identifies rows in a table. Examples:

- For log data or other transactional data, the timestamp is typically a unique identifier.

Tip: If you think you need a primary identifier for your dataset, you should try to identify it or create it before you delete potentially useful columns.

- Product information typically contains an SKU identifier. If that is not available, you may need brand, make, and model combinations, which can be created using the method described below.

A well-organized source of data is likely to contain this information for you, but in some cases, you may be required to generate your own primary key.

Tip: In the Transformer page, a quick way to check if there is a primary key in your dataset is to compare the count of categories in the data histograms for string-based data against the count of rows. If the numbers are equal, then the column is suitable for use as a primary key. However, if you ever join with another dataset, you must re-review the suitability of the field and may need to build a new primary key field. Keep in mind that counts apply to the displayed sample, instead of the entire dataset.

This section provides two methods for generating primary keys in your datasets.

The unique row identifier method

When a dataset is loaded into the Transformer page for the first time, you can see a set of black dots along the left side. Hover over these dots to reveal the row numbers retrieved from the original source, if that information is still available. This method relies on these numbers for generating primary keys and is suitable when your final output contains a relatively few number of combined datasets.

NOTE: Some transforms and datasources, like relational sources, make original row order information unavailable.

When you first load your dataset into the Transformer page, you should generate a column containing the original row information, such as the following:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	SOURCEROWNUMBER()
Parameter: New column name	origRowId

This transform is useful to include after initial inference and structuring of each recipe for all of your datasets.

Standardize formatting

The output of this column is a list of numeric values from 1 or 2 to the final row of your dataset. As a unique identifier, you might want to standardize these values. For example, you are transforming a set of orders. You may want to prepend your unique row identifiers with a code and to format them based on a fixed length, as in the following:

origRowId	keyPrefix	primaryKey
1	ORD000	ORD0001
2	ORD000	ORD0002
...	ORD000	...
10	ORD00	ORD0010
...	ORD00	...
99	ORD00	ORD0099
100	ORD0	ORD0100

This structuring generates primary keys of consistent length. You can use the following steps to standardize their formatting, assuming that you have already created the `origRowId` column.

Steps:

1. Change this column to be of String type. Select **String** from the data type drop-down for the column.
2. Create a column containing your prepended identifier and the proper number of zeroes. The following bit of logic generates a string with the proper number of zeroes depending on the length of the value in `origRowId`:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>IF(LEN(origRowId) > 3, 'ORD', IF(LEN(origRowId) > 2, 'ORD0', IF(LEN(origRowId) > 1, 'ORD00', 'ORD000')))</code>
Parameter: New column name	keyPrefix

NOTE: The following works for up to 10,000 rows in the original dataset. You need to add additional `IF` clauses when your row counts exceed 10,000.

3. Now, you can merge these columns together:

Transformation Name	Merge columns
Parameter: Columns	keyPrefix, origRowId
Parameter: New column name	primaryKey

4. You can now delete the prefix column:

Transformation Name	Delete columns
Parameter: Columns	keyPrefix
Parameter: Action	Delete selected columns

These steps should be applied across all datasets that you intend to combine into your output dataset.

Combine across datasets

After you have combined or enriched your dataset, you can combine these original row ID fields from each dataset to create a super primary key in the combined dataset using the method described below.

The combined field method

If your final dataset contains more than a few combined datasets, this basic method for creating a primary key is to find a combination of fields that collectively represent a unique identifier from the final dataset. Columns:

- LastName
- FirstName
- TestNumber
- TestScore

Since there are multiple instances of test data for each person, there is no single column to use as a primary key.

Steps:

1. Load the dataset into the Transformer page.
2. Identify the columns that together can uniquely identifier a row. In the TestScores-All example, these columns are the following:
 - a. LastName
 - b. FirstName
 - c. TestNumber

NOTE: It may be possible to set up a key using LastName and TestNumber, but that is not guaranteed. If the dataset changes over time, a working key based on these columns may become broken.

3. Use the `merge` transform to combine these columns together into a new column, such as the following:

Transformation Name	Merge columns
Parameter: Columns	LastName,FirstName,TestNum
Parameter: Separator	' - '
Parameter: New column name	TestID

The `with` clause identifies the delimiter between the merged column values.

4. Values should look like the following:

TestID
Smith-Joe-2
Doe-Jane-4

5. In some cases, you may want to delete the source columns for the primary key.

Add Lookup Data

Contents:

- *Set up Your Lookup Data*
- *Perform the Lookup*
- *Example - Lookup for Timezones*

You can integrate data from other sources into your current dataset. Based on a key column that you identify in the lookup dataset, you can insert the corresponding values in other columns of the lookup dataset as new columns in your source dataset.

Tip: Column lookups are useful for adding reference data based on a column's values.

For example, your data contains the two-letter abbreviations for U.S. states, yet the target system is expecting the full name of each state. You need to replace the **XY** state abbreviation with the full name of each state in each row.

Set up Your Lookup Data

Your data table should like the following:

State-2Letter	State
AL	Alabama
AK	Alaska
AZ	Arizona
AR	Arkansas
CA	California
CO	Colorado
CT	Connecticut
DE	Delaware
DC	District of Columbia
FL	Florida
GA	Georgia
HI	Hawaii
ID	Idaho
IL	Illinois
IN	Indiana
IA	Iowa
KS	Kansas
KY	Kentucky
LA	Louisiana
ME	Maine

MD	Maryland
MA	Massachusetts
MI	Michigan
MN	Minnesota
MS	Mississippi
MO	Missouri
MT	Montana
NE	Nebraska
NV	Nevada
NH	New Hampshire
NJ	New Jersey
NM	New Mexico
NY	New York
NC	North Carolina
ND	North Dakota
OH	Ohio
OK	Oklahoma
OR	Oregon
PA	Pennsylvania
RI	Rhode Island
SC	South Carolina
SD	South Dakota
TN	Tennessee
TX	Texas
UT	Utah
VT	Vermont
VA	Virginia
WA	Washington
WV	West Virginia
WI	Wisconsin
WY	Wyoming

Tip: You can download a version of this table, which also includes some timezone information. See *Dict-TimezoneByState.csv*.

This data table must be uploaded as a new dataset.

Perform the Lookup

Steps:

1. In the Transformer page, click the drop-down on the column that contains your two-letter state abbreviations. Select **Lookup**
2. In the Lookup Wizard, select the dataset to use for your lookup.
3. For the lookup key, select the column in the dataset to use as the key value. In the above example, it is `State_2Letter`.
4. Click **Execute Lookup**.
5. The lookup key value is used to locate all of the other column values in the reference dataset. These values are inserted in separate columns to the immediate right of the source column.
6. You might need to delete some of the imported columns. In the above case, you might decide to delete the two-letter state identifier column, which has been replaced by the full state name column.

See *Lookup Wizard*.

Example - Lookup for Timezones

The CSV linked above also contains timezone information for each state, which you can use to provide higher fidelity information on timestamps.

U.S. timezones are not consistently demarcated by state lines. Some states are split across multiple timezones. For more accurate representation of timezones, you should download and use a zipcode database, many of which are freely available online. This CSV is provided for demonstration purposes only.

In this case, you are working with a dataset that contains timestamps, which are stored in different timezones based on the location where an event or transaction occurred. However, the timestamps do not contain any timezone information.

You can use an external source of timezone information to insert timezones into your dataset. In the following example, timezones are derived based on two-letter abbreviations for U.S. state. A more accurate representation would be based on zipcode data.

Steps:

1. Complete steps 1-5 in the previous section.
2. Delete all columns except the one containing timezone information. The `Time Offsets` column identifies the predominant timezone in each state as an offset of the UTC timezone (Greenwich Mean Time).
3. Move this column to the right of the column containing your timestamps.

NOTE: Depending on the requirements of your target system, you can use the Split transformation to break up column data so that only the numerical offset (e.g. `-6:00`) is present. Then, you can use the `DATETIMEOFFSET` function to apply the timezone offset to your timestamps. In this manner, you can convert timestamps to the source timezone before they are consumed by the target system.

Append Datasets

If you are wrangling datasets that represent transactional or serialized data, you can append together slices of data to build a larger dataset for richer analysis.

For example, you are cleansing log messages on a weekly basis. You can create separate datasets for each day's log messages and then bring them altogether into a single dataset for processing through a single recipe. This method works best for datasets that have identical or very similar structures.

Below, you can see two datasets of contact information. These simplified datasets track customer contact records.

Dataset01:

Name	Email	Last Contact
Jack Jones	jack@example.com	06/15/2015
Tina Toms	tinat@example.com	08/02/2015
Larry Lyons	larry.lyons@example.com	03/22/2015

Dataset02:

Name	Last Contact Date	Email
Amy Abrams	07/24/2015	amy.abrams@example.com
Tina Toms	05/12/2015	tinat@example.com
Samantha Smith	04/22/2015	samantha@example.com

Notes:

- There is one overlapping record for Tina Toms.
- There is a mismatch in one column name ("Last Contact" vs. "Last Contact Date").
- The columns are in a different order.

Steps:

1. Load your first dataset (Dataset01).
2. In the recipe panel, add a step. In the Transformation textbox, enter `union`.
3. In the Union page, you bring together two or more datasets based on a shared set of fields.
 - a. A **union** operation appends datasets together.
4. To add another dataset, click **Add datasets**. Navigate to select the file to add to the union (Dataset02).
5. Initially, fields are mapped based on the column names. However, in this example, the `Last_Contact_Date` field from Dataset02 is not included. You can:
 - a. Click the + icon next to the `Last_Contact_Date` field in the left panel. The field is added as a separate field. However, it is not matched with the other contact date field from the original dataset.
 - b. From the Match columns drop-down menu, select **By Position**. In this case, you can see that there are only three fields, but the order is mismatched.

Tip: When possible, you should try to rename or align columns in your datasets prior to building a Union transformation step. Otherwise, you might have to edit the columns after the union has been completed.

To rename a column, click **Rename** from the column drop-down in the Transformer page. You can use the same drop-down to move a column.

6. In this case, you can cancel the union and reposition the `Email` column after the `Last Contact` column in `Dataset01`.
7. Then, open the Union page again and add `Dataset02`. Select **By Position** from the Match columns drop-down menu. Your columns are matched.
8. Click **Add to Recipe**.

`Dataset02` records have now been added to `Dataset01`, which now contains all of the records from both datasets. Note that the record for Tina Toms appears twice in the appended dataset.

- If the appended dataset is a record of all contacts, you should leave the duplicate record in place.
- If the appended dataset is a record of the most recent contact with each customer, you should remove the duplicate record with the Deduplicate transformation. For more information, see *Deduplicate Data*.

NOTE: Be sure to verify that the data type for each column is accurate.

Join Data

Contents:

- Overview
 - Create Join
 - Step - Choose dataset or recipe to join
 - Step - Choose join keys and conditions
 - Step - Specify output columns for the join
 - Step - Review join
 - Modify Keys and Conditions
 - Ignore special characters
 - Create fuzzy join
 - Create range join
 - Add multiple join keys
-

You can join together data based on the presence of one or more keys in your source dataset and the joined-in dataset or recipe. A **join** is a data operation in which two or more tables or datasets are merged into one based on the presence of matching values in one or more key columns that you specify. These shared columns are called the **join keys** of the two sets of rows that you are attempting to join.

Overview

Using a Join transformation, you can join a recipe or dataset to any of the following objects:

- Another recipe
- An imported dataset
- A reference dataset

Create Join

You can join datasets through the following mechanisms:

- **Flow View:** Select a dataset or recipe object. Right-click and select **Append Join**.

Tip: The join you specify from Flow View is added as the last step to the recipe. If you selected a dataset to which to add the join, a recipe is created from the object, and the join is added as the first step of the new recipe.

- **Transform Builder:** Search for and select `Join`.

Joins are created through the Join window. This workflow is described below.

Step - Choose dataset or recipe to join

Steps:

1. In the Choose dataset or recipe panel:
 - a. Search for a dataset or recipe to which you have access. Your search includes objects outside of the current flow.
 - b. You can also select from:
 - i. Recipes in your current flow
 - ii. Datasets in your current flow

- iii. All datasets to which you have access.
2. When you have found the dataset to use in your join, click **Accept**.

Step - Choose join keys and conditions

Steps:

1. Next, you select the join key columns and other conditions from each dataset.
2. **Join type:** Select the type of join to apply. See "Join types" below.
3. **Join keys:** The application attempts to find the best columns to match as the join keys.
 - a. Mouse over the percentage match to get more detailed statistics.

NOTE: For formatted data types, such as Datetime, the formatting of the join keys must also match. For example, the values 2021-01-01 and January 01, 2021 may not be interpreted as matching values.

- b. To change a join key, mouse over the key name and then click the Pencil icon. Select your new key.
 - c. For more information on the options, see "Modify Keys and Conditions" below.
 - d. Click **Save & Continue**.
4. Click **Next**.

Example datasets

For discussion purposes, the following datasets are referenced in the sections below.

- The `CustId` column is shared between both datasets. This column is the **join key**, as there are no matches between the other columns.
- Some values in `CustId` in one dataset do not appear in the other.

Dataset A:

The first dataset to which you are joining in another is typically called the **left dataset**.

CustId	LastName	FirstName
c001	Jones	Jack
c002	Kim	Ken
c003	Lee	Larry
c004	Miller	Mike
c005		

Dataset B:

The second dataset that you are joining in to the first is typically called the **right dataset**.

CustId	Region	CompanyName
c002	East	ACME, Inc.
c003	West	Trifax, Inc.
c005	North	Example Co.
c006	South	Ace Industries

Join types

There are multiple types of joins, which generate very different results. When you perform a join, you specify the type of join that is applied. The joined-together rows that appear in the output dataset are determined by the type of join that you selected and matching of values in the join key columns.

The following are the basic join types. The Example column references Dataset A (left) and Dataset B (right) from above.

Join Type	Description	Example
inner join	If a join key value appears in the left dataset and the right dataset, the joined rows are included in the output dataset.	In the above output, rows c002 and c003 are included only.
left join	In a left join, all of the rows that appear in the left dataset appear in the output, even if there is no matching join key value in the right dataset.	In the above output, rows c001, c002, c003, c004, and c005 are included. Rows c006 is excluded.
right join	In a right join, all of the rows that appear in the right dataset appear in the output, even if there is no matching join key value in the right dataset.	In the above output, rows c002, c003, c005, and c006 are included. Rows c001 and c004 are excluded.
outer join	An outer join combines the effects of a left and a right join. Each key value from both datasets is included in the output. If the key value is not present in one of the datasets, then null values are written into the columns from that dataset.	In the above output, rows c001, c002, c003, c004, c005, and c006 are included. Rows c001, c004, c005, and c006 contain some null values.
cross join	A cross join matches every row in the source dataset with a row in the joined-in dataset, regardless of whether the join keys match. <div>NOTE: A cross join can greatly expand the number of rows in your dataset, which may impact performance.</div>	If Dataset A has 5 rows and Dataset B has 4 rows, the output has 20 rows.
self join	A self join matches the rows in the left dataset with a version of itself (dataset or recipe) on the right side. Some limitations apply.	

Step - Specify output columns for the join

Steps:

1. In the Output columns step, you can specify the columns to include in the output dataset.
 - a. Include All: To include all columns from the left and right datasets, click the checkbox below All.
 - b. Use the Search box to search for specific columns to include or exclude.
2. Advanced options: See below.
3. Click **Review**.

Apply prefix for column names

In the output dataset, the column names are taken directly from the column names in the source dataset. Potential issues:

- In some cases, source column names may be an exact match between datasets.
- For development purposes, you may wish to track the source of a column for a period of time.

You can apply a prefix to the column names that are sourced from the left dataset, the right dataset, or both.

- **Name prefix for columns in Current data:** Enter a text value to include as the prefix to any output columns that are sourced from the current (left) dataset. For example, you could enter `left_`.
- **Name prefix for columns in Joined-In data:** Enter a text value to include as the prefix to any output columns that are sourced from the joined-in (right) dataset. For example, you could enter `right_`.

Apply dynamic updates of selected columns

In the recipe step that produces the join, the columns that you select are mentioned specifically by name. Optionally, you can choose to automatically add in all columns to your output. For example, if your source data for an imported dataset is augmented with 10 new columns, when you re-run your join, those new columns can be automatically added to the output dataset.

Tip: You should consider using these options if the schema of your data sources is likely to change in the future.

- **Include all columns from Current data:** When selected, all columns that are subsequently added to the Current (left) dataset are automatically included as part of the join.
- **Include all columns from Joined-In data:** When selected, all columns that are subsequently added to the Joined-In (right) dataset are automatically included as part of the join.

Step - Review join

Steps:

1. In the Review step, you can verify that the specified join is as you expected.
2. You should review the columns that are previewed as in the data grid.
3. To add the join as a recipe step, click **Add to Recipe**.

Modify Keys and Conditions

NOTE: If you modify the selected dataset to join, the joined dataset, the join keys, or the fields to include in the output, subsequent steps in your transform recipe can be broken by the change. After you modify the join, you should select the last step in your recipe to validate all steps in the recipe.

You can apply the following modifications to how keys are matched. To modify a join key and condition, click the Pencil icon in the Join Keys & Conditions panel.

Ignore special characters

Optionally, you can configure the Trifacta application to ignore the following special characters, when matching values in join keys:

- **Ignore case:** Ignore differences in case between values in the join key columns. `MyValue` matches with `MYVALUE`.
- **Ignore special characters:** Ignore special characters that appear in the join key values.
- **Ignore whitespace:** Ignore spaces, tabs, and other whitespace values that may appear in join key values.

Create fuzzy join

A **fuzzy join** applies a fuzzy matching algorithm to String values in the join key column to account for slight differences in how values are written.

NOTE: Fuzzy joins can only be applied to String data types. Other data types cannot be fuzzy-matched using the algorithm.

This algorithm relies on the doublemetaphone function, which attempts to normalize text values based on how the string is spoken by an English speaker. For more information, see https://en.wikipedia.org/wiki/Metaphone#Double_Metaphone.

- **Fuzzy match:** Enable fuzzy matching based on English language pronunciation using the doublemetaphone function.

Create range join

NOTE: This feature may need to be enabled in your environment. See *Workspace Settings Page*.

Values in the join key columns are matched across a range of values, instead of matching single value to single value. When range joins are enabled, you can set the Condition value between the two join key columns when specifying the join keys. For more information, see *Configure Range Join*.

Add multiple join keys

For more complex join operations, you can add additional join keys to evaluate. Multi-key joins can be helpful for:

- Providing more finely specified join keys. For example, `lastName` and `firstName`.
- Performance

To add a second join key, click **Add** when modifying the join keys and conditions. Specify the keys in each dataset as needed.

Configure Range Join

In most join operations, the values in primary keys across two tables must match exactly for the related columns to be included in the join. In a **range join**, you can change the comparative operator for the keys from Equal to one that specifies a range of matching values.

Comparative operators:

- Not equal to
- Greater than
- Greater than or equal to
- Less than
- Less than or equal to

NOTE: A Trifacta administrator may need to enable this feature in your environment. See *Workspace Settings Page*.

Limitations:

Range joins allow you to include many more matching values and therefore rows in the join. Depending on the matches and the included columns, your resulting dataset can become very large. You should use this feature with some caution.

- Range joins apply only keys whose data types can be compared.
 - For example, for joins involving keys of Binary data type, you can use Equal to or Not equal to joins.

Tip: Range joins cannot be applied to Datetime data type values directly. However, you can use the UNIXTIME function to convert the values to numeric Unix time values. Then, you can specify a range join.

- Any range comparison that includes one or more string columns as keys uses the string comparison greater/less than, not the numerical comparison.

After range joins have been enabled, you can specify them as part of performing any join operation.

Steps:

1. In the Search panel, enter `join datasets` in the search box.
2. Select the dataset with which to join the current one. Then, click **Accept**.
3. In the Join window, select the join type.
4. In the Join Keys area, click the Pencil icon.
5. Specify the fields in the current dataset and the joined-in dataset.

6. From the Condition drop-down, select the range operator to use:

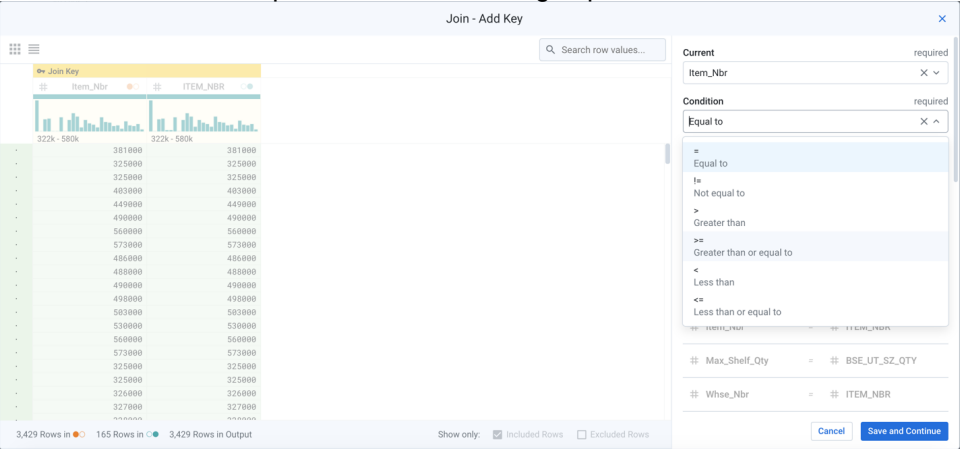


Figure: Select range operator

- 7. Specify other properties for the matching keys.
- 8. Click **Save and Continue**.
- 9. Specify other elements of the join. When finished, click **Add to Recipe**.

Insert Metadata

Contents:

- *Insert filepath*
 - *Insert source row number*
 - *Insert a single metadata column*
 - *Insert multiple columns of metadata*
-

Metadata is data about your data. In some use cases, you may need to insert data about your data into your dataset for downstream consumption.

For example, you might decide that one or more of the following types of information about your dataset should be tracked:

- Source system(s)
- Source filepath and filename (for non-uploaded files)
- Source creation date
- Date of import
- Date of wrangling
- Name of person who performed the wrangling

This section provides some methods for how to insert metadata into your dataset.

Insert filepath

For file-based data sources that were loaded from a dedicated storage layer, you can insert the path to the source file in your dataset using the `$filepath` reference.

Tip: Filepath information can be lost when multi-dataset operations, such as unions and joins, are performed on your dataset. These steps should be added very early in your recipe.

In your recipe, insert the following transformation:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>\$filepath</code>
Parameter: New column name	<code>sourceDatasetPath</code>

Insert source row number

You can insert the row number in the source file from which rows in your dataset are sourced, using the `$source_rownumber` reference.

Tip: Source row number information can be lost when multi-dataset operations, such as unions and joins, are performed on your dataset. These steps should be added very early in your recipe.

In your recipe, insert the following transformation:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	\$sourcerownumber
Parameter: New column name	sourceRowNumber

Tip: Use the ROWNUMBER function to derive the current row number in your dataset.

Insert a single metadata column

The following example describes how to insert a single column of metadata. In this case, the full path to the source is inserted as a new column in the dataset.

Steps:

1. In the Dataset page, locate the imported dataset that is the source for your recipe. Click the Imported filter to show only the imported datasets.
2. For the imported dataset, click **Details**.
3. In the Dataset Details page, select the entire value for the Location, which is the storage location of the source.

Tip: If the full path of the dataset is too long for screen display, be sure to include the ellipsis (...) at the end of the Location value.

4. Copy the value. Paste the value into a text editor. You should see the full path, like the following:

```
<root_dir>/uploads/1/2580298d-3477-4907-bfa7-f71978eace04/SF Restaurants - businesses.csv
```

5. Load the dataset in the Transformer page.
6. Specify the following transformation:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	'<root_dir>\uploads\1\2580298d-3477-4907-bfa7-f71978eace04\SF Restaurants - businesses.csv'
Parameter: New column name	datasetPath

Insert multiple columns of metadata

You might need to track more fields of dataset information. While you might be able to perform these kinds of individual inserts, it might be easier to build this information from a separate file.

NOTE: This method uses the FILL function, which should be limited to smaller datasets when applied with a single key. Otherwise, there might be performance impacts when running the job against the full dataset.

Tip: You can perform a similar merging of datasets using the `Join datasets` transformation. See *Join Data*.

For example, you want to track the following fields as metadata:

- `source_system`
- `source_author`
- `source_date_create`

You could create a CSV file that looks like the following:

```
source_system,source_author,source_date_create
Excel,Joe Guy,12/9/15
```

In this case, the column headers are in the first line, and the values for each column are in the second line.

Steps:

1. Use your CSV file as the source for a new dataset within the flow containing the associated dataset.
2. In the data grid, make sure that the first line of data is treated as the header. If not, add a `header` transform to your recipe.
3. Open the other (source) dataset in the Transformer page.
4. In the recipe panel of the Transformer page, add a new step. In the Transformation textbox, enter `union`.
5. Create a union using the Union transformation:
 - a. Include all columns from both datasets.
 - b. Configure the step to perform the union by name, instead of by position.
6. Add this step to your recipe.
7. You should see one row in the union recipe that contains the new data.
8. Sort your data by a key value (e.g. `business_id`).
9. Determine an appropriate grouping parameter. This step is necessary to simplify the filling process when the job runs at scale. Ideally, you should choose a grouping column that contains a relative few number of values in it (e.g. `region`).
10. Fill values in the data rows with metadata column values. For each metadata column, add the following transformation, done here for the `source_system` column of metadata.

Transformation Name	Window
Parameter: Formula	<code>FILL(source_system)</code>
Parameter: Group by	<code>region</code>
Parameter: Order by	<code>business_id</code>

11. Repeat the above step for each metadata column you want to insert.
12. Delete the source metadata columns.
13. Rename the `window` columns to use a more appropriate name.
14. Delete the row containing the original metadata values.

Invoke External Function

Contents:

- *Prerequisites*
- *Invoke*
- *Examples*
 - *ConcatUDF*
 - *AdderUDF*

Through the Search panel, you can access and apply functions that have been developed external to Trifacta®.

NOTE: This method of invocation applies only to Java UDFs created and applied to a specific deployment of Trifacta Self-Managed Enterprise Edition.

Prerequisites

Also known as user-defined functions, external functions must be developed in an environment external to the product and then registered for use in it. These steps require developer skills. For more information, see *User-Defined Functions*.

Invoke

After an external function has been registered with the product, you can complete the following steps to invoke the function within your recipe.

Steps:

1. In the Transform Builder, you can search for any of the following:
 - a. `udf`
 - b. `invoke external function`

NOTE: You cannot search for the name of the external function.

2. Select **Invoke external function**.
3. The list of available external functions is displayed. Select the function to use.
4. Depending on the function, the following options may be available:
 - a. Columns: specify the column or columns to which to apply the function.
 - b. Arguments: If the function accepts arguments, you can enter them on individual lines.
 - c. New column name: Some functions generate a new column. Enter a new column name.
5. To add the instance of the function to your recipe, click **Add**.
6. The step is added to your recipe.

Examples

You can create these examples functions in Java for use in the platform. For more information, see *Java UDFs*.

ConcatUDF

The ConcatUDF function concatenates two strings together.

Tip: This function is provided for demonstration purposes only. In practice, you should use the MERGE function instead.

Transformation Name	Invoke external function
Parameter: Column	colA, colB
Parameter: Arguments	(empty)
Parameter: New column name	myConcatUDFColumn

AdderUDF

The AdderUDF function adds an input value to a constant that is submitted by parameter. The following invocation of AdderUDF adds colA and the constant 100.

Tip: This function is provided for demonstration purposes only. In practice, you should use the ADD function instead.

Transformation Name	Invoke external function
Parameter: Column	colA
Parameter: Arguments	100
Parameter: New column name	myAdderUDFColumn

Publishing Tasks

These tasks provide information on the various methods of getting your data out of Trifacta®. These tasks include imported datasets, recipes, generated results, and work-in-progress versions of them.

Create Outputs

Contents:

- *Create an Output*
 - *Create a File-Based Output*
 - *Create a Table-Based Output*
 - *Create an Output With Parameters*
 - *Parameterize path or bucket name with a variable*
 - *Parameterize path with a timestamp*
 - *Edit an Output*
 - *Delete an Output*
-

An output is defined as a set of files or tables, formats, and locations where results are written after a job run on the recipe has completed. To run a job from a flow, you must create an output object that defines where results are delivered after a job is successfully executed.

Every flow requires an output in order to publish results. An output object is composed of one or more publishing actions. A **publishing action** defines the output type, format, location, and other settings where results from a recipe are delivered.

You can create publishing actions in multiple formats and publish those to different databases and file storage formats. The following are the output types:

- File-based outputs such as CSV.
- Table-based outputs such as Oracle or PostgreSQL.

Create an Output

You can use either of the following methods to create an output object and its related publishing action.

From Flow View:

In Flow View, an output object extends from a recipe, indicating the results of the recipe are delivered to the output object.

1. Open your flow in Flow View.
2. In Flow View, you can:
 - a. Right-click a recipe. Select **Add Output to run**.
 - b. If an output already exists, select it.
3. The output is displayed in the Details panel on the right-side.
4. In the Details column under Manual Settings, click **Edit**.
5. In the Publishing Settings page, click **Add Publishing Action**.

Tip: For scheduled runs of your flow, you must specify Scheduled Settings to automatically generate the output when the flow is executed by a schedule. For more information on scheduling, see *Overview of Automator*.

From Run Job page:

For an existing output, you can create new destinations from the Run Job page.

1. In Flow View, click **Run Job**.
2. In the Run Job page, click **Add Action** to add a new destination.

Create a File-Based Output

You can create a file-based output by performing the following steps.

For more information on creating an output from Flow View and Run Job page, see above sections.

Steps:

1. In the Publishing action page, select the connection where you wish to write file from the left panel. In the following example, the HDFS connection has been selected:

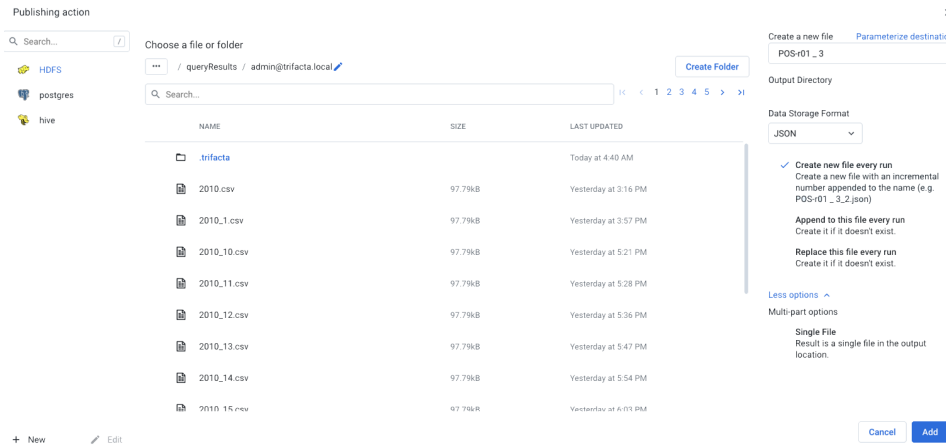


Figure: Publishing action page for file output

2. Select the file. You can select the existing file from the search list or click a **Create a new file** in the right panel.
 - a. Enter a file name in the **Create a new file** field.
3. To create output parameters, click the Parameterize destination link. See "Create an Output with Parameters" below.
4. From the **Data Storage Format** drop-down list, select the output format for the file.
5. The publishing actions vary based on the options selected. Select the required publishing actions below the drop-down list. For more information, see *File Settings*.
6. Update the **Delimiter** field, if required.
7. You can choose to generate the file as a Single File or as Multiple Files.
8. To apply compression to the file, select the compression type from the **Compression** drop-down list.
9. Click **Add**.

Tip: You can define SQL scripts that are executed before or after generation of your output objects. For more information, see *Create Output SQL Scripts*.

Create a Table-Based Output

You can create output objects for publishing to tables by performing the following steps:

For more information on creating an output from Flow View and Run Job page, see above sections.

Steps:

1. In the Publishing action page, select the connection to the database where you wish to store the table from the left panel. In the following example, the `postgres` connection is selected:

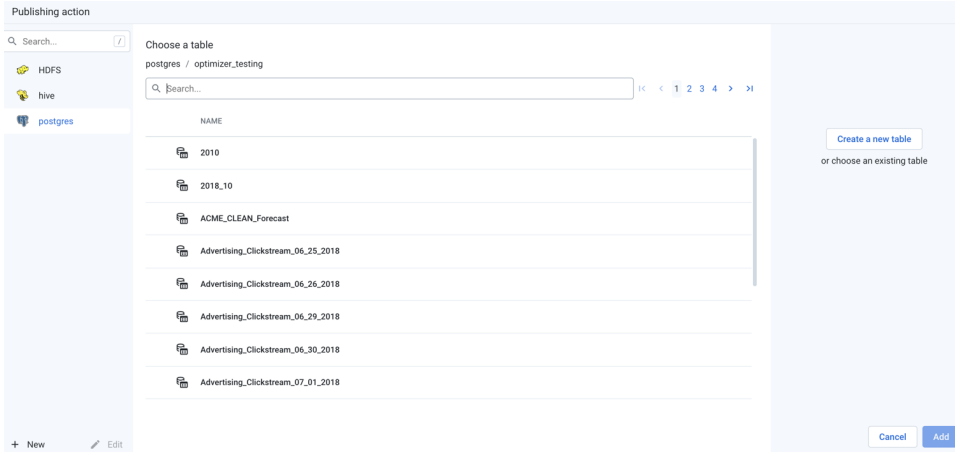


Figure: Publishing action for a table output

2. Search the table. You can select an existing table from the list or click **Create a new table** in the right panel.
 - a. Enter a table name in the **Create a new table** field.
3. To create output parameters, click the Parameterize destination link. See "Create an Output with Parameters" below.
4. Select the required publishing actions below the drop-down list. For more information, see *Relational Table Settings*.
5. Click **Add**.

Tip: You can define SQL scripts that are executed before or after generation of your output objects. For more information, see *Create Output SQL Scripts*.

Create an Output With Parameters

For any outputs, you can parameterize elements of the output path. You can parameterize your path with the following options.

Tip: You can define multiple parameters per output.

- **Timestamps:** Inserts a formatted timestamp as part of the output path or filename
- **Variables:** Inserts a value for the variable.
 - This variable has a default value that you assign.
 - Whenever you execute a job through the Run Job page, you can pass in the default value or an override value for the variable.

For more information on parameters, see *Overview of Parameterization*.

Parameterize path or bucket name with a variable

For file- or table-based publishing actions, you can replace the bucket name (if applicable) or elements of the output path with variable values. When you define the output, you replace an element of the output path with the variable name. At runtime, the variable name is replaced by the appropriate value.

Tip: You can use environment parameters to parameterize bucket names across your environment. For more information, see *Environment Parameters Page*.

1. In the Publishing action page, click the **Parameterize destination** link. The Define Parameterized destination dialog is displayed.
2. On the listed output path, highlight the part that you wish to parameterize. You can select part of the path or bucket name.
3. Then, select **Add Variable**.

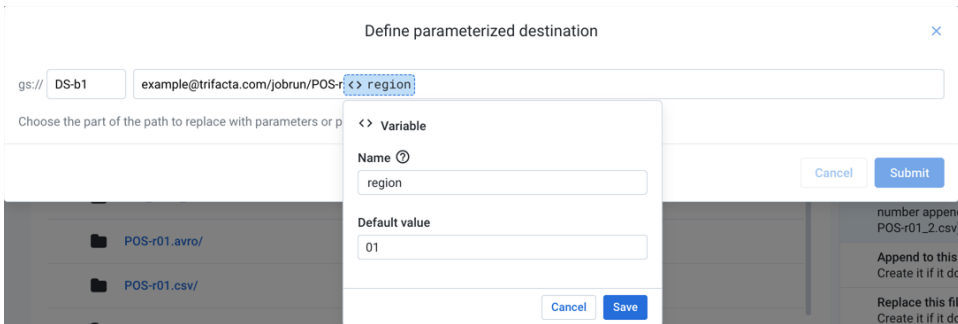


Figure: Define parameterized destination

- a. **Name:** Enter a display name for the variable.

Tip: Type `env.` to see the environment parameters that can be applied. These parameters are available for use by each user in the environment.

NOTE: If multiple variables within a flow (or its dependent flows) have the same name then they are treated as the same variable.

- b. **Default value:** Enter a default value for the parameter.
4. Click **Save**.
 5. To save the parameters for the output path, click **Submit**.

The created parameter is displayed in the right context menu of the publishing action page.

Tip: If you created a variable parameter, you can apply override values to the variable when you are running a job. For example, you can modify a variable called `baseFileName` to generate an output with a different base filename for your job run.

Parameterize path with a timestamp

Timestamp parameters can be helpful when you want to create outputs based on date and time format, time zone, or exact and relative start time. For file- or table-based publishing actions, you can create outputs based on the specific region or time zone for which the data is generated. When you define the output, you can replace an element of the output path with the timestamp parameters.

Steps:

1. In the Publishing action page, click the **Parameterize destination** link. The Define Parameterized destination dialog is displayed. See example above.
2. On the listed output path, highlight the part that you wish to parameterize. Then, select **Add Timestamp Parameter**.

3. In the Timestamp Parameter dialog, enter the following details:
 - a. **Timestamp format:** Specify the format for timestamp values.
 - i. Example: YYYY-MM-DD_hh_mm.
 - ii. Values can express both date and time elements. For more information on the available tokens for formatting date and time values, see *Datetime Data Type*.
 - b. **Timestamp value:** Select the value to record in the path:
 - i. **Exact job start date:** recorded timestamp in path is the start time of the job.
 - ii. **Relative to the job start date:** recorded timestamp in path is relative to the start time of the job according to the settings that you specify here.
 - c. **Time zone:** Click **Change** to change the time zone recorded in the timestamp.
 - i. Example: America/Los Angeles or Asia/Calcutta.
 - ii. For more information on the available time zones, see *Supported Time Zone Values*.
4. Click **Save**.
5. To save the specified parameter for the output path, click **Submit**.

The created parameter is displayed in the right context menu of the publishing action page.

Edit an Output

From Flow View page:

1. Right-click an output object. The object details are displayed in the context panel.
2. In the context panel, select the Manual Settings tab. Then, click **Edit**. The Publishing Actions page is displayed.
3. Make changes as needed in the Publishing Actions page. To save your changes, click **Update**.

From Run Job page:

In the Run Job page, hover over the publishing action to modify. Click **Edit**.

Delete an Output

You can delete the output object from the Flow View and from Run Jobs page:

Flow View page:

1. In the Flow View, select the output.
2. In the right panel, select **Delete Output** from the context menu.

Run Jobs page:

In the Run Jobs page, you can delete publishing actions. From the context menu for a publishing action, select **Delete**.

Create Output SQL Scripts

Contents:

- *Script Types*
 - *Script execution*
 - *Limitations*
 - *Enable*
 - *Create Output SQL Script*
 - *Parameterize values*
 - *Monitoring execution*
 - *Example Scripts*
 - *Example - log entries*
 - *Example - updates based on job results*
 - *Edit Output SQL Script*
 - *Delete Output SQL Script*
 - *Create Output SQL Script via API*
 - *Create SQL script*
 - *List SQL scripts*
 - *Edit SQL script*
 - *Delete SQL script*
-

As part of job execution for an output, you can define SQL scripts to run before the job, after it, or both. These SQL scripts are stored as part of the output object definition and can be executed through any database connection to which the user has access. SQL scripts can be applied to file-based and table-based job executions.

- When flows are shared, the shared user can modify SQL Scripts if the user has Editor permissions on the flow. See *Overview of Sharing*.

Example uses:

- Insert or update log entries in a database log table before or after a job that publishes to file or database destinations.
- Perform custom inserts, updates, and delete logic to other database tables based on job output data that is published to a database.
- Create and refresh tables or materialized views that join the job's output data with data from other tables using `CREATE AS SELECT`.
- Operational tasks such as disabling/enabling indexes and managing partitions on supported databases.

Script Types

NOTE: If one of these scripted steps fails, then all downstream phases of the job also fail.

- **Pre-job:** After a job has been initiated and before data is ingested into the platform, a SQL script can be executed by the Trifacta application.
- **Post-job:** After job results have been generated and published, a SQL script can be executed.

NOTE: If publishing job fails, then all downstream tasks also fail, including the SQL script, which is not executed and is recorded as a failed phase of the job execution.

Script execution

- SQL lines in an individual script are executed in the order listed in the script.
- If you have defined multiple scripts of the same type (pre-job, for example), those scripts may be executed in parallel.

NOTE: The order of listing of scripts in the Trifacta application does not affect the order of execution of those scripts.

- A warning message is displayed if the pre/post SQL scripts do not have any valid connection.

Limitations

These SQL scripts are executed without validation through the selected database connection. There are no explicit limitations on the types of SQL statements that can be executed. It is possible to do harm through this feature.

- After each SQL statement in a script, a semi-colon is required.
- SQL validation is not supported for some connection types.
- When flows containing output SQL scripts are imported, the connection to the database where the script is to be executed must exist in the new environment. Otherwise, the SQL script is dropped from the import.
- Output SQL script actions may not be supported for all connection types. If the connection is not available in the dropdown for selecting a connection, then the feature may not be available for the connection type.
- Output SQL script actions are only supported for default connection types provided with the product. Custom connection types modified for a specific customer environment are not supported.

Enable

This feature may need to be enabled in your environment.

A workspace administrator can enable the use of SQL scripts. For more information, see *Workspace Settings Page*.

Create Output SQL Script

Through the Trifacta application, you add SQL scripts as part of the output object definition.

Tip: Depending on the nature of your SQL script, you may choose to test it first in a demo environment on a demo database.

Where to add:

You can create SQL scripts for the following types of outputs:

- **Manual Settings destinations:** In Flow View, you can select an output object and then modify one of its Manual Settings destinations.
- **Scheduled Settings destinations:** In Flow View, select an output object and then modify one of its Scheduled Settings destinations.

Steps:

1. In Flow View, select the output object for which you wish to create the SQL script.
2. In the Outputs panel on the right, click the Manual Settings tab.

3. For the type of destination, click **Edit**.
4. In the SQL Scripts panel at the bottom of the screen, click **Add Script**.
5. In the Add SQL Script window:
 - a. Select the database connection to use for executing the SQL script.
 - b. Enter the SQL script in the panel.
 - c. Choose when you would like to execute the script:
 - i. Run before data ingest - before the job is executed
 - ii. Run after data publish - after the job results have been written
 - d. Before you save your changes, click **Validate SQL**.

NOTE: Some connection types do not support SQL validation.

NOTE: Validating the SQL does not execute the SQL script on the database. It performs a check of SQL syntax against the selected database.

6. To save your SQL script, click **Add**.

For more information, see *SQL Scripts Panel*.

Parameterize values

You can add variable or Datetime parameters to your SQL scripts.

- Parameters with the same name that are also defined on input datasets, flow parameters, and output objects can be referenced during job execution to pass the same value for consistency.

Tip: You can parameterize values in your SQL script. Parameters can be variables, Datetime parameters, or environment parameters. For more information, see *Overview of Parameterization*.

Monitoring execution

You can monitor the execution of any SQL scripts that are part of a job execution. For more information, see *Overview of Job Monitoring*.

Example Scripts

In the following sections, you can review some common examples for how to use SQL scripts in your data pipelines.

Example - log entries

In this example, you insert log entries into a log table in your database before and after the execution of your job.

Pre-job:

Your SQL script might look like the following:

```
CREATE TABLE IF NOT EXISTS "transactions"."log-tri" (  
  timestamp date,  
  jobType varchar(255),  
  jobStatus varchar(255)  
);  
INSERT INTO "transactions"."log-tri"(timestamp, jobType, jobStatus)  
VALUES ('2021-06-22', 'transformation', 'started');
```

The above script is composed of two statements:

1. **CREATE TABLE IF NOT EXISTS** - This statement creates the `log-tri` table in the `transactions` database.
 - a. This table is defined with three fields: `timestamp`, `jobType`, and `jobStatus`, each of which is assigned a data type.
 - b. The **IF NOT EXISTS** keyword ensures:
 - i. The table is created if it does not exist.
 - ii. If it exists, then no error is returned, which could stop the job run.

INSERT INTO - This statement inserts a record into the `log-tri` table, populating each column with an appropriate **VALUE**:

Column name	Value
timestamp	'2021-06-22'
jobType	'transformation'
jobStatus	'started'

Tip: In the above example, the value for the `timestamp` is a literal value. If needed, you can parameterize that value, so that a Datetime parameter can be inserted into the record as needed. See "Parameterize values" above.

Post-job:

After the job results have been published, a post-job SQL script might look like the following:

```
CREATE TABLE IF NOT EXISTS "transactions"."log-tri" (  
    timestamp date,  
    jobType varchar(255),  
    jobStatus varchar(255)  
);  
INSERT INTO "transactions"."log-tri"(timestamp, jobType, jobStatus)  
VALUES ('2021-06-22','transformation','complete');
```

This script is very similar to the previous:

1. Create the table if it doesn't exist. This statement also provides schema information if you need to make modifications in the future.
2. Inserts a new row in the table, indicating the `transformation` job type is now complete.

Example - updates based on job results

If you write your job results through the same connection where you are executing your SQL script, you can leverage the data directly from your job results into your SQL script.

In the following scenario, a customer account dimension table in the datawarehouse (`dw.DimCustAccount custdim`) is updated with data enriched through Trifacta in the job results. In this case the `num_emp`, `industry_cd`, and `duns` columns are mapped to the corresponding columns in the `custenr` enriched data table with values where the customer identifier in the customer dimension table (`custdim.custId`) matches the customer identifier in the enriched data table (`custenr.custId`).

```
UPDATE TABLE dw.DimCustAccount custdim  
SET num_emp = custenr.empcnt, industry_cd = custenr.ind_cd, duns = custenr.duns_num  
FROM tri.cust_enriched custenr  
WHERE custdim.custId = custenr.custId;
```

Edit Output SQL Script

Steps:

1. In Flow View, select the output object.
2. In the context panel on the right, select the Manual Settings tab.
3. Click **Edit** next to the type of destination to modify.
4. In the dialog, locate the one to modify in the SQL Scripts panel. Click **Edit**.
5. Make changes as need. Click **Save**.

Delete Output SQL Script

After you deleting a SQL script and save the output object, the SQL script is removed permanently. Before deleting, you may wish to copy the script and paste it into a text editor.

Steps:

1. In Flow View, select the output object.
2. In the context panel on the right, select the Manual Settings tab.
3. Click **Edit** next to the type of destination to modify.
4. In the dialog, hover over the one to modify in the SQL Scripts panel. From the More menu, select **Delete**.

Create Output SQL Script via API

You can create SQL scripts via API. These scripts can then be associated with specific output objects.

Create SQL script

Key information:

Attribute	Description
sqlScript	Text of the SQL script. You should validate this script before inserting it into the API.
type	Set type to be: <ul style="list-style-type: none">• pre - execute before data ingest• post - execute after data publish
vendor	The vendor type of the database to which you are connecting. See <i>Connection Types</i> .
outputObjectId	Internal identifier of the output object to which you are associating the SQL script. When the object is selected in Flow View, the identifier is part of the URL.
connectionId	Internal identifier of the connection that you are using to execute the SQL script.

Endpoint	/v4/sqlScripts
Method	POST

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/createSqlScript>

List SQL scripts

List all SQL scripts.

Endpoint	/v4/sqlScripts
Method	GET

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/listSqlScripts>

Edit SQL script

Endpoint	/v4/sqlScripts/{id}
Method	PATCH

where:

- {id} is the internal identifier of the SQL script

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/patchSqlScript>

Delete SQL script

Endpoint	/v4/sqlScripts/{id}
Method	DELETE

where:

- {id} is the internal identifier of the SQL script

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/deleteSqlScript>

Publish Results on Demand

After a set of results have been generated from a job, you can export those results to different environments in different formats as long as the job remains in Trifacta®.

This feature is also known as **ad-hoc publishing**.

Steps:

- In the left menubar, click the Jobs icon.
- In the Jobs page, select the job whose results you wish to publish.

Tip: If you enabled profiling of the results, you can click the job identifier to open the visual profile of the job results.

Tip: If you know the recipe and flow from which the job was executed, you can also select the recipe in Flow View and then select the output object for that recipe. In the right panel, you can click the job identifier in the Jobs tab to open the job results for export.

- In the Job Details page, click the Output Destinations tab.
- Publishing:
 - **Export file:** Click one of the links to the generated output files to download the results in that file format. If you do not see your preferred export format, please rerun the job.
 - **File path:** You can use the provided file path to get the export from the backend datastore outside of the application.
 - **Create dataset:** From the context menu for the output, select **Create imported dataset** to turn the results into a new imported dataset in Trifacta.

Tip: In Flow View, you can create a reference object for any recipe in your flow. This reference object makes the output of the recipe available as an input object in other flows. So, you can use this as a method of creating a new dataset from the output of your recipe that is automatically updated without having to regenerate the imported dataset.

- Close the window when you are done.

Reuse Recipe

Contents:

- *Reuse Recipe with the Same Flow*
 - *Reuse Copy of Recipe in the Same Flow*
 - *Move or Copy Recipe to a Different Flow*
 - *Reuse Recipe in a Different Environment*
 - *Download Recipe*
-

This section describes multiple ways in which you can leverage a recipe developed in one flow in other flows.

Reuse Recipe with the Same Flow

The easiest way to reuse a recipe is to change its inputs in Flow View.

Steps:

1. Open the flow containing the recipe.
2. Select the recipe that you'd like to reuse.
3. From the recipe's context menu, select **Change input....** Select the object to be the input to the recipe.

Reuse Copy of Recipe in the Same Flow

You can also create copies of recipes within the same flow. This step also copies:

- All outputs attached to the recipe.
- (optionally) All inputs to the recipe.

Steps:

1. In Flow View, select the recipe to copy.
2. In the context panel on the right, select **Make a copy > Without inputs**.
3. The recipe is copied and added to your flow.
4. To apply the recipe to a dataset, select from the new recipe's context menu, **Change input....**

Move or Copy Recipe to a Different Flow

You can move a recipe from the current flow a different one. These steps move a recipe from one flow to another.

- If you want to reuse the recipe in a different flow, create a copy of it first. See above.
- In some cases, it may be easier to duplicate the whole flow and then remove objects from the copied flow. In the Flow View menu, select **Duplicate** from the flow context menu.

Tip: When you move a recipe to a new flow, all attached objects appear in the new flow. If the same objects in the source are used by other recipes, then copies are moved. If the copied object already exists in the target flow, the moved recipe is attached to the corresponding object in the new flow.

Steps:

1. In Flow View, select the recipe to move.
2. In the context panel on the right, select **Move....**

- a. To move to a new empty flow, select `Create New Flow`. You can specify a name for the new flow.
 - b. To move to an existing flow to which you have access, select the flow from the drop-down.
3. Click the **Move** button.
4. The recipe is moved, along with any related objects.

Reuse Recipe in a Different Environment

If you need to reuse a recipe in a different instance of Trifacta®, you have two choices:

1. Export the entire flow and import it into the new environment. Open the flow in the new environment. In Flow View, remove all objects that are not of interest. See *Export Flow*.
2. Turn all of the steps of a recipe into a macro. Export the macro and then import into the new environment. You may choose to remove the macro from the original environment. See *Export Macro*.

Download Recipe

You can download a recipe in text form in the following ways:

NOTE: A downloaded recipe is in a text form of Wrangle (a domain-specific language for data transformation). In this form, it cannot be used in the application. Downloaded recipes are for archival purposes only.

- In Flow View, select the recipe to download. From the context menu, select **Download recipe....**
- In the Recipe panel in the Transformer page, click the context menu, and select **Download recipe as Wrangle**.

Project Management Tasks

These topics provide guidance on how to better manage your data wrangling efforts in Trifacta®.

Take a Snapshot

Contents:

- *Duplicate*
 - *Flows*
 - *Recipes*
 - *Download Work in Progress*
 - *Download Sample Data*
 - *Download Recipe*
 - *Backup*
-

You can use the following techniques to capture snapshots of your Trifacta® application work in progress.

Duplicate

You can make a copy of individual recipes and flows.

NOTE: Copied recipes and datasets are independent objects and do not continue to inherit any changes in the original.

Flows

In Flow View, click the context menu and select **Duplicate**.

NOTE: Sharing permissions are not inherited in the copied flow. You must re-share the flow with any users who need access to the copy.

Recipes

In Flow View, select a recipe to duplicate. In the right panel, select **Make a copy** from the context menu. You can link the recipe to the same inputs or to no inputs.

NOTE: This recipe is still available to all who have access to the flow. If needed, select **Move** to relocate the copied recipe to another flow to which other users do not have access.

Select the copied recipe and click **Edit Recipe** to begin working with the recipe in the Transformer page.

Download Work in Progress

From the Recipe panel in the context panel, you can download your work in progress, including the recipe and the dataset sample as reflected in the current recipe step.

Download Sample Data

From the Transformer page, you can download the dataset sample as it is currently reflected in the Transformer page.

NOTE: A sample downloaded from the Transformer page reflects all recipe steps up to the step that is currently selected. Steps that occur after the current one are not applied to the dataset sample.

Tip: You can use this as a work-in-progress backup if you select the final step of the recipe and if the dataset sample represents the entire dataset.

From the Recipe panel, click the context menu and select **Download Sample data as CSV**.

The CSV file is written to your desktop.

Download Recipe

In the Recipe panel, click the context menu and select **Download recipe as Wrangle**.

The entire recipe is downloaded to your desktop as a text file.

Tip: If you are attempting to capture the recipe as a work-in-progress of the dataset sample, you can just delete the steps that aren't executed from the downloaded file.

Backup

Backups of the Trifacta databases (flows, recipes, and other metadata) and source datastores (imported datasets) should be executed according to your enterprise requirements.

Track Data Changes

Contents:

- *Create Backup*
 - *Track Source Filepath and Filename*
 - *Track Source Row Information*
 - *Track Steps Affecting a Column*
 - *Track Column Value Changes*
 - *Track Row Changes*
-

You can use these techniques for tracking changes to your datasets over time.

Create Backup

After you have created the flow and the datasets within the flow and before applying recipe steps to change the data, create a duplicate of the flow. This becomes a snapshot of your original dataset. Since the imported datasets are not affected, the storage overhead for creating backups is relatively low.

Track Source Filepath and Filename

When you first load your dataset in the Transformer page, you can add the following to capture the full path to the original file that is the source of the data:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	\$filepath
Parameter: New column name	sourceRowNumber

With a few extra steps, you can extract the filename from the above output.

Track Source Row Information

You can mark the original row numbers of your source data. In the first step in your recipe after initial parsing, add the following:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	\$sourcerownumber
Parameter: New column name	sourceRowNumber

This step generates a new column that contains the source row number from the source dataset.

NOTE: Source row information can become invalid if you perform multi-dataset operations such as lookups, unions, and joins. For more precise tracking of source information, you should consider creating multi-column keys, including the source row number information. For more information, see *Generate Primary Keys*.

Track Steps Affecting a Column

To see all of the steps in your current recipe that reference a specific column, select **Show related steps...** from the column menu.

All steps are highlighted in the Recipe panel.

NOTE: If another column is dependent on the selected column, all steps pertaining to that column are highlighted as well.

Track Column Value Changes

Trifacta® enables you to easily move between steps in your transform recipe so that you can check the state of your dataset at any point during the transformation. In some cases, you may want to be able to track the changes made to an individual column side-by-side with the original column. This section provides a generalized approach for tracking column changes in this manner.

NOTE: Use this workflow only if it is important to monitor which values have changed in a column. For most use cases, the Transformer page provides sufficient visibility over your sample data to manage column values.

Steps:

In the following sequence, the original column is called `String`. For numeric columns, you can perform more detailed analysis between original and modified column values.

1. After you have completed your general setup steps of your transform, create a copy of the original column:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	String
Parameter: New column name	String_orig

2. You now have a copy of the original column before any manipulations were applied to it.
3. Add any transforms to your recipe, including any that change the values of `String`. In the example below, the following transform has been applied:

Transformation Name	Edit with formula
Parameter: Columns	String
Parameter: Formula	TRIM(String)

4. At the point in your recipe where you would like to test the column for changes, insert the following:

Transformation Name	New formula
----------------------------	-------------

Parameter: Formula type	Single row formula
Parameter: Formula	String <> String_orig
Parameter: New column name	String_changes

5. The String_changes column now contains true values where the values in String have been changed from their original values (String_orig).

6.

To see just the values that are different, sort in descending order.

Tip: You can reposition this test anywhere in your recipe after you have created the String_orig column.

7. Before you run your recipe, you may want to remove the tracking columns that you generated (String_orig and String_changes in our example).

ABC

String

▼

ABC

String_orig

▼

🔄

String_changes

▼

ABC

Description

▼

<div>7 Categories</div> <div>My String</div> <div>My String extra</div> <div>My String</div> <div>My String</div> <div>MyString</div> <div>My String</div> <div>"My String"</div> <div>"My String"</div> <div>"My String"</div>	<div>9 Categories</div> <div>My String</div> <div>My String extra</div> <div>My String</div> <div>My String</div> <div>MyString</div> <div>My String</div> <div>"My String"</div> <div>"My String"</div> <div>"My String"</div>	<div>2 Categories</div> <div>false</div> <div>false</div> <div>true</div> <div>true</div> <div>false</div> <div>false</div> <div>false</div> <div>false</div> <div>false</div>	<div>9 Categories</div> <div>"Base string: "My String""</div> <div>"Base string + " extra""</div> <div>A space in front of base string</div> <div>A space after base string</div> <div>No space between the two words</div> <div>Two spaces between the two word</div> <div>Base string + a tab character</div> <div>Base string + a return character</div> <div>Base string + a newline character</div>
---	---	--	--

1

Rename all columns by converting row 1 to a header

2

Create String_orig from String

3

Set String to TRIM(String)

4

Create String_changes from String != String_orig

👁

4 Columns

9 Rows

2 Data Types

Figure: Example tracking column changes

Track Row Changes

Steps:

1. In Flow View, create a copy of the flow. Create a name for it that identifies it as your original.
2. In the other flow, create your recipes as normal.
3. When done, you can add the following steps:
 - a. Union the two datasets together.
 - b. Sort them by a key column.
 - c. Add the deduplicate transform.

NOTE: This method may not work if your recipe includes joins or added or removed columns.

4. If the rows are exact duplicates, they are removed. The remaining rows contain data that has been changed.

Add Comments to Your Recipe

As needed, you can insert non-functional comments in your recipes. These comments are stored as a Comment transformation but do not make changes to the dataset.

Tip: Adding comments to your recipes can be helpful for providing notes or other guidance to yourself for later or to other recipe builders who are reviewing your recipe.

Steps:

1. In the Transformer page, open the Recipe panel in the context panel.
2. In your list of recipe steps, select the location in the recipe where you wish to insert the comment. From the recipe step context menu, select the appropriate Insert Step command.
3. In the Search panel, enter `comment`.

Tip: You can also paste full comments of the following format into the textbox. These comments are reformatted into the supported format:

```
// This is a comment.
```

```
/* This is also a comment. */
```

4. In the comment textbox, enter the comment that you would like to include.
5. Click **Save**.
6. The comment is stored in the recipe as text of a different color.

Create Target

Contents:

- *Create Schema*
 - *Before you create*
 - *Create*
 - *Update Schema*
 - *Use Schema*
 - *Remove Schema*
-

To assist in building your recipe, you can associate a target with the recipe. This target schema is displayed in the Target Matching bar in the Transformer page above your column histograms, so that you can track how you are progressing toward completion of the recipe.

- A **target** is the representation of the columns to which you are building your recipe to match. When you know the column order, names, and data types for which you are building your recipe, you can more quickly develop the recipe steps to match this schema.
- For more information, see *Overview of RapidTarget*.

Create Schema

Before you create

A target is created from one of the following sources:

- An imported dataset
- A recipe in the current flow
- A dataset reference from another flow

Before you create a new target, you must identify, create, or import one of the above objects. Your target must be in a finished state.

NOTE: A target is a snapshot of the target at the time of creation. It does not inherit changes from the source after creation. To update the target for later changes, you must delete and recreate the target. Instructions are provided below.

Create

Each recipe can have one and only one target. Please use the following steps to create a target for your recipe.

Steps:

1. Open the flow containing the recipe. In Flow View, create or select the recipe.
2. If a target already exists for the recipe, select **Remove Target** from the context menu in the right panel.

NOTE: A recipe can have one and only one target associated with it.

3. After deleting the old one, from the context menu, select **Assign Target to Recipe**.
 4. Select the imported dataset, recipe, or reference to use as your target for this recipe. Click **Add**.
 5. If the target looks accurate, click **OK**. If not, click **Cancel**.
 6. The target is associated with your recipe.
-

Tip: You can also assign a target from the Transformer toolbar in the Transformer page.

Update Schema

NOTE: You cannot edit a target through Trifacta®. To make changes, remove the target and add in a modified target.

If there are have been changes to the source schema of your target, please complete the following steps to update your target.

Steps:

1. If the source of the target needs to be re-imported into Trifacta, please do so now.
2. In Flow View, select the recipe to which the target is assigned. From the context menu for the recipe, select the following:
 - a. Select **Remove Target** to remove the current target.
 - b. Select **Assign Target to Recipe** to select the new target.

Use Schema

- **Data grid:** After a schema has been associated with your recipe, schema information and a few example rows are displayed in the data grid of the Transformer page. These examples serve to guide your transformation operations.
- **Column Browser:** In the Column Browser panel, you can select one or more columns and apply schema-related transformations to them.

Remove Schema

Steps:

1. In Flow View, select the recipe whose schema you wish to remove.
2. In the right panel, click the context menu. Select **Remove Target**.

NOTE: Removing a target from a recipe does not remove the underlying dataset from the platform.

NOTE: Deleting a dataset does not remove any target based off of it. You can still perform alignment operations to match the schema. However, you cannot view example rows from the target in the Transformer page.

Optimize Job Processing

Contents:

- *Run jobs on the default running environment*
 - *Filter data early*
 - *Perform joins early*
 - *Perform unions late*
-

This page contains a set of tips for how to improve the overall performance of job execution.

Run jobs on the default running environment

When configuring a job, Trifacta analyzes the size of your dataset to determine the best of the available running environments on which to execute the job. This option is presented as the default option in the dialog. Unless you have specific reasons for doing otherwise, you should accept the default suggestion.

Filter data early

If you know that you are deleting some rows and columns from your dataset, add these transformation steps early in your recipe. This reduction simplifies working with the content through the application and, at execution, speeds the processing of the remaining valid data. Since you may be executing your job multiple times before it is finalized, it should also speed your development process.

- To delete columns:
 - Select **Delete** from the column drop-down for individual columns.
 - Use the Delete Columns transformation to remove multiple discrete columns or ranges of columns.
- To delete rows: The following example removes all rows that lack a value for the `id` column:

Transformation Name	Filter rows
Parameter: Condition	Is missing
Parameter: Column	id
Parameter: Action	delete matching rows

- To keep rows: The following example keeps all rows that lack a value in the `id` column:

Transformation Name	Filter rows
Parameter: Condition	Is missing
Parameter: Column	id
Parameter: Action	keep matching rows

See *Filter Data*.

Perform joins early

After you have filtered out unneeded rows and columns, join operations should be performed in your recipe. These steps bring together your data into a single consistent dataset. By doing them early in the process, you reduce the chance of having changes to your join keys impacting the results of your join operations. See *Join Data*.

Perform unions late

Union operations should generally be performed later in the recipe so that you have a small chance of changes to the union operation, including dataset refreshes, affecting the recipe and the output.

NOTE: If your dataset requires a significant amount of data cleaning, you should perform your unions early in your recipe, so that all cleaning steps can be applied once across the dataset.

See *Append Datasets*.

Diagnose Failed Jobs

Contents:

- [Job Types](#)
 - [Identify Job Failures](#)
 - [Invalid file paths](#)
 - [Jobs that Hang](#)
 - [Spark Job Error Messages](#)
 - [Databricks Job Errors](#)
 - [Try Other Job Options](#)
 - [Review Logs](#)
 - [Hadoop logs](#)
 - [Learn More](#)
-

Use these guidelines and features to begin the process of diagnosing jobs that have failed.

Job Types

The following types of jobs can be executed in Trifacta®:

- **Convert jobs:** Some datasources, such as binary file or JSON formats, must be converted to a format that can be easily read by the Trifacta application. During data ingestion, the datasource is converted to a natively supported file format and stored on backend storage.
- **Transform job:** This type of job executes the steps in your recipe against the dataset to generate results in the specified format. When you configure your job, any set of selected output formats causes a transform job to execute according to the job settings.
- **Profile job:** This type of job builds a visual profile of the generated results. When you configure your job, select **Profile Results** to generate a profile job.
- **Publish job:** This job publishes results generated by the platform to a different location or datastore.
- **Ingest job:** This job manages the import of data from a JDBC source into the default datastore for purposes of running a transform or sampling job.

For more information, see [Run Job Page](#).

Tip: Information on failed plan executions is contained in the `orchestration-service.log` file, which can be acquired in the support bundle. For more information, see [Support Bundle Contents](#).

NOTE: For each collected sample, a sample job ID is generated. In the Samples panel, you can view the sample job IDs for your samples. These job IDs enable you to identify the sample jobs in the Sample Jobs page.

Identify Job Failures

When a job fails to execute, a failure message appears in following locations:

- Jobs tab in Flow View.
- Individual job listings in the Jobs page.

The following is an example from the Jobs page:

35	Automated Demo	airports_info Airport Flow	<div> <div>Transform</div> <div>Finished Today at 10:25 AM Environment: Hadoop</div> </div>	<div> <div>Completed</div> <div>Today at 10:26 AM Ran for a minute</div> </div>
33	Automated Demo	airports_info Airport Flow	<div> <div>Profile</div> <div>Finished Today at 10:25 AM</div> </div>	<div> <div>Failed</div> <div>Today at 10:24 AM Ran for 2 minutes</div> </div>
32	elmer	Customer_C Getting Star	<div> <div>Publish</div> <div>Failed Today at 10:25 AM 1 Failed</div> </div>	<div> <div>Completed</div> <div>Today at 10:24 AM Ran for a few seconds</div> </div>

Figure: Publish job failed

In the above example, the Transform and Profile jobs completed, but the Publish job failed. In this case, the results exist and, if the source of the problem is diagnosed, they can be published separately. From the job's context menu, select **Download Logs**. You can download the jobs logs to look for reasons for the failure. See below.

Invalid file paths

When your job uses files as inputs or outputs, you may receive invalid file path errors. Depending on the backend datastore, these can be one of the following:

- Path to the file is invalid for the current user. Path may be been created by another user that had access to the location.
- Path contains invalid characters in it. For more information, see *Supported File Formats*.
- Resource was deleted.

Jobs that Hang

In some cases, a job may stay in a pending state indefinitely. Typically, these errors are related to a failure of the job tracking service. You can try to the following:

- Resubmit the job.
- Have an administrator restart the platform. See *Start and Stop the Platform*.
- Submit the job again.

Spark Job Error Messages

The following error messages may appear in the Trifacta application when a Spark job fails to execute.

"Aggregate too many columns" error

Your job could not be completed due to one or more Pivot, Window or other Aggregation recipe steps having too many aggregate functions in the `values` parameter.

Solution: Please split these aggregates across multiple Aggregation steps.

"Binary sort" error

Sorting a nested column such as an array or map is not supported.

Codegen error

Your job could not be completed due to the complexity of your recipe.

Tips:

- Look to break up your recipe into sequences of recipes. You can chain recipes together one after another in Flow View.

- If you have complex, multi-dataset operations, you should try to isolate these into smaller recipes.
- Use sampling to checkpoint execution after complex steps.

"Colon in path" error

Your job references one or more invalid file paths. File and folder names cannot contain the colon character.

"Invalid input path" error

Your job references one or more invalid file paths. File names cannot begin with characters like dot or underscore.

"Invalid union" error

Union operations can only be performed on tables with compatible column types.

Tip: Edit the union in question. Verify that the columns are properly aligned and have consistent data types. For more information, see *Union Page*.

"Job service unreachable" error

There was an error communicating with the Spark Job Service.

Tip: An administrator can review the contents of the `spark-job-service.log` file for details. See *System Services and Logs*.

"Oom" error

When you encounter out of memory errors related to job execution, you should review the following general items related to your flow.

General Tips:

- Review your recipes to see if you can identify ways to break them up into smaller recipes.
- Operations such as joins and unions can greatly increase the size of your datasets.
- Resource consumption is also affected by the the complexity of your recipe(s).
- If you suspect that there are several jobs running in parallel, you can drop the job launch batch size to 2 or 1 , which serializes job execution while preserving memory. For more information, see *Configure Application Limits*.
- You might be able to configure overrides to the Spark settings to allocate more memory for job execution.
 - This feature may need to be enabled in your environment. See *Enable Spark Job Overrides*.
 - See *Configure User-Specific Props for Cluster Jobs*.

"Path not found during execution" error

One or more datasources referenced by your job no longer exist.

Tip: Review your flow and all of its upstream dependencies to locate the broken datasource. Reference errors for upstream dependencies may be visible in downstream recipes.

"Too many columns" error

Your job could not be completed due to one or more datasets containing a large number of columns.

Tip: A general rule of thumb is to avoid over 1000 columns in your dataset. Depending on your environment, you may experience performance issues and job failures on narrower datasets.

"Version mismatch" error

The version of Spark installed on your Hadoop cluster does not match the version of Spark that Trifacta is configured to use.

Tip: For more information on the appropriate version to configure for the product, see *Configure for Spark*.

Databricks Job Errors

The following error messages are specific to Spark errors encountered when running jobs on Databricks.

NOTE: When a Databricks job fails, the failure is immediately reported in the Trifacta application. Collection of the job log files from Databricks occurs afterward in the background.

Tip: A platform administrator may be able to download additional logs for help in diagnosing job errors.

"Runtime cluster" error

There was an error running your job.

"Staging cluster" error

There was an error launching your job.

Try Other Job Options

You can try to re-execute the job using different options.

Tips:

- **Disable flow optimizations.** If your job is using data from a relational source that supports pushdowns, you can try to disable flow optimizations and then re-run the job. For more information, see *Flow Optimization Settings Dialog*.
- **Look to cut data volume.** Some job failures occur due to high data volumes. For jobs that execute across a large dataset, you can re-examine your data to remove unneeded rows and columns of data. Use the Deduplicate transformation to remove duplicate rows. See *Remove Data*.
- **Gather a new sample.** In some cases, jobs can fail when run at scale because the sample displayed in the Transformer page did not include problematic data. If you have modified the number of rows or columns in your dataset, you can generate a new sample, which might illuminate the problematic data. However, gathering a new sample may fail as well, which can indicate a broader problem. See *Samples Panel*.
- **Change the running environment.** If the job failed on Trifacta Photon, try executing it on another running environment.

Tip: The Trifacta Photon running environment is not suitable for jobs on large datasets. You should accept the running environment recommended in the Run Job page.

Review Logs

Job logs

In the listing for the job on the Jobs page, click **Download Logs** to send the job-related logs to your local desktop.

NOTE: If encryption has been enabled for log downloads, you must be an administrator to see a clear-text version of the jobs listed below. For more information, see *Configure Support Bundling*.

When you unzip the ZIP file, you should see a numbered folder with the internal identifier for your job on it. If you executed a transform job and a profile job, the ZIP contains two numbered folders with the lower number representing the transform job.

`job.log`. Review this log file for information on how the job was handled by the application.

Tip: Search this log file for `error`.

Support bundle: If support bundling has been enabled in your environment, the `support-bundle` folder contains a set of configuration and log files that can be useful for debugging job failures.

Tip: Please include this bundle with any request for assistance to *Alteryx Support*.

For more information on configuring the support bundle, see *Configure Support Bundling*.

For more information on the bundle contents, see *Support Bundle Contents*.

Support logs

For support use, the most meaningful logs and configuration files can be downloaded from the application. Select **Help menu > Download logs**.

NOTE: If you are submitting an issue to *Alteryx Support*, please download these files through the application.

For more information, see *Download Logs Dialog*.

The admin version of this dialog enables downloading logs by timeframe, job ID, or session ID. For more information, see *Admin Download Logs Dialog*.

Trifacta node logs

NOTE: You must be an administrator to access these logs. These logs are included when an administrator downloads logs for a failed job. See above.

On the Trifacta node, these logs are located in the following directory:

```
<install_dir>/logs
```

This directory contains the following logs:

- `batch-job-runner.log`. This log contains vital information about the state of any launched jobs.
- `webapp.log`. This log monitors interactions with the web application.

Issues related to jobs running locally on the Trifacta Photon running environment can appear here.

Hadoop logs

In addition to these logs, you can also use the Hadoop job logs to troubleshoot job failures.

- You can find the Hadoop job logs at port 50070 or 50030 on the node where the ResourceManager is installed.
- The Hadoop job logs contain important information about any Hadoop-specific errors that may have occurred at a lower level than the Trifacta application, such as JDK issues or container launch failures.

Contact Support

If you are unable to diagnose your job failure, please contact *Alteryx Support*.

NOTE: When you contact support about a job failure, please be sure to download and include the entire zip file, your recipe, and (if possible) your dataset.

Learn More

- <https://community.trifacta.com/s/article/how-to-use-the-trifacta-job-log-files>
- <https://community.trifacta.com/s/article/troubleshooting-a-hadoop-job-failure>

Schedule a Job

Contents:

- *Add a Schedule*
- *Schedule a Destination*
- *Edit Schedule*
- *Disable Schedule*
- *Delete*
 - *Delete a schedule*
 - *Delete a destination*

After you have finished developing the recipes in your flow, you can define scheduled executions of the recipe or recipes within the flow to deliver outputs to known locations. Using the Automator, you can automate execution of jobs on source data, which can be replenished with fresh data asynchronously.

NOTE: Before you begin, you should verify that your data management pipeline into and out of the platform has been appropriately defined. This pipeline includes how data is written to the output location. For more information, see *Overview of Automator*.

When you schedule a job, you create two objects in Flow View:

NOTE: You must create both of these objects to schedule a job execution.

Object	Description
Schedule	A schedule applies to the entire flow. It contains one or more intervals at which the recipes of the flow are executed. Recipes are executed if their outputs include a scheduled destination.
Scheduled destination	A scheduled destination is an output location, format, and other settings that is populated with the results of executing the related recipes.

Tip: You can create schedules for datasets with parameters. Any overrides specified through Flow View are automatically applied at runtime for the scheduled job.

Add a Schedule

Steps:

1. Open the flow in Flow View.
2. From the context menu, select **Schedule**.
3. In the Add Schedule dialog, select your scheduling options:
 - a. Timezone: Select the timezone to use to determine when to execute the specified schedule.
 - b. Frequency: Select the time and frequency of execution: Hourly, Daily, Weekly, Monthly, or cron.

NOTE: Scheduling supports a modified version of cron scheduling syntax. For more information, see *cron Schedule Syntax Reference*.

c. To add another scheduled time, click **Add**.

4. To save your schedule, click **Save**.

5. A Calendar icon appears in Flow View to indicate that the flow has a schedule associated with it.

Schedule a Destination

Steps:

1. To specify a destination for your schedule, click the recipe you wish to execute at the scheduled time.
2. If you have not done so already, click the Output icon next to the recipe to create an output for it.
3. In the right panel, locate the Scheduled destinations header. Click **Add**.
4. Specify an output location, format, and updating method.
5. Click **Save**.

Edit Schedule

- To edit the scheduled times, click the Calendar icon. Then, click **Edit**. Make changes as needed and save.
- To edit a scheduled destination, select the output in Flow View. In the right panel, click **Edit** next to the appropriate scheduled destination.

Disable Schedule

- To disable a schedule you control, click the Calendar icon in Flow View. Then, move the slider to disable it.
- Administrators can disable schedules for all flows in the workspace in the Schedules page.

Delete

Delete a schedule

In Flow View, click the Calendar icon. Then, click **Delete**.

Delete a destination

Tip: If you have deleted the schedule for the flow, you do not need to delete the scheduled destination. It cannot run without a schedule.

1. In Flow View, select the output.
2. In the right panel, select **Delete Output** from the context menu.

Create Branching Outputs

From a single collection of datasets, you may need to generate multiple outputs for downstream purposes.

Examples:

- You want to preserve the ability to review and profile your source data. For more information, see *Profile Your Source Data*.
- You need different pivot tables produced from the wrangled data.
- You need to filter down the set of rows or columns to deliver to one user community while delivering a different set of columns to another.

Reshaping Transformations

If your next step is to add any of the following transformations and you wish to preserve the existing data for other uses, you should consider adding these steps in a separate dedicated recipe.

Transformation Name	Description
Union	A union appends one or more datasets to your current one. To preserve the original, you may need to create a branching output. See <i>Append Datasets</i> .
Join	A join combines two datasets based on common values in specified columns in both datasets. These types of transformations can greatly change the shape of your data. See <i>Join Data</i> . Similarly, a lookup uses values from a column in your source data to pull in corresponding rows of data from a reference dataset. These transformations add columns to your dataset. See <i>Add Lookup Data</i> .
Remove duplicate rows	This transformation removes identical rows from your dataset. However, there may be a set of steps required to standardize values in various columns before applying the de-duplication. You may choose to manage this process in a branching recipe.
Delete columns	When a column is removed, it is no longer available for use in any downstream output. See <i>Remove Data</i> .
Filter	Rows can be filtered from your dataset to render different perspectives. These changes may be best moved to a secondary, branching recipe. See <i>Filter Data</i> .
Pivot data	When you create a pivot table, all source data that is not explicitly specified in the pivot is dropped from the dataset. For more information, see <i>Pivot Data</i> .
Group by	You can perform aggregation calculations within a table, which may force column data to be dropped. See <i>Create Aggregations</i> .

Basic Technique

Whenever you are applying a transformation that destroys data or otherwise reshapes your dataset and you wish to preserve the current state of the dataset, you should do the following:

1. In Flow View, select your current recipe. Click **Add new recipe**.
2. This recipe becomes the source for a branched output. Give the new recipe an appropriate name. For example, `Pivot-SalesPerProductPerStore`.
3. For this recipe, click the Output icon. Specify the appropriate output format and location that you'd like to generate for this branched output.
4. Select your current recipe again. Click **Add new recipe**.
5. This recipe becomes the extension of your current recipe. Give the new recipe an appropriate name. For example, `MyRecipe-Part2`.
6. Select the `Pivot-SalesPerProductPerStore` recipe. Click **Edit recipe**.
7. Build your pivot transformation in this recipe.
8. When ready, run the job. The output should be generated in the appropriate format and location.

Tip: When you run a job, all upstream dependencies are generated as part of the job. However, if you have multiple branches in your flow, you must run multiple outputs to generate all of the results. Generating these results may be easier if you create scheduled destinations and then add a schedule to trigger them. For more information, see *Overview of Automator*.

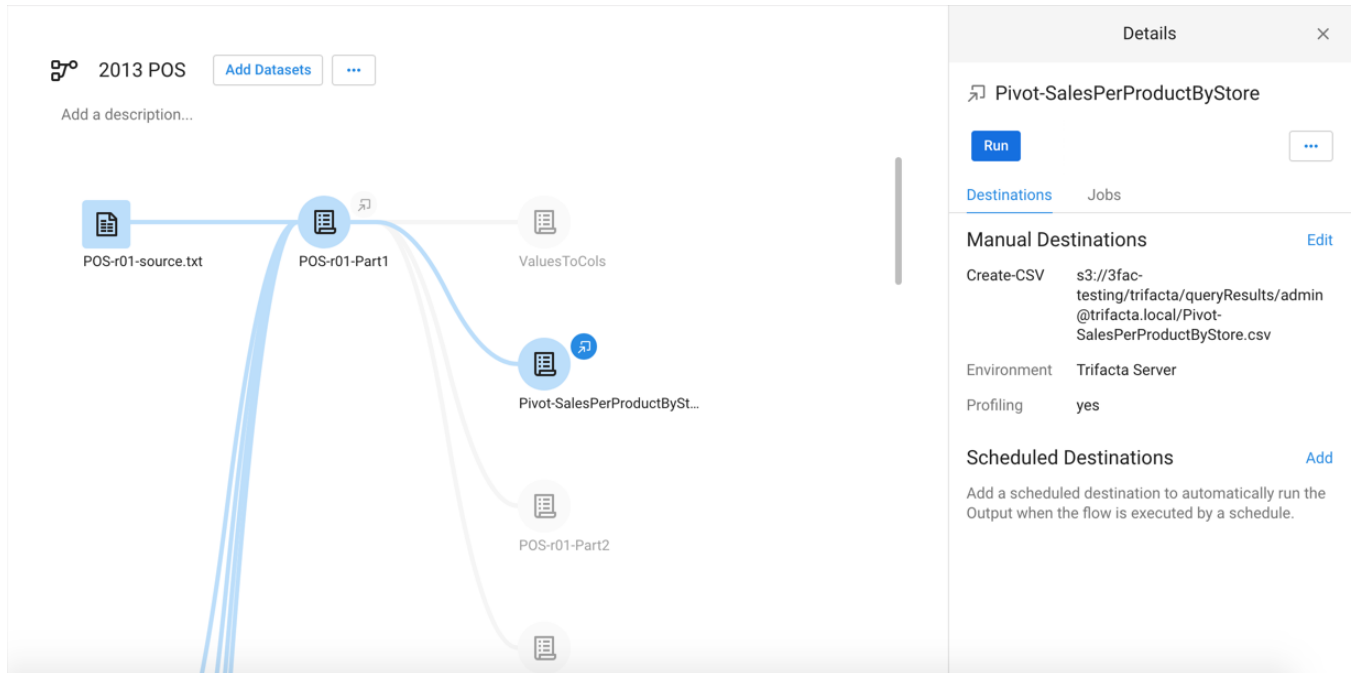


Figure: Multiple pivot tables sourced from output of a primary recipe for the flow. POS-r01-Part2 can be used for continued wrangling of primary recipe.

Build Sequence of Datasets

Contents:

- *Chain Recipes in Same Flow*
- *Create Reference Objects*
- *Create Imported Dataset from Output*

In some situations, you may need to create a sequence of datasets, in which the output of one recipe becomes the input of another recipe.

Potential uses:

1. You may want to handle data cleanup tasks in one set, before that data is made available to other users for customization for their needs.
2. Columns or rows of data may need to be dropped before the dataset is made available to other users.
3. You may want to have different individuals working on each phase of the data transformation process. For example, one individual may be responsible for cleansing the data, while another may be responsible for transforming the data into final format.

Depending on your situation, you can apply one of the following solutions.

Chain Recipes in Same Flow

Within a flow, you can chain together recipes. For example, you may wish to use the first recipe for cleansing and then second recipe for transforming. This method is useful if you are using a single imported dataset for multiple types of transformations within the same flow.

Steps:

1. Click the imported dataset. Click **Add new recipe**.
2. Click the new recipe. Name it, `Cleanse`.
3. With the new recipe selected, click **Add new recipe**.
4. Click the new recipe. Name it, `Transform`.

The output of `Cleanse` recipe becomes the input of `Transform` recipe.

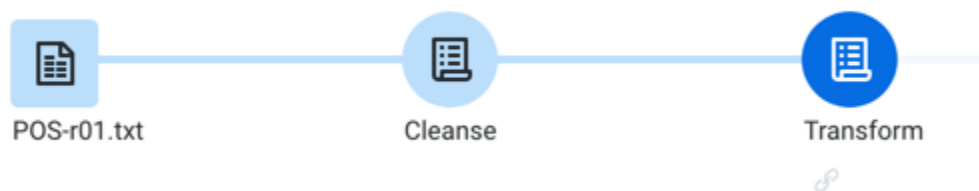


Figure: Chained recipes

Create Reference Objects

If you need to make the output of a recipe available in other flows, you can create a reference object. This reference is available in other flows that you control.

Steps:

1. In Flow View, select the recipe whose output you wish to make available to other flows.
2. Click the Create Reference icon:



Details

×

Transform

Add to Flow...

...

Data Preview

#	Store_Nbr	Item_Nbr	WM_Week
1		381000	201050
2		325000	201049
2		325000	201049
2		403000	201049
2		449000	201049
2		490000	201049
2		560000	201049

Updated

Today at 9:56 AM

Created

Today at 9:56 AM

Used in

0 Flows [More details](#)

Figure: Create reference object

3. To use it in one of your other flows, click **Add to Flow....**
4. In the target flow, the reference object appears as a **reference dataset**. It works like an imported dataset with the following considerations.

Key Considerations:

- When you run a job in a flow that contains a reference dataset, all upstream dependencies of that reference dataset are executed. For the source reference object, all imported datasets and recipes are gathered and executed to populate the reference dataset with fresh data.
- The above has the following implications:
 - If the user running the job in flow #2 does not have permissions to access all of the upstream dependencies of the reference dataset, the job may fail. These dependencies include imported datasets and any connections.
 - If the upstream objects are owned by other users, you may not be able to review these items. For example, if the source recipe is changed by another user, your downstream recipe may break without notice. If you cannot review that recipe, then you can see what was changed and how to fix it.

Create Imported Dataset from Output

If any of the above considerations are a concern, you can create an imported dataset from the job results of flow #1.

In the Job Details page, click the Output Destinations tab. For the generated output, select **Create imported dataset** from its context menu.

From the results of wrangling your first dataset, you can create a new dataset. This dataset is wrangled in a separate recipe, the output of which can become a third dataset. In this manner, you can create sequences of datasets.

Key Considerations:

- The imported dataset in flow #2 is not refreshed until you run the job that generates it in flow #1.

- If the output of flow #1 uses the same filename each time, you may not know if the data has been refreshed. When the job is executed in flow #2, it collects the source imported dataset and executes, whether the data is new or not. Workarounds:
 - **Dataset with parameters:** In flow 2, you can create a parameterized dataset, which collects source data, with some variation in parameters. As long as the output of flow #1 follows the naming convention for the parameterized dataset for flow #2, you should be able to run the job on fresh data on-demand.
 - After the job in flow #2 executes, rename or remove the output of flow #1 from its target location. That way, whenever job #2 executes again, any data that it collects from the source location is likely to be newer.

Fix Dependency Issues

Contents:

- *How to Identify*
 - *Dependent datasets*
 - *Broken data integrations*
 - *Hidden breakages*
- *Fixing Dependencies*

This section describes how to identify dependency issues between your current recipe and other recipes or datasets and includes general steps for fixing them.

Where possible, changes made in one dataset or recipe propagate to the datasets that consume it. Datasets that join, union, or lookup against your dataset are likely to be impacted if you delete columns or rows or otherwise change the data. In some cases, the recipes of these dependent datasets can break.

How to Identify

Dependent datasets

When making edits to a recipe, you can verify if your changes potentially impact other recipes or reference datasets that rely on it. In the Transformer page, click the drop-down next to the current dataset's name to open the Recipe Navigator. Select the Flow View tab.

Tip: If your current dataset is connected to datasets to the right of it, those datasets are dependent on the current one. After you make changes to the current one, you should use the Recipe Navigator to open recipes and datasets that are connected to it and to the right of it in flow view.

Broken data integrations

When you make some changes in an upstream recipe or dataset, the recipes for any downstream datasets can break, such that you cannot generate satisfactory results. In the downstream recipe, you may see errors in the Recipe panel, such as the following:

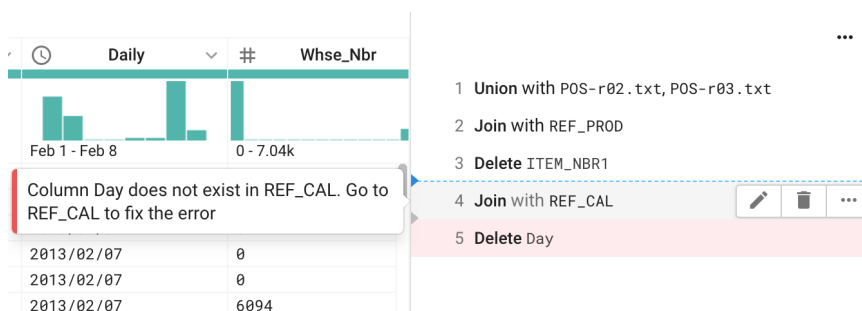


Figure: Dependency error in the Recipe panel

In the above, the column `Day` does not exist in the current dataset, which is causing problems in the last two recipe steps. These types of errors may be generated when a column in the upstream dataset has been dropped or renamed.

Steps:

1. Open the object where the column was dropped:
 - a. If the recipe or dataset is from the same flow, you can use the Recipe Navigator in the Transformer Page. See *Recipe Navigator*.
 - b. If the recipe or dataset is in a different flow, use the Flows page to locate it (`REF_CAL.txt` in the above).
2. In the Flow View tab, open the dataset referenced in the error message.
3. In the Recipe panel, locate the step where the column was removed.
4. Fix the issue. Details are below.

Hidden breakages

If you make changes to specific values in a dataset, recipe steps in downstream datasets can break if they rely on detecting specific values. Depending on the usage, the step may not actually be broken, but the generated results are incorrect.

For example, a downstream dataset recipe includes the following step:

Transformation Name	Filter rows when value is exactly
Parameter: Condition	Is exactly
Parameter: Column	company_name
Parameter: Value	'My Co.'
Parameter: Action	Delete matching rows

If the `company_name` column is sourced from another dataset and the `My Co.` value is changed to `My Company`, the downstream dataset that includes this transform doesn't break in an easily noticeable way. The data is simply not removed from the dataset and any generated results.

Fixing Dependencies

When you locate a dependency issue in the upstream dataset, you can fix it using one of the following methods:

1. Fix the issue in the source dataset. Verify that the change does not impact other datasets.

NOTE: If you fix the issue in the source dataset, you should verify if any other downstream datasets are impacted by this change.

2. Change the input dataset to use a dataset that is not broken.

Tip: If you must freeze the data in the dataset that you are using as an input, you can create a copy of the dataset as a snapshot from the Dataset Details page.

To use the copy, repair or rebuild the integration using the copied version.

3. Fix the issue in the dataset that depends on it. In this case, you must redefine the transformation that brings in the data.

Share a Flow

You can allow other users to work on flows that you own. Flow **collaborators** have almost all of the same permissions as flow **owners**.

NOTE: Users of a shared flow must have read access to the underlying data sources to access the datasets of a shared flow. If they do not have dataset access, collaborators can still access the flow but have more limited capabilities.

NOTE: When you share a flow that contains a dataset sourced from Microsoft Excel, the user with whom the flow is shared may receive a `Could not parse` error. In this case, the user does not have access to the original sample. The workaround is to take a new sample or to run a job on the full dataset.

Steps:

1. In the Flows page, locate the flow to share.
2. From the context menu on the right side of the screen, select **Share**.
3. In the Share Flow dialog, enter the name of the user or users with whom you would like to share the flow.
4. You can specify the privilege level of the user to whom you are sharing.
5. Click **Add**.
6. The selected users can now see the flow and interact with its objects in the Shared with Me tab of the Flows page.

For more information on the privileges of collaborators, see *Overview of Sharing*.

Export Flow

As needed, you can export a flow from Trifacta®. An exported flow is stored in a ZIP file that contains all objects needed to use the flow in any instance of that platform that can access the flow's sources. Exported flows can be imported into the same system or different systems.

Flow export is useful for:

- Backups of work in progress

You cannot import flows that were exported from a version before Release 6.8.

- Archiving of completed development work
- Migrating flows from one instance to another
- Deployment of work to Production environments

An exported flow ZIP also includes:

- Any `.data` files, which may be included as artifacts of feature usage.
 - For transformation by example, artifact files include the value transformation information for the TBE step. For more information, see *Overview of TBE*.
 - For cluster clean, artifact files contain the mappings between source values and clustered values. For more information, see *Overview of Cluster Clean*.
- Any configured webhook tasks are part of the flow definition. For more information, see *Create Flow Webhook Task*.

Export from Flows Page

Steps:

1. From the menu, select **Flows**.
2. In Flows page, locate the flow to export. From the context menu, select **Export**.
3. To export, click **Download**.
4. The ZIP file is downloaded to the default download location on your local desktop.

Tip: You can also export from Flow View.

NOTE: When you import a flow, you import this ZIP file. You cannot import the contents of the ZIP. If your local environment automatically unzips ZIP files, please re-ZIP before you import. For more information, see *Import Flow*.

Export from Production instance

Tip: In general, avoid making changes in a Production environment. Instead, you should make changes in a Development environment, export from there, and reimport into the Production environment.

Steps:

1. Login to the Production instance. The Deployment Manager is displayed.
2. From the menu, select **Deployments**.
3. Select the deployment that you wish to export.
4. In the list of releases, locate the release to export. From the context menu, select **Export**.
5. Add any optional notes for the export. When the flow is imported into another environment, this notes are displayed in the user interface.
6. To export, click **Download**.
7. The ZIP file is downloaded to the default download location on your local desktop.

This file can be stored for safekeeping or imported back into the instance. For more information, see *Import Flow*.

Import Flow

Contents:

- *Limitations*
 - *Define import rules*
 - *Import*
 - *Import into Prod instance*
-

An exported flow can be imported into Trifacta®.

- **Dev instance:** If you are using an instance of the platform for developing and testing your flows, you can import a new flow through the Flows page.

NOTE: Unless your instance of the platform has been specifically configured to support deployment management, you are using a Dev instance of the platform.

NOTE: If you are attempting to share a flow with other users on the same instance of the platform, you should use the sharing functions. See *Overview of Sharing*.

- **Prod instance:** If you are importing a flow into a Production instance of the platform, you import it as a package through the Deployment Manager.

NOTE: Deployment Manager is a feature that enables segmentation of platform usage between Dev instances and Prod instances. This feature must be enabled and configured. For more information, see *Overview of Deployment Manager*.

Limitations

You cannot import flows that were exported before Release 6.8. See *Changes to the Object Model*.

NOTE: You cannot import flows into a version of the product that is earlier than the one from which you exported it. For example, if you develop a flow on free Trifacta Wrangler, which is updated frequently, you may not be able to import it into other editions of the product, which are updated less frequently.

Imported flows do not contain the following objects:

NOTE: Depending on the import environment, some objects in the flow definition may be incompatible. For example, the connection type may not be valid, or a datasource may not be reachable. In these cases, the objects may be removed from the flow, or you may have to fix a reference in the object definition. After import, you should review the objects in the flow to verify.

- Reference datasets
- Samples
- Connections

NOTE: Exported flows do not contain connections. If your flow relies on a connection to the source, you must create the connection in the Prod environment and create an import mapping rule to assign the local connection ID to the import package. Flows that do not require connections may not require remapping before import. See *Define Import Mapping Rules*.

NOTE: If the flow's output object uses connections that are not used for importing datasets in the flow, the output is broken on import. Those outputs and their associated connections must be recreated in the environment into which the flow is imported.

Imported datasets that are ingested into backend storage for Trifacta may be broken after the flow has been imported into another instance. These datasets must be reconnected to their source. You cannot use import mapping rules to reconnect these data sources. This issue applies to the following data sources:

- Microsoft Excel workbooks and worksheets. See *Import Excel Data*.
- PDF tables. See *Import PDF Data*.

Define import rules

Before you import a package, you may need to create import mapping rules to apply to your package. For example, if the Development data is stored in a different location than the Production data, you may need to create import rules to remap paths and connections to use to acquire the data from the Production environment.

NOTE: Import rules are applied at the time of import. They cannot be retroactively applied to releases that have already been imported.

For more information, see *Define Import Mapping Rules*.

Import

NOTE: You cannot import into a Dev instance if your account for the instance contains the Deployment role.

NOTE: If the exported ZIP file contains a single JSON file, you can just import the JSON file. If the export ZIP also contains other artifact files, you must import the whole flow definition as a ZIP file. For best results, import the entire ZIP file.

Steps:

1. Export the flow from the source system. See *Export Flow*.
2. Login to the import system, if needed.
3. Click **Flows**.

4. From the context menu in the Flow page, select **Import Flow**.

Tip: You can import multiple flows (ZIP files) through the file browser or through drag-and-drop. Press **CTRL/COMMAND + click** or **SHIFT + click** to select multiple files for import.

5. Select the ZIP file containing the exported flow. Click **Open**.

If there are issues with the import, click the Download link to review the missing or malformed objects.

Tip: When you import the flow, click the Warnings link to review the list of objects that must be remapped.

The flow is imported and available for use in the Flows page. After import:

- You may need to reconnect your imported datasets to data sources that are available in the new workspace or project. See *Reconnect Flow to Source Data*.
- You may also need to reconnect your outputs. See *Reconnect Flow to Outputs*.

Import into Prod instance

After creating any import rules in your Prod instance, please do the following.

Steps:

1. Export the flow from the source system. See *Export Flow*.
2. Login to the Prod instance. The Deployment Manager is displayed.
3. Click **Deployments**.
4. Select or create the deployment into which to import the package.
5. Within the deployment, click **Import Package**.
6. Select the ZIP file containing the exported flow. Click **Open**.
 - a. Any defined import rules are applied to the package during import.
 - b. The package is selected as the active one for the deployment.
 - c. If there are issues with the import, click the Download link to review the missing or malformed objects.

Tip: After you import, you should open the flow in Flow View and run a job to verify that the import was successful and the rules were applied. See *Flow View Page*.

Reconnect Flow to Source Data

When you import a flow into a new project or workspace, you may need to remap the imported datasets in the flow to source data.

Tip: When you import the flow, click the Warnings link to review the list of objects that must be remapped.

When the flow is imported from one instance to another instance, the imported datasets may be broken after the flow has been imported into another instance. You must map the imported flow to the corresponding data sources.

Steps:

1. Open the imported flow.
2. In Flow View, the datasets that need remapping have a red dot in the corner of their icon. This dot means that the Trifacta application is unable to connect to the dataset.
3. In the Connections page, you may need to recreate the connections that were used to import the dataset in the original workspace.
4. For each broken dataset:
 - a. Right-click the dataset and click **Replace**.
 - b. From the Replace dialog, select the existing dataset or click **Import Datasets** and select the required dataset.
 - c. Select **Replace**.
 - d. The selected dataset is replaced with another dataset.
5. Repeat the above steps for each broken dataset in the flow.

You may also need to remap the flow's outputs.

Reconnect Flow to Outputs

When you import a flow into a new project or workspace, you may be required to remap the flow outputs to accessible publishing destinations.

Tip: When you import the flow, click the Warnings link to review the list of objects that must be remapped.

Tip: You should remap the data sources first. See *Reconnect Flow to Source Data*.

Steps:

1. Open the imported flow.
2. In Flow View, for each output:
 - a. Select the required output. The object details are displayed in the Details panel.
 - i. If you cannot connect to the data, you do not have permissions to use the connection specified in the flow or the connection may not be available in the current project or workspace. You must create a new connection to access the source data.
 - b. In the Details panel, click **Edit** or **Add**. The Publishing Settings page is displayed.
 - c. Edit the changes as required.
 - d. To save your changes, click **Update**.
3. Repeat the above steps for the other outputs in the flow.
4. To verify, run a job that generates one of the outputs.

Define Import Mapping Rules

Contents:

- *Import Rules*
 - *Notes on import rules*
- *Import Rule Requirements*
- *Import Rule Types*
 - *Object Mapping Types*
 - *Value Mapping Types*
- *Examples*
 - *Example - Replace a connection*
 - *Example - Remap an HDFS location*
 - *Example - Remap an S3 location*
 - *Example - Remap a WASB location*
 - *Example - Remap an ADLS Gen2 location*
 - *Example - Remap an ADLS Gen1 location*
 - *Example - Remap a relational datasource*
- *Import Dry-Run*

Before you import a packaged flow into a Production environment, you may need to apply import rules to remap objects and locations from the source instance to the new instance. Import mapping rules are not required when importing into the same environment, although they may be helpful in some cases.

Tip: If you are importing a flow that references file-based sources and wish to use the original files in your imported file, you may find it easier to configure the importing user's permissions to access the appropriate directories of the sources and then to swap datasets as needed after you complete the import. This method is suitable and easier to do across a fewer number of flows.

NOTE: Import mapping rules apply to deployments in a Production instance under deployment management. You cannot apply import mapping rules between two Dev instances.

NOTE: Import mapping rules require the use of the APIs made available from the Trifacta® platform. API usage is considered a developer-level skill.

- For more information on creating an export package, see *Export Flow*.
- For more information on how to import, see *Import Flow*.

You can apply the following types of remappings:

Type	Description
Value	<p>For value remappings, you can specify rules to match on specific values or patterns of values in the import package and remap those values for use in the new instance.</p> <div>NOTE: In this release, value remapping is supported only for S3 bucket names and paths to imported datasets and output locations. Examples are provided below.</div>
Object	<p>For object remappings, you can specify rules to match a value listed in the import package and remap that value to a defined object in the new instance.</p> <div>NOTE: In this release, object remapping is supported only for connections. An example is provided below.</div>

Import Rules

When a flow is imported, references in the flow definition that apply in the source instance may not apply in the target instance. For example, the location paths to the source datasets may need to be rewritten to point to a different location in the target instance.

Before you import your flow definition, you need to define rules for any value or object remapping that must be done in the target environment.

Notes on import rules

1. Value and object remapping rules should be completed before you import the flow. The flow may be non-functional until the rules are applied.

Tip: After you create your import rules, you can perform via API a dry run of the import. Any errors are reported in the response. Details are provided below.

2. Value and object remapping rules are applied at the time of import. If you add new rules, they are not retroactively applied to release packages that have already been imported.
3. When changing rules:
 - a. Any previously applied rules to the same import object are deleted.
 - b. You can apply multiple rules in the same change.
 - c. Rules are applied in the order in which they are listed in the request. Rules listed later in the request must be compatible with expected changes applied by the earlier rules.
4. Value and object remapping must be completed via API. API usage is considered a developer-level skill. Examples are provided below.

NOTE: Import mapping rules do not work for parameterized datasets. If the imported dataset with parameters is still accessible, you should be able to run jobs from it.

Import Rule Requirements

- If you are importing into the same instance from which you exported (Dev/Test/Prod on the same instance):
 - Import rules are not required.
 - If you want to use a different source of data in your Prod flow, you must create import rules.
- If you are importing into a different instance from which you exported (Dev and Prod on different instances):
 - Import rules are required, except in unusual cases.

Import Rule Types

The following types of rules can be applied to import mappings.

NOTE: Depending on the type of mapping, some of these rules may not apply. Please be sure to review the Examples below.

Object Mapping Types

Type	Description
table Name	Set this value to <code>connections</code> . You must then specify the <code>uuid</code> of the connection identifier in the imported flow and replace it with the internal identifier of the connection in the importing instance.

Value Mapping Types

Type	Description
fileLocation	This type is used to remap paths to files. <div>NOTE: filelocation rules apply to both input and output paths. Paths and their rules should be defined with care.</div>
s3Bucket	(AWS) Name of the S3 to remap.
dbTableName	(relational source) Name of the table to remap.
dbPath	(relational source) Path to the database table. This value is an array.
host	(Azure) Depending on the Azure datastore, this rule replaces: <ul style="list-style-type: none">• WASB: blobhost name• ADLS Gen2: storage account• ADLS Gen1: datastore in the datalake
userinfo	(Azure) Depending on the Azure datastore, this rule replaces: <ul style="list-style-type: none">• WASB: container name• ADLS Gen2: filesystem name

Examples

The following are some example import rules to address specific uses.

Example - Replace a connection

In this following example, you must remap the connection from the source instance of the platform to the corresponding connection in the instance where you are importing.

First, you must be able to uniquely identify the connection from the source that you wish to remap.

- While the connection Id may work in a limited scope, that identifier is unlikely to be unique within your environment.
- If you do know the connect Id from the source system, you can skip the first step below.

In the API response in a connection definition, you can acquire the `uuid` value for the connection, which is a unique identifier for the connection object across all instances of the platform:

Item	v4 APIs
API Endpoint	From the source instance: <div>/v4/connections</div>
Method	<div>GET</div>
Request Body	None.
Response Body	<div>{</div>

```

"data": [
  {
    "connectParams": {
      "vendor": "redshift",
      "vendorName": "redshift",
      "host": "redshift.example.com",
      "port": "5439",
      "extraLoadParams": "BLANKSASNULL EMPTYASNULL TRIMBLANKS TRUNCATECOLUMNS",
      "defaultDatabase": "test"
    },
    "id": 2,
    "host": "redshift.example.com",
    "port": 5439,
    "vendor": "redshift",
    "params": {
      "extraLoadParams": "BLANKSASNULL EMPTYASNULL TRIMBLANKS TRUNCATECOLUMNS",
      "defaultDatabase": "test"
    },
    "ssl": false,
    "vendorName": "redshift",
    "name": "redshift",
    "description": null,
    "type": "jdbc",
    "isGlobal": true,
    "credentialType": "iamRoleArn",
    "credentialsShared": true,
    "uuid": "097c2300-2f6a-11e9-a585-57562e0d9cd6",
    "disableTypeInference": false,
    "createdAt": "2019-02-13T08:33:28.368Z",
    "updatedAt": "2019-02-13T08:33:28.381Z",
    "credentials": [
      {
        "iamRoleArn": "arn:aws:iam:something",
        "username": "UserName"
      }
    ],
    "creator": {
      "id": 1
    },
    "updater": {
      "id": 1
    },
    "workspace": {
      "id": 1
    }
  },
  {
    "connectParams": {
      "vendor": "hive",
      "vendorName": "hive",
      "host": "hadoop",
      "port": "10000",
      "jdbc": "hive2",
      "defaultDatabase": "default"
    },
    "id": 1,
    "host": "hadoop",
    "port": 10000,
    "vendor": "hive",
    "params": {
      "jdbc": "hive2",
      "connectStringOptions": "",
      "defaultDatabase": "default"
    },
    "ssl": false,
    "vendorName": "hive",
    "name": "hive",
    "description": null,
    "type": "jdbc",
    "isGlobal": true,
    "credentialType": "conf",

```

	<pre> "credentialsShared": true, "uuid": "08a1a180-2f6a-11e9-b2b2-85d2b0b67f5e", "disableTypeInference": false, "createdAt": "2019-02-13T08:33:26.936Z", "updatedAt": "2019-02-13T08:33:26.952Z", "credentials": [], "creator": { "id": 1 }, "updater": { "id": 1 }, "workspace": { "id": 1 } }, "count": 2 } </pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/getConnection

In the above, you identify that the connection used for the exported flow is the Redshift one. This object has the following unique identifier:

```
"uuid": "097c2300-2f6a-11e9-a585-57562e0d9cd6"
```

In the target system, you must now create a rule in the deployment into which you are importing that searches for this unique value. In the following example:

- The deploymentId is known to be 4.
- The connectionId for the equivalent Redshift connection in the target system is 1.

The uuid field in the import package is searched for the matching string. If it is found, the connection in the import package is replaced with the connection in the target system with an Id of 1:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/objectImportRules</code>
Method	PATCH
Request Body	<pre> [{ "tableName": "connections", "onCondition": { "uuid": "097c2300-2f6a-11e9-a585-57562e0d9cd6" }, "withCondition": { "id": 1 } }] </pre>
Status Code - Success	200 - OK
Response	

Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateObjectImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap an HDFS location

In this example, your import rule must remap the path to the source from your Dev paths to your Prod paths. Suppose the pattern looks like this:

Dev Path	hdfs://datasets/dev/1/164e0bca-8c91-4e3c-9d0a-2a85eedec817/myData.csv
Prod Path	hdfs://datasets/prod/1/164e0bca-8c91-4e3c-9d0a-2a85eedec817/myData-Prod.csv

Note the differences:

- The `/dev/` part of the path has been replaced by `/prod/`.
- The filename is different.

You can use the following value import rules to change the path values. In the following example, the rules are applied separately.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

Request:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/valueImportRules</code>
Method	PATCH

Request Body:

```
[
  { "type": "fileLocation", "on": "\\dev\\", "with": "/prod/" },
  { "type": "fileLocation", "on": "\\([a-zA-Z0-9_]*).csv/", "with": "$1-Prod.csv" }
]
```

Response:

Item	v4 APIs
Status Code - Success	200 - OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap an S3 location

For S3 sources, you can apply remapping rules including changing to a new S3 bucket.

In this example, your import rule must remap the path to the source from your Dev paths to your Prod paths. Suppose the pattern looks like this:

Dev S3 Bucket Name	wrangle-dev
Dev Path	/projs/tweets/v04/tweets_month.csv
Prod S3 Bucket Name	wrangle-prod
Prod Path	/tweets/tweets_month.csv

You can use the following value import rules to change the bucket name and path values.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

s3Bucket name rule: This rule replaces the name of the S3 bucket to use with the new one: wrangle-prod.

fileLocation rule: This rule uses regular expressions to match each segment of the path in the bucket's paths.

- Files are located at a consistent depth in the source bucket.
- Path segments and filename use only alphanumeric values and underscores (_).
- The replacement path is shortened to contain only the parent name (\$2) and the filename (\$4) in the path.
- This rule applies to both input and output object file paths.

Request:

Item	v4 APIs
API Endpoint	/v4/deployments/4/valueImportRules

Method	PATCH
--------	-------

Request Body:

```
[
  { "type": "s3Bucket", "on": "wrangle-dev", "with": "wrangle-prod" },
  { "type": "fileLocation", "on": "/\\([a-zA-Z0-9_]*\\)\\([a-zA-Z0-9_]*\\)\\([a-zA-Z0-9_]*\\)\\([a-zA-Z0-9_]*\\).csv/", "with": "/$2/$4.csv" }
]
```

Response:

Item	v4 APIs
Status Code - Success	200 - OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap a WASB location

For WASB sources, you can apply remapping rules during import.

In this example, your import rule must remap the blob host, container, and file location:

Dev Blobhost	storage-wasb-account-dev.blob.core.windows.net
Dev Container	container-dev
Dev File Location	/projs/work/orders.csv
Prod Blobhost	storage-wasb-account-prod.blob.core.windows.net
Prod Container	container-prod
Prod File Location	/2003/transactions/orders.csv

You can use the following value import rules to change the blobhost, container, and file paths.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

host rule: This rule replaces the blobhost name to use with the new one: `storage-wasb-account-prod.blob.core.windows.net`.

userinfo rule: This rule replaces the container name to use with the new one: `container-prod`.

fileLocation rule: This rule performs a text substitution to replace the file path. This rule applies to both input and output object file paths.

Request:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/valueImportRules</code>
Method	PATCH

Request Body:

```
[
  {
    "type": "host", "on": "storage-wasb-account-dev.blob.core.windows.net", "with": "storage-wasb-account-prod.blob.core.windows.net",
  },
  {
    "type": "userinfo", "on": "container-dev", "with": "container-prod",
  },
  {
    "type": "fileLocation", "on": "/projs/work/orders.csv", "with": "/2003/transactions/orders.csv"
  }
]
```

Response:

Item	v4 APIs
Status Code - Success	200 - OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap an ADLS Gen2 location

For ADLS Gen2 sources, you can apply remapping rules during import.

In this example, your import rule must remap the storage account, filesystem, and file location:

Dev Storage Account	<code>storage-adlsgen2-account-dev.blob.core.windows.net</code>
---------------------	---

Dev Filesystem	filesystem-dev
Dev File Location	/projs/work/orders.csv
Prod Storage Account	storage-adlsgen2-account-prod.blob.core.windows.net
Prod Filesystem	filesystem-prod
Prod File Location	/2003/transactions/orders.csv

You can use the following value import rules to change the storage account, filesystem, and file paths.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

host rule: This rule replaces the storage account name to use with the new one: storage-adlsgen2-account-prod.blob.core.windows.net.

userinfo rule: This rule replaces the filesystem name to use with the new one: filesystem-prod.

fileLocation rule: This rule performs a text substitution to replace the file path. This rule applies to both input and output object file paths.

Request:

Item	v4 APIs
API Endpoint	/v4/deployments/4/valueImportRules
Method	PATCH

Request Body:

```
[
  { "type": "host", "on": "storage-adlsgen2-account-dev.blob.core.windows.net", "with": "storage-adlsgen2-account-prod.blob.core.windows.net" },
  { "type": "userinfo", "on": "filesystem-dev", "with": "filesystem-prod" },
  { "type": "fileLocation", "on": "/projs/work/orders.csv", "with": "/2003/transactions/orders.csv" }
]
```

Response:

Item	v4 APIs
Status Code - Success	200 - OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": {</pre>

	<pre> "data": [] } } </pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap an ADLS Gen1 location

For ADLS Gen1 sources, you can apply remapping rules during import.

In this example, your import rule must remap the Azure data lake store and file location:

Dev data store	adl://storage-adlsgen1-account.azuredatalakestore.net
Dev File Location	/projs/work/orders.csv
Prod data store	adl://storage-adlsgen1-account-prod.azuredatalakestore.net
Prod File Location	/2003/transactions/orders.csv

You can use the following value import rules to change the datastore and file paths.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

host rule: This rule replaces the datastore name to use with the new one: storage-adlsgen1-account-prod.azuredatalakestore.net.

fileLocation rule: This rule performs a text substitution to replace the file path. This rule applies to both input and output object file paths.

Request:

Item	v4 APIs
API Endpoint	/v4/deployments/4/valueImportRules
Method	PATCH

Request Body:

<pre> [{ "type": "host", "on": "storage-adlsgen1-account-dev.azuredatalakestore.net", "with": "storage-adlsgen1-account-prod.azuredatalakestore.net" }, { "type": "fileLocation", "on": "/projs/work/orders.csv", "with": "/2003/transactions/orders.csv" }] </pre>

Response:

--	--

Item	v4 APIs
Status Code - Success	200 – OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Example - Remap a relational datasource

When you migrate a relational source from a Dev instance to a Prod instance, you may need to remap your flow to use the production database and table.

NOTE: These rules can be applied to sources or publications of a flow.

In this example, you are replacing the input and output source databases and tables with the corresponding production DB values.

Item	Dev value	Prod value
Table name 1	dev_trans	prod_trans
Path value 1	dev_db2_src	prod_db2_src
Table name 2	dev_trans_out	prod_trans_out
Path value 2	dev_db2_out	prod_db2_out

In a single request, you can apply the rules changes to map the above Dev values to the Prod values.

NOTE: You can specify multiple rules in a single request. Rules are applied in the order that they are listed. Latter rules must factor the results of earlier rules.

The `on` parameter accepts regular expressions. In the following example request, the `on` parameter has been configured to use a regular expression, under the assumption that all current and future imports will respect the current pattern or database paths and table names.

dbTableName rule: This rule replaces the name of the table to use.

dbPath rule: This rule replaces the path value to database table.

NOTE: The content of a dataset or output `dbPath` is an array. The regular expression for `on` is applied to every element in the `dbPath` value. Typically, there's only one element in the `dbPath` array. In some cases, there may be multiple elements, so be careful when specifying the `on` value.

Request:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/valueImportRules</code>
Method	PATCH

Request Body:

```
[
  {
    "type": "dbTableName", "on": "/dev_([a-zA-Z0-9_]*)/", "with": "prod_$1",
    "type": "dbPath", "on": "/dev_([a-zA-Z0-9_]*)_src/", "with": "prod_$1_out"
  }
]
```

Response:

Item	v4 APIs
Status Code - Success	200 - OK
Response Body	<p>When the new rules are applied, all previously existing rules for the object in the deployment are deleted. The response body contains any rules that have been deleted as part of this request.</p> <p>In the following example, there were no rules, so nothing was deleted:</p> <pre>{ "deleted": { "data": [] } }</pre>
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/updateValueImportRules

To test your rule, perform a dry run of the import. See below.

Import Dry-Run

After you have specified a set of import rules, you can perform a dry-run of an import of an import package. This dry-run does not perform the actual import but does report any permissions errors or other issues in the response.

In this example, the `flow2import.zip` file contains the package to import into deployment 4.

Request:

Item	v4 APIs
API Endpoint	<code>/v4/deployments/4/releases/dryRun</code>
Method	POST

Request Body	In form data submitted with the request, you must include the following key-value pair:				
	<table> <tr> <th>Key</th><th>Value</th></tr> <tr> <td>data</td><td>"@flow2import.zip"</td></tr> </table>	Key	Value	data	"@flow2import.zip"
Key	Value				
data	"@flow2import.zip"				

Response:

Item	v4 APIs
Status Code - Success	200 – OK
Response Body	The response body contains any import remapping rules that have been applied during the import process.
Documentation	See https://api.trifacta.com/ee/es.t/index.html#operation/importPackageForDeploymentDryRun

After the above dry-run has been executed, the import package can be imported and is automatically connected to the appropriate connection. See <https://api.trifacta.com/ee/es.t/index.html#operation/importPackageForDeployment>

Create or Replace Macro

Contents:

- *Create Macro*
 - *Define macro inputs*
 - *Edit Macro*
 - *Convert Macro to Steps*
 - *Replace Macro*
 - *Replace macro with another macro*
 - *Replace macro with steps*
 - *Update macro inputs*
 - *Inspect Macro*
 - *Apply Macro*
 - *Manage Macros*
-

<ac:structured-macro ac:name='excerpt' ac:schema-version='1'><ac:parameter ac:name='atlassian-macro-output-type'>BLOCK</ac:parameter><ac:rich-text-body>You can create reusable macros from sequences of steps in your recipe. These macros can be applied in other locations of the recipe or in other recipes. If needed, you can modify the steps in an instance of the macro to replace the existing steps, allowing you to make changes and updates to your macros.</ac:rich-text-body></ac:structured-macro>

Macro Definition:

Macros are user-defined sequences of recipe steps that can be referenced independently and parameterized as needed. A macro is composed of the following types of information:

- **Steps** are the recipe steps that are executed each time that the macro is invoked. A macro contains one or more steps.
- **Inputs** are variables that can be modified wherever the macro is placed. For example, you might have an input that contains the name of a column. This column name may change between recipes, so you can create a macro input to capture the column name, which is the **input value** for the macro input. A macro input can be referenced one or more times in your macro steps.

For more information, see *Overview of Macros*.

Create Macro

Steps:

1. In the Recipe panel, select the step or steps to include in your macro.

NOTE: Source steps from your recipe do not have to be consecutive. In the macro, steps are listed in the order in which they appear in the recipe.

2. From the recipe toolbar context menu, select **Create or replace macro**.

NOTE: The dialog name and options vary based on the selection of create or replace macros.

3. From the drop-down, select **Create a macro**. Enter a Name and an optional Description.

NOTE: The Name of the macro appears in the Trifacta application. Please verify that the Name is unique.

4. Click **Next**.
5. In the Create macro dialog, you can review the selected steps and the inputs for the macros:



Figure: Create macro inputs

6. For each step in the macro:
 - a. Left column: Select the step.
 - b. Middle column: For the selected step, review the values that were specified for the step in the original recipe.
 - c. Right column: As needed, you can provide values for the currently selected inputs from the middle column. For a selected value, you can choose to create a new input or use an existing input.
7. You can review the macro inputs separately in the Inputs tab. For more information, see "Define macro inputs" below.
8. When you have finished specifying your macro and its inputs, click **Create**.
9. The macro is created.
10. In the recipe location where you created it from, the steps from which you created your macro are replaced with an Apply transformation step that references your macro name.

Define macro inputs

A **macro input** is a variable within the macro whose value can be set to a default or, if needed, modified in each instance of the macro.

When you are specifying a macro, the Trifacta application reviews the steps of the macro to identify the values that can be modified in it. In the middle column of Steps tab:

Value type	Description
Columns  POS_Sales1	Column names are automatically turned into inputs.
	<p>These two values could be turned into macro inputs but are not currently defined as such.</p> <p>IFMISMATCHED is a function name, which cannot be parameterized as inputs.</p>

```
IFMISMATCHED($col, [ ABC strDataType × ],  
# intMismatchReplacementValue × )
```

These two values have been turned into macro inputs.

Create macro inputs:

When you are defining a macro, you can create or modify macro inputs.

Tip: Macro inputs can be created or modified in the Steps or Inputs tabs.

Tip: Column names are always recognized as inputs. They can be modified as needed in each instance of the macro.

Steps:

1. To create a new macro input, select a value that is not highlighted.
2. In the right column, enter a name for this new macro input.
3. Specify its default value, and click **Create**.
4. The macro input is created. In the middle column, the highlighted value has been replaced by the name of the macro input.

To modify a macro input, click the entry in the middle column. Then, specify values as needed in the right column, and click **Save**.

To delete a macro input, select it in the middle column. In the right column, click **Remove**.

NOTE: You cannot delete column names as macro inputs.

Edit Macro

When you edit a macro, you can modify the name, description of the macro, as well as the names for any of its inputs.

Tip: To modify the steps of a macro, you must replace it. See "Replace Macro" below.

Steps:

1. You can use either of the following methods to edit the macro:
 - a. In the Macros page, click **Edit** from the context menu of the macro.
 - b. From the recipe toolbar context menu, select **Edit macro**.
2. In the Edit macro dialog, modify the name and description as needed.
3. Click **Next**.
4. In the Edit Macro dialog, click the Inputs tab.
5. Review the listed inputs:
 - a. To change the name of any input, select it.
 - b. In the right panel, enter a new name and description value for the input. Click **Save**.
6. Repeat the previous step for other macro inputs as needed.
7. To save your modifications to the macro definition, click **Save**.

Convert Macro to Steps

After you have created a macro, you may need to convert an instance of a macro to plain steps in your recipe for any of the following reasons:

- The macro definition is going to be changed, and you do not want this instance of the original macro steps to be affected by that change.
- The macro definition is going to be changed, and you want to use this instance as the basis for the new definition. See "Replace Macro" below.

To convert a macro to steps, select the macro instance in your recipe. Then, select **Convert macro to steps** in the context menu of the recipe toolbar.

NOTE: This operation converts the selected instance of the macro to a set of steps. It does not modify the definition of the macro. If preferred, you can delete the macro from the Macros page, which forces all instances of the macro in the workspace to be automatically converted to steps.

Replace Macro

To modify the steps in your macro, you must perform a replacement of all steps in the current definition.

Replace macro with another macro

You can replace a macro's steps with all of the steps of a macro that you have exported to your desktop.

Tip: This method is useful for publishing changes to a macro from one workspace to other workspaces.

Steps:

1. Export a macro definition to your desktop.
2. In the Macros page, find the macro whose steps you'd like to replace with a macro that you've exported to your desktop. From its context menu, select **Replace**.
3. You may need to remap macro inputs in the imported steps to the existing references. See "Update macro inputs" below.

Replace macro with steps

The following method can be used to replace a macro definition with steps that you have created in a recipe.

Tip: When replacing a macro, you can create new inputs for new steps and reassign inputs from the previous version to the steps that haven't changed.

Please complete the following steps.

Steps:

1. To replace all steps in the macro with new ones:
 - a. Create the steps in a recipe that you wish to use.
 - b. When you are ready to use them to replace a macro, select **Create or replace macro** from the context menu.
2. To modify the steps currently in the macro:
 - a. Open a recipe containing an instance of the macro.
 - b. Select the step that applies the macro. From the context menu, select **Convert macro to steps**.
 - c. All of the macro steps are now listed as individual steps in your recipe.
 - d. Add, remove, or modify steps to define your new macro.

Tip: You may want to remove the comment steps that mark the beginning and ending of the converted macro.

- e. When you are ready to use them to replace the macro, select all of the steps. From the context menu, select **Create or replace macro**.
3. In the Create macro dialog, select **Replace an existing macro** from the drop-down.
4. From the Replace macro dialog, select the existing macro to replace.
5. If you want to save the copy of the existing macro, select the corresponding checkbox.

NOTE: Replacing an existing macro replaces all the macro steps with the steps of the new macro. All instances of the previous definition of the macro now reference the new macro definition. In some cases, you may need to reassign input values on old instances to align with the inputs in the updated macro definition.

6. Define macro inputs:

- a. If the old version of the macro contained inputs, you should review those inputs and reassign them to values in the new macro definition.

NOTE: If you do not reassign the macro inputs from the old definition to the new one, then the values used for those inputs in macro instances created under the old definition are lost. After the replacement version is saved, you must review each instance of the macro to verify that it is working properly.

- b. You can also create new macro inputs that apply to the added or modified steps.
- c. See "Update macro inputs" below.
7. After you have reviewed the input, to replace the macro with the existing inputs, click **Replace**.
- a. If you do not specify a relationship between the existing inputs and the replacement macro's inputs, a warning message is displayed.

NOTE: If you discard and save the changes, then any references to those inputs in the instances of the macro in the previous definition are broken.

- b. Click **Discard** to save the macro.

Update macro inputs

When you are replacing a macro, the macro inputs from the old version are carried over into the new version that you are defining.

NOTE: To preserve the values that are stored in the macro inputs from the old version, you must reassign the old macro input to its corresponding input in the new version. If this reassignment is not completed, the input values specified in the old version are lost, and each existing instance of the macro must be reviewed and updated with new macro input values.

Replace macro ✕

1 input from the original macro has been carried over. This input can be found in the inputs tab. Reusing this input will preserve the values where this macro is being used. ✕

Steps
Inputs

1 Replace text or patterns
 1 input

Step 1

Replace text or patterns

Column

email1

Find

{alpha}*

Start search after

empty

Stop search before

empty

Ignore case

True

Match all occurrences

False

Column input

Use existing input ▼

Input

phone_number1 ✕ ▼

Cancel

Save

Cancel

Replace

Figure: Reuse existing macro inputs

1. For each step in the new macro definition,
 - a. Review the inputs in the middle column.
 - b. If a listed input has a corresponding macro input in the old version, select the input. In the right column, select **Use existing input** from the drop down. Then, select the existing input to reassign to the new one. Click **Save**. The input values from the old macro input are preserved.
 - c. If needed, you can create new macro inputs from values in the middle column. See "Define macro inputs" above.
2. Repeat the above steps for each input.

Inspect Macro

When you inspect a macro definition, you review the steps that comprise the macro.

1. You can use either of the following methods to inspect the macro:
 - a. In the Macros page, click **Inspect** from the context menu of the macro.
 - b. From the recipe toolbar context menu, select **Inspect macro**.
2. The steps of the macro are displayed in raw Wrangle .

Tip: You can see the raw Wrangle for your macros in the Macros page in the Library.

Apply Macro

You can use macros that you have created in other recipe locations. See *Apply a Macro*.

Manage Macros

You can manage macros through the Library page.

Apply a Macro

Contents:

- *Insert in Recipe*
 - *Modify a Macro Instance*
 - *Replace the macro definition*
-

After you have created a macro, you can apply it into any of your recipes.

- **Macros** are user-defined sequences of recipe steps that can be referenced independently and parameterized as needed.
- For more information, see *Overview of Macros*.

Insert in Recipe

Steps:

1. Through Flow View, edit the recipe into which you are inserting the macro.
2. In the Recipe panel, click the recipe cursor to the location where you are inserting it.
3. In the Transformer toolbar, click the Macros icon.

Tip: In the Search panel in the Transform Builder, you can search for `Macro` and then select the macro to use.

4. Search for and select the macro to insert. The macro is displayed in the Transform Builder.
5. Specify any macro input values required for the macro.

NOTE: Macro input values must be literal values. Use of flow parameters or metadata references is not supported.

6. To add the macro to the recipe, click **Add**.
7. The macro is added as an **Apply** step.

Modify a Macro Instance

After a macro has been added to your recipe, the following options are available in the **Apply** step's context menu:

- **Inspect macro:** Click to see the definition of the macro. Definition is displayed in Wrangle .
- **Convert macro to steps:** Convert the instance of the macro to a set of static steps.

NOTE: This option converts the instance of the macro. The macro still exists.

Replace the macro definition

If you want to modify the steps in your macro, please do the following:

1. Convert the macro to steps.
2. Perform your modifications to the steps. You can add or remove steps, too.
3. Select all of the steps that are to be used in the new version of the macro.

4. From the context menu, select **Create or replace macro**.

Export Macro

As needed, you can export a macro from Trifacta®. An exported macro is stored in a JSON file that contains all of the information required to use the macro in any instance of the product.

NOTE: Only the creator of a macro can export it.

Exported macros can be imported into the same system or different systems. Macro export is useful for:

- Backups of work in progress

You cannot import macros into an earlier release of the product.

- Archiving of completed development work
- Migrating macros from one instance to another

Export

Steps:

1. From the left navigation bar, select **Library**.
2. In the Library page, click **Macros**.
3. In the Macro page, locate the macro that you wish to export. In its context menu, select **Export**.
4. The JSON file is downloaded to the default download location on your local desktop.

When you import a macro, you import this JSON file.

Export via API

You can export macro definitions using the APIs.

Tip: This method is useful for publishing macro definitions across all deployments in your organization.

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/getMacroPackage>

Import Macro

Contents:

- *Limitations*
 - *Import*
 - *Import via API*
-

A macro that has been exported from the Trifacta® can be imported back into the product.

A **macro** is a reusable set of steps that are specified from within a recipe. For more information, see *Overview of Macros*.

Limitations

You cannot import macros that were exported from a later release of the product.

- You cannot modify the macro definition JSON file outside of Trifacta.
- If you are importing a flow into a Production instance of the platform, any macros referenced in the imported flow are expanded into their original steps in the recipes where they are referenced.
 - Macros cannot be imported, referenced, or viewed directly in a Production instance.
 - A Production instance is available only if you have enabled the Deployment Manager. For more information, see *Overview of Deployment Manager*.

Import

Tip: If you re-import a macro into the same instance that still contains the source macro, the imported version is named the same as the source and automatically versioned for you.

Steps:

1. Export the macro from the source system. See *Export Macro*.
2. Login to the import system, if needed.
3. In the left nav bar, click **Library**.
4. In the Library page, click **Macros**.
5. In the Macros page, click **Import Macro**.
6. Select the JSON file containing the exported macro.

Tip: You can import multiple macros at the same time. Select each JSON file in the dialog box that you wish to import. Press **CTRL/CMD** + click or **SHIFT** + click to select multiple macros for import.

7. Click **Open**.

The macro is imported and available in the Macros page.

To use an imported macro, enter `macro` in the Search panel in the Transform Builder. Select your macro and modify any macro inputs. For more information, see *Apply a Macro*.

Import via API

If you have exported your macro using the APIs, you can import it into a new environment. For more information, see

[*https://api.trifacta.com/ee/es.t/index.html#operation/importMacroPackage*](https://api.trifacta.com/ee/es.t/index.html#operation/importMacroPackage)

Create Flow Parameter

Contents:

- *Limitations*
 - *Limitations on usage*
 - *Create Parameter*
 - *Parameter Names*
 - *Apply Parameter Override*
 - *Override Evaluation*
 - *Use Parameter*
 - *Examples*
 - *Example - String parameter*
 - *Example - parameter with multiple values*
 - *Example - Date parameter*
 - *Apply Parameter Override via API*
-

At the flow level, you can define flow parameters to reference in your recipes. A **flow parameter** is a variable that is assigned a String value.

NOTE: Flow parameters apply to recipe steps only.

- To flow parameters and parameters of other types, you can apply override values at the flow level through the same interface. Details are below.
- For more information on flow parameters, see *Overview of Parameterization*.

Limitations

- Flow parameters are of String data type.

Tip: You can wrap flow parameter references in your transformations with one of the `PARSE` functions. See "Examples" below.

- Flow parameters are converted to constants in macros. Use of the macro in other recipes results in the constant value being applied.

Limitations on usage

A flow parameter cannot be used in the following transformation steps or fields.

Transformations:

- Rename columns: Cannot use a flow parameter as a new column name.

Transformation fields:

- The `as` clause when creating a New formula transformation.

Create Parameter

Steps:

1. Open the flow where you wish to apply the flow parameter.
2. From the Flow View context menu, select **Parameters**.
3. In the Manage Parameters dialog, click the Parameters tab.
4. Click **Add parameter**.
5. Enter a Name for your parameter.

NOTE: Name values are case-sensitive. After saving a flow parameter, its name cannot be changed.

6. Enter a default value for this parameter.

NOTE: Input Values are evaluated as String type.

7. Click **Save**.

The parameter is available for use in any recipe in your flow. See "Use Parameter."

Parameter Names

Parameter names can contain alphanumeric characters and spaces. In the following table, you can see how parameter names must be referenced in recipe steps.

Parameter name	Valid references	Notes
paramRegion	<pre>\$paramRegion \${paramRegion}</pre>	Both references are valid.
param Region	<pre>\${paramRegion}</pre>	<p>NOTE: If the parameter name contains a space, the curly brackets are required. As a matter of habit, you might want to use the curly brackets for all parameter references. This syntax also helps to distinguish your named parameters from metadata references, which are fixed. See <i>Source Metadata References</i>.</p>

Apply Parameter Override

NOTE: Parameter overrides that were defined in a pre-Release 7.1 version of the software now appear in the Overrides tab.

You can apply overrides to all parameter types, including flow parameters, at the flow level. An overridden value applies to all references of the parameter within the flow.

NOTE: You can apply override values for any parameter of any type that is referenced in the flow: dataset parameters, flow parameters, and object parameters.

- **Upstream parameter values:** Parameter values can be inherited from upstream recipes and datasets.

NOTE: Override values applied in a downstream flow are applied to the upstream flow when its objects are invoked for purposes of generating data for use in the downstream flow.

- **Downstream parameter values:** Downstream flows receive parameter values, default or overridden, from upstream flows. These values can be overridden at the flow level.

Steps:

1. Open the flow where you wish to apply the flow parameter.
2. From the Flow View context menu, select **Manage parameters.....**
3. In the Manage Parameters dialog, click the Overrides tab.
4. Click **Add override**.
5. Select the parameter to override from the drop-down list.
6. Set the override value for this flow. Click **Save**.
7. Click **Save**.

This override value is applied to all references to the parameter in the flow.

Tip: Through Flow View, overrides can also be applied to the recipe parameters that are included when flow tasks are executed as part of a plan.

Override Evaluation

Override values can be applied in multiple locations. Parameter values are evaluated in the following order of precedence (highest to lowest):

1. Overrides at run-time in the Run Job page.
2. Overrides at the flow level.
3. Default values for the flow.
4. Inherited values from upstream flows.

For more information, see *Overview of Parameterization*.

Use Parameter

In your recipe step, you can add references to your flow parameter in the following format:

```
${MyRecipeParameter}
```

In a recipe, flow parameters can be applied to:

- Function parameters
- Replacements for String values

Examples

Below are examples of how to use flow parameters.

NOTE: When a parameter value is displayed in a column, the column type in the data grid may be correctly inferring the type to your desired data type. However, the underlying type is still String type. To convert the underlying type, you must use one of the `PARSE` functions on your String values.

Example - String parameter

In this example, data is segmented by time zone. You must create a parameter to capture the following U.S. time zones, which must be specified explicitly:

```
'Hawaii'  
'Alaska'  
'Pacific'  
'Mountain'  
'Central'  
'Eastern'
```

In your flow, you create the following flow parameter:

Setting	Value	Notes
Name	paramTimezone	Tip: It's a good habit to specify named variables in an identifiable way. By adding the <code>param</code> prefix, you identify references to it as a parameter. If you change the name to <code>param-recipeTimezone</code> or similar to distinguish it as a flow parameter, then overrides specified at the flow level do not apply to any other parameter types that are performing the same function in the data.
Value	##UNSPECIFIED##	Since this value must be specified explicitly, you set this value as the default value. If this value appears in the generated output, then the flow parameter was not specified when the job was run. NOTE: Before you begin working with this parameter in your dataset, you should consider setting an override for it to a valid value.

In the following transformation, the parameter value is inserted into a new column, `paramTZ` in your dataset:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>\${paramTimezone}</code>
Parameter: New column name	<code>'paramTZ'</code>

You can also use the parameter as an input to a function. In the following example, the `paramTimezone` parameter is merged with the values in the `Store_Nbr` to compute primary key `storeId` field:

NOTE: You cannot use the Merge transformation column for the following transformation, since it requires named columns as inputs.

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>merge([\$paramTimezone,Store_Nbr], '-')</code>
Parameter: New column name	<code>'storeId'</code>

Example - parameter with multiple values

Suppose you wish to create a flow parameter that contains multiple values. Typically, you must track these values through an array, such as the following containing a set of colors:

```
["red","white","blue","black"]
```

Flow parameters that are literals are String values only. As a workaround, you can define the above as a Pattern .

Setting	Value	Notes
Name	myColors	
Value	<code>`red white blue black`</code>	Note how the value is specified using backticks (`), which are used to indicate a Pattern . The vertical bars are delimiters to separate the values, when they are processed within the application.

Within your recipe, you can test for the presence of a parameter value. In the following transformation, a value of true is set in the new column isBlue if the value of \$myColors is blue:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>MATCHES([blue], \$myColors, true)</code>
Parameter: New column name	'isBlue'

Example - Integer parameter

Instead of segmenting the data by named time zone values, suppose your data is segmented by regions, which are numeric in number. Your flow parameter definition could look like the following:

Setting	Value	Notes
Name	paramRegionId	Note the more appropriate name.
Value	0	In this case, there is no region identifier value 0. You choose to set the default to a value that is valid for the target data type (Integer) but is invalid for the scope of the data itself.

To use this flow parameter as an integer, you must reference it wrapped in the PARSEINT function, which evaluates the input value against the Integer data type:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	<code>PARSEINT({paramRegionId})</code>
Parameter: New column name	paramRegionId

In the column histogram for the paramRegionId column, you can verify that the value 0 is present. Set an override outside at the flow level to insert a different value in the column.

Example - Date parameter

Suppose you need to be able to pass a date into the execution of a recipe. If no date is passed in, then the current time is used. The variable is declared as follows:

Instead of segmenting the data by named time zone values, suppose your data is segmented by regions, which are numeric in number. Your flow parameter definition could look like the following:

Setting	Value	Notes
Name	paramDate	Note the more appropriate name.
Value		<p>In this case, the value is left empty to be overridden as needed in the application with the current timestamp.</p> <p>You should decide on the expected values for this parameter, as you must apply them to:</p> <ul style="list-style-type: none"> Parameter overrides Recipe steps (e.g PARSEDATE function parameters) <p>It may be easier to insert the format string here as the default value. For example:</p> <div>yyyy-mm-dd HH:MM:SS</div>

You can use the following to insert the parameter value into your dataset. Note that the value is initially inserted as a String value, so the PARSEDATE function is used as a wrapper:

Transformation Name	New formula
Parameter: Formula type	Single row formula
Parameter: Formula	PARSEDATE(\${paramDate}, ['yyyy-mm-dd HH:MM:SS'])
Parameter: New column name	paramDate

If the inserted value is empty or null, you can insert the current timestamp:

Tip: You could also overwrite invalid values in the following manner. However, that may mask problems with your inserted values.

Transformation Name	Edit column with formula
Parameter: Columns	execDate
Parameter: Formula	IF((execDate == '') ISNULL(execDate), NOW('UTC'), execDate)

In the above, the value in execDate is tested to see if it is either:

- empty
- null

If so, the output of the NOW function is written. By default, this function returns the timestamp value at UTC time.

If there is a valid value, then it is written back to the column.

You can use the following to extract the time value from the parsed date param:

Transformation Name	New formula
Parameter: Formula type	Single row formula

Parameter: Formula	DATEFORMAT(execDate, 'HH:MM:SS')
Parameter: New column name	Time

Since this value is not the parameter value specifically, the column name was listed simply as `Time`.

Apply Parameter Override via API

When you run a job via the APIs, you can apply parameter overrides to the following parameter types:

- dataset parameters
- output parameters
- flow parameters

For more information, see *API Workflow - Run Job*.

Flag for Review

Contents:

- *Enable*
 - *Limitations*
 - *Flag for Review*
 - *Context menu*
 - *Mark as reviewed*
 - *Mark as pending review*
 - *Rename review step*
 - *Unflag for review*
-

As needed, you can flag recipe steps for review in the recipe panel. You can use flags to set up checkpoints in your recipes, which enable flow users to evaluate the data, provide inputs, and sign off before jobs are executed based on the recipe.

Examples:

- You could flag steps in recipes within flows that other users may copy. These flags and their related descriptions can be used to provide guidelines for how to implement the step in any copy.
- Among your collaborators, you may have experts in specific aspects of the data. You can flag steps for their review, perhaps even including their name in the description value for easy review.

When you flag a step for review:

- The step is marked for review in the recipe panel.

NOTE: A flagged step must be reviewed before you can edit later steps in the recipe or run jobs based on the recipe.

- In Flow View, the recipe icon is highlighted with a warning.
- The Flow View page header summarizes the total number of flagged steps and recipes that are pending for review.
- If you have created a reference dataset, it is also highlighted with a warning wherever it is used.

Enable

This feature may need to be enabled in your environment.

Steps:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
2. In the Admin Settings page, set the following to `true`:

```
"feature.haltExecution.enabled": true,
```

3. Save your changes and restart the platform.

Limitations

- When a step is flagged for review, all downstream steps are disabled.

- Steps must be reviewed in descending, top-to-bottom order.
- You cannot run the job until all flagged steps are reviewed.
- Flags can be applied and cleared one at a time.
- Undo / Redo options are not applicable to flagged steps.
- Flag for Review is not supported for the following features: Join, Union, Standardize, Transform by Example, or Macros.

Tip: You can flag the following step or add a comment step and flag the comment if you want to call attention for these transforms.

Flag for Review

Steps:

1. In the Recipe panel, select the step to flag.
2. From the Recipe toolbar context menu, select **Flag for review**.
3. In the Flag for review dialog:
 - a. (optional) Enter a name or title for the flag.
 - b. (optional) Enter a description.
 - c. Click **Flag**.
4. A warning icon is displayed over the selected step. You can hover the warning icon to read the description.
5. The step has been marked for review.

NOTE: When one or more steps has been flagged in your recipe, the **New Step** and **Run** options are disabled.

Context menu

The following menu options are available in the context menu for flagged steps.

Mark as reviewed

From the Recipe toolbar context menu, select the required step and select **Mark as reviewed**. A tick mark is displayed over the reviewed step, indicating that the step has been cleared.

NOTE: If there are no additional flagged steps, you can add new recipe steps or run jobs for your recipe.

Mark as pending review

Revert the Mark as reviewed flag. The tick mark is replaced by the warning icon over the selected step.

NOTE: You can toggle between **Mark as reviewed** and **Mark as pending review** options to mark the review as complete or to mark the step as pending review.

Rename review step

Edit the name and description values.

Tip: You can add hyperlinks as part of the Description value.

Unflag for review

Removes the flag for review from the step.

NOTE: The step is now cleared of the flag. If there are no additional flagged steps, you can add new recipe steps or run jobs for your recipe.

Manage Environment Parameters

Contents:

- *Create Environment Parameter*
 - *Use Environment Parameter*
 - *Limitations*
 - *Edit Environment Parameter Value*
 - *Delete Environment Parameter*
 - *Export Environment Parameters*
 - *Import Environment Parameters*
-

You can define parameters that are applicable across the entire project or workspace environment.

An **environment parameter** is a variable of String type defined by an administrator that any user of the environment can reference in their flows and flow-related objects.

NOTE: You must be a project owner or workspace administrator to manage environment parameters.

For more information on parameters, see *Overview of Parameterization*.

Create Environment Parameter

Steps:

1. A project owner or workspace administrator can select **User menu > Admin console > Environment parameters**.
2. In the Environment Parameters page, click **Create**.
3. Specify the parameter:
 - a. **Name:** Enter the display name for the parameter.

NOTE: All environment parameter names are automatically prepended with `env..`

- b. **Default value:** Enter the default value.

NOTE: The default value is stored as a String value.

4. To create another environment parameter, click **Add another**.
5. To save your changes, click **Save**.

Use Environment Parameter

Environment parameters can be referenced in the following locations:

Tip: When specifying a variable, enter `$env` to see the list of available environment parameters.

- Parameterized datasets. See *Create Dataset with Parameters*.

- Datasets created with SQL. See *Create Dataset with SQL*.
- Output paths. See *Create Outputs*.

Limitations

- You cannot use environment parameters in recipes.
- You cannot use environment parameters in plans.
- Environment parameter names must be unique within the project or workspace. You can apply override values to them at runtime.
- You cannot use environment parameters in Deployment Manager. For more information, see *Overview of Deployment Manager*.

Edit Environment Parameter Value

Administrators can change the default value for an environment parameter.

NOTE: Modifying the default value of an environment parameter immediately applies the change across the entire environment. All subsequent job and plan runs are affected.

NOTE: After you have created an environment parameter, you cannot change the name. You must create a new environment parameter.

Steps:

1. In the Environment Parameters page, locate the parameter whose default value you wish to modify.
2. In the More menu, select **Edit value**.
3. Enter a new value, and click **Save**.

Delete Environment Parameter

When you delete an environment parameter, all references to the parameter are converted to empty string values. Job executions can fail, and recipe steps can break.

Tip: If you delete an environment parameter and then recreate it using the exact same name, references to the parameter are updated with the new default value, which replaces the empty string value for the deleted parameter.

Steps:

1. In the Environment Parameters page, locate the parameter whose default value you wish to modify.
2. In the More menu, select **Delete**.

Export Environment Parameters

You can export the environment parameters from your project or workspace.

NOTE: All environment parameters are exported at the same time into a ZIP file. Do not modify this file outside of the Trifacta application.

Steps:

1. In the Environment Parameters page, select **More menu > Export**.
2. The ZIP file is downloaded to your local desktop.

Import Environment Parameters

If you have exported a set of environment parameters, you can import them into another workspace or project.

NOTE: If an environment parameter that you are importing has a name that conflicts with an environment parameter that already exists, you must either rename the imported parameter or delete it from the import set.

Steps:

1. In the Environment Parameters page, select **More menu > Import**.
2. Select or drag-and-drop the ZIP file. Click **Import**.

NOTE: Select the ZIP file or its embedded JSON5 file for import.

3. The Import environment parameters dialog is displayed:

Name	Default value	
env. region	01	
env. bucket_name	myBucket	
env. name	Prod	
env. admin_contact	admin@example.com	

[Add another](#)

Cancel Save

Figure: Import environment parameters dialog

4. Review each environment variable and its assigned value from the import package:
 - a. Modify values as needed.
 - b. To delete a parameter from the import process, click the Trash icon.
 - c. To add another parameter as part of the import package, click **Add another**.
5. To save your changes and complete the import, click **Save**.
6. The environment parameters and your modifications to them are imported.

Operationalization Tasks

These topics cover how to operationalize your Trifacta® flows for use in your enterprise data pipelines.

Operationalization includes all tasks that occur after principle flow development has completed. After you have your flow working and delivering useful data, operationalization can include:

- **Scheduling.** For more information, see *Schedule a Job*.
- **SQL scripting.** Before or after job execution, you can configure your job to execute a specified SQL script. For more information, see *Create Output SQL Scripts*.
- **Macros.** You can build reusable sets of recipe steps, which can be used in other flows and workspaces for consistency. See *Create or Replace Macro*.
- **Flow Webhooks.** After a flow has executed, you can configure a webhook task to deliver messages about the flow execution to other systems. See below.
- **Plans.** A plan is a sequence of tasks that can be executed based on logic and can be scheduled. See below.

Create Flow Webhook Task

Contents:

- *Limitations*
 - *Prerequisites*
 - *Requirements for receiving application*
 - *Steps*
 - *Flow metadata references in body*
 - *Examples*
 - *Run another job*
 - *Slack channel message*
 - *Verify Webhook Signatures*
 - *Webhook Signature Header*
 - *Check Application Tools*
 - *Process Signed Requests*
-

You can send webhook messages to third-party applications based on the results of job executions in your flow.

- A **webhook task** is a callback message between Trifacta® and another application. They are typically delivered using JSON over HTTP and can be interpreted by the receiving application to take action.

NOTE: Your receiving application may require that you whitelist the host and port number or IP address of the platform. Please refer to the documentation for your application.

- A webhook task is defined at the flow level, although an individual webhook task can be restricted to specific outputs. It is shared between ad-hoc and scheduled executions.
- This capability may need to be enabled in your environment. For more information, see *Workspace Settings Page*.
- Additional configuration may be required. See *Configure Webhooks*.

For more information on how to orchestrate execution of your flows, see *Overview of Operationalization*.

Limitations

- Custom security certificates cannot be used.
- HTTP-based requests have a 30-second timeout limit.
- Webhook tasks are not included when a flow is copied. They are available to collaborators for review, editing, and execution, when a flow is shared.

Tip: You can export and import the flow, which includes the webhook task definition.

- You can create a maximum of 50 webhooks per flow.

NOTE: Administrators can change this limit as needed. For *Configure Webhooks*.

Prerequisites

NOTE: It's possible that webhook requests can be submitted back to the platform to execute API tasks within the platform. However, there are security concerns. Additional configuration is required. For more information, see *Configure Webhooks*.

Requirements for receiving application

To send webhooks to a target application, the application must be configured to receive the webhook:

- Incoming webhooks must be enabled.

NOTE: Your receiving application may require that you whitelist the host and port number or IP address of the platform. Please refer to the documentation for your application.

- You must acquire the URL of the endpoint to which to send the webhook request.
- You must acquire any HTTP headers that must be inserted with each webhook request.
- If the request must be signed, additional configuration is required. Details are below.

Steps

1. Open your flow in Flow View. From the flow context menu, select **Webhooks**.
2. In the right panel, select **Create webhook task**.
3. Set the following parameters:

Parameter	Description
Name	User-visible name of the task.
Url	URL where the webhook message is received by the other application.
Trigger event	Select the event that triggers the message.
Trigger object	Select the object or objects that can trigger the message: Any job executed in this flow - Any scheduled or ad-hoc job triggers the message Only specific objects - Select the output or outputs whose success or failure triggers the message
Headers	<p>Insert HTTP content headers as key-value pairs. For example, if your body is in JSON format, you should include the following header:</p> <pre>key: Content-Type value: application/json</pre> <p>NOTE: You may be required to submit an authentication token as the value for the <code>Authorization</code> key.</p> <p>Please refer to the documentation for your receiving application about the required headers.</p>
Body	(POST, PUT, or PATCH methods only) The body of the request submitted to the receiving application. In the body, you can use the following references: <code>jobId</code> - the internal identifier for the jobGroup that was executed. <code>jobStatus</code> - the status for the job after execution. For more information, see <i>Jobs Page</i> . You can apply metadata references to the flow in the Body text. See below for examples.
Method	Select the HTTP method to use to deliver the message. The appropriate method depends on the receiving application. Most use cases require the POST method.
Secret key	

	<p>(Optional) A secret key can be used to verify the webhook payload. This secret value must be inserted in this location, and it must be included as part of the code used to process the requests in the receiving application. Insert the secret value here as a string without quotes.</p> <p>For more information on how this secret key is used to generate a signature, see Verify Webhook Signatures below.</p>
Validate SSL certificate	<p>When set to <code>true</code>, HTTPS (SSL) communications are verified to be using a valid certificate before transmission.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>NOTE: If you must send a request to an endpoint that has an expired/invalid certificate, you must disable SSL verification.</p> </div>
Retry on failure	<p>If the returned status code is outside of the 200-299 range, then the webhook is considered to have failed. When this option is enabled, the request is retried. The number of retry attempts can be configured. See Configure Webhooks.</p>

4. To test the connection, click **Test**. A success message is displayed.
5. To add the webhook task to the flow, click **Create**.
6. When the job is executed:
 - a. Depending on the outcome, the webhook task is executed through the other application.
 - b. The webhook is listed in the Job Details page.

Flow metadata references in body

In the body of your webhook, you can use the following references:

Reference	Description
<code>\$jobId</code>	Internal identifier to the job in the platform.
<code>\$jobStatus</code>	The current status of the webhook job. For more information on job status messages, see Jobs Page .

Examples

Run another job

You can create a webhook task to run another job on the successful execution of this one.

Tip: Use this method to create conditional sequences of job executions.

As needed, you can specify webhook overrides as part of a launching a job via API. For more information, see [API Workflow - Run Job](#).

Prerequisites

NOTE: For this example, the platform must be whitelisted to receive webhooks from itself. Additional configuration is required. For more information, see [Configure Webhooks](#).

You must acquire the recipe identifier for the next job to execute.

1. Open the flow containing the next recipe.
2. In Flow View, click the recipe whose outputs you wish to generate.
3. Review the URL for the recipe object. In the example below, the recipe Id value is 4:

`http://www.example.com:3005/flows/1?recipe=4&tab=recipe`

4. Retain this value for below.

Define the flow webhook task

Parameter	Description
Name	This name appears in the Trifacta application only.
Url	<p>Specify the URL as follows, replacing the example values with your own:</p> <pre>http://www.example.com:3005/v4/jobGroups/</pre>
Trigger event	Select <code>Job success</code> .
Trigger object	Select the <code>any</code> option to execute all jobs in the target flow, or you can specify individual jobs to execute.
Headers	<p>Insert the following two headers:</p> <pre>key: Content-Type value: application/json</pre> <pre>key: Authorization value: Bearer <paste your access token here></pre> <p>NOTE: The token value must be preceded by the string: <code>Bearer</code>.</p>
Body	<p>In the body, insert the recipe Id for the value for <code>wrangledDataset</code>, which is the internal platform term for recipe:</p> <pre>{ "wrangledDataset": { "id": 4 } }</pre>
Method	Select the <code>POST</code> method.

Verify

1. Run the job for which the webhook was created.
2. When the job successfully completes, open the flow containing the other job to execute.
3. When you select the target recipe, a new job should be queued, in-progress, or completed.

Slack channel message

You can create a webhook task to deliver a text message to a Slack channel of your choice.

Prerequisites

Set up your Slack installation to receive webhook messages:

1. If needed, create a Slack channel to receive your messages.
2. Create an app.
3. Activate incoming webhook messages for your app.
4. Specify the channel to receive your incoming webhook messages.
5. Copy the URL for the incoming webhook from the cURL statement.

Define the flow webhook task

Parameter	Description
Name	This name appears in the Trifacta application only.
Url	Paste the URL that you copied from Slack.
Headers	Copy the content headers from the Slack cURL command: <pre>key: Content-Type value: application/json</pre>
Body	<pre>{"text":"Job \$jobId has completed. Status: \$jobStatus."}</pre>
Method	Select the POST method.

Verify

1. Click **Test** to validate that this webhook task will work.
2. Run a job:
 - a. Check the Slack channel for a message.
 - b. Check the Webhook tab in the Job Details page.

Verify Webhook Signatures

Depending on the target application, implementing Webhook signature verification may require developer skills.

Optionally, you can configure the platform to sign the Webhook requests sent for a flow. Signed requests guarantee that the requests are sent from the platform, instead of a third party.

Below, you can review how the signature is created, so that you can configure the receiving application to properly process the signature and its related request.

Webhook Signature Header

Webhook requests are signed by inserting the `X-Webhook-Signature` header in the request. These signatures are in the following form:

```
X-Webhook-Signature: t=<timestamp>,sha256=<signature>
```

where:

- `<timestamp>` - Timestamp when the signature was sent. Value is in UNIX time.
- `<signature>` - SHA256 signature. The platform generates this signature using a hash-based message authentication code (HMAC) with SHA-256.

More information on these values is available below.

Example:

```
X-Webhook-Signature: t=1568818215724,sha256=55fa71b2e391cd3ccba8413fb51ad16984a38edb3cccf81f381c4b8197ee07a
```

Check Application Tools

Depending on the application, you may need to complete one of the following sets of tasks to verify the Webhook signatures:

NOTE: You may need to whitelist the platform in your application. See the application's documentation for details.

You may be required to create some custom coding for your application. Below, you can review details on how to do so, including a JavaScript example.

Process Signed Requests

Timestamp

The timestamp value (`t=<timestamp>`) appears at the beginning of the header value to prevent replay attacks, where an attacker could intercept a valid payload and its signature and re-transmit them.

- To avoid such attacks, a timestamp is included in the signature header and is also embedded as part of the signed payload.
- Since the timestamp is part of the signed payload, an attacker cannot change the timestamp value without invalidating the signature.
 - If the signature is valid but the timestamp is too old, you can then choose to reject the request.
 - For example, if you receive a request with a timestamp that corresponds to a date from one hour ago, you should probably reject the request.
- For more information on replay attacks, see https://en.wikipedia.org/wiki/Replay_attack.

Signature

The Webhook signature includes as part of its hashed value:

- The secret key (entered above)
- The timestamp value
- Request data:
 - (POST/PUT/PATCH) - the body of the request
 - (GET/DELETE) - URL of the request

Step 1 - Extract the timestamp and signatures

Split the `X-Webhook-Signature` header:

1. Split values using the `,` character as a separator.
2. Split each of the parts using the `=` character.
3. Extract the values for the timestamp and signature. From the above example:
 - a. timestamp: 1568818215724
 - b. signature: 55fa71b2e391cd3ccba8413fb51ad16984a38edb3cccf81f381c4b8197ee07a

Step 2 - Create the expected signature

In the receiving application, you can recompute the signature to verify that the request was sent from the platform.

1. Concatenate the timestamp, the dot character `.` and the request body (POST/PUT/PATCH methods) or the url (GET/DELETE methods).
2. Suppose the above example is the signature for a `POST` request, and the request body is `test`. The concatenated value is the following:

```
1568818215724.test
```

3. You can now compute the HMAC authentication code in your receiving application. In the following JavaScript example, the secret key value is `mySecret`:

```
const crypto = require('crypto');

const message = '1568818215724.test'; // as defined above

const hmac = crypto.createHmac('sha256', 'mySecret');
hmac.update(message)
const expectedSignature = hmac.digest('hex');
```

Step 3 - Compare the signatures

The value returned by your code and the value included as the signature in the `X-Webhook-Signature` header should be compared:

- If the values do not match, reject the request.
- If the values do match, compute the difference between the current timestamp and the timestamp in the header. If the difference is outside of your permitted limit, reject the request.
- Otherwise, process the request normally in your application.

Create a Plan

Contents:

- *Before You Begin*
 - *Workflow*
 - *Create Plan*
 - *Add Plan Schedule*
 - *Add Task*
 - *Apply Parameter Overrides*
 - *Create Branching Plan*
 - *Add Task Execution Rule*
 - *Add Parallel Task*
 - *Example - Success or Failure Tasks in a Plan*
 - *Test Plan*
 - *Monitor Plan Runs*
-

A **plan** is a sequence of tasks that are executed manually or based on a schedule. Plans can be used to automate the execution of multiple related tasks, such as all of the outputs generated from a set of multiple related flows.

- When a plan is triggered:
 - A snapshot of the objects in the plan is capture. This snapshot defines the set of tasks that are executed as part of a plan run.

NOTE: A snapshot does not capture the objects underlying the tasks. After a snapshot is taken, subsequent changes to the underlying flows could impact the outcome of the flow tasks when they are later executed during the plan run.

- The set of tasks in the plan are triggered in the order listed in the plan.
- All of the dependencies for any task are also executed. For example, if a flow output requires the outputs from another upstream flow, then that flow's output is also generated.
- If one task fails to execute, the other tasks are not executed.
- For more information on plans, see *Overview of Operationalization*.

Before You Begin

Before you begin, please verify the following:

- You have access to all of the flows that you wish to use in your plan.
- For each flow in your plan:
 - All of the recipes whose results you wish to generate have output objects associated with them.
 - Each output object has at least one of the following that has been created for it:
 - file-based output
 - table-based output

NOTE: In a flow, all recipes that you wish to have executed by the corresponding task must have a defined output object. For each output object, you must create at least one write file or table settings definition. During plan runs, these objects are not validated, and missing outputs are ignored.

Workflow

NOTE: Parameter values are applied to a plan, but you cannot apply parameter overrides to the plan. You can apply flow parameter overrides on individual flows in Flow View. These overrides are applied at the time of plan execution.

Workflow steps:

1. Identify the tasks that you wish to execute.

NOTE: You must have access to any flows that you wish to execute.

2. Add a task.
 - a. In Plan View, click the Plus icon at the bottom of your plan.
 - b. Specify the task to execute.
3. Repeat the previous step to add additional tasks as needed.

Tip: You can insert tasks between other tasks. Use the Plus icon between two plan objects.

4. To test your plan, click **Run now**. The plan is immediately executed.
5. Edit the plan and repeat the above steps until the plan is ready for production runs.

Tip: While a plan is in development, you may wish to disable its schedule, which prevents execution according to the schedule. You can still run test executions using the Run Now button.

6. Create the schedule for the plan.
 - a. In the context menu for the plan, select **Schedule**.
 - b. Specify one or more triggers for the schedule. When a trigger occurs, the plan is queued for execution.
7. When ready, the plan runs at the time scheduled in the trigger.

Create Plan

To begin, you must create a plan object.

Steps:

1. From the left nav bar, click the Plans icon.
2. The Plans page is displayed.
3. In the Plans page, click **Create**. A new plan with the name `Untitled - x` is created, where x is a number.
4. Click the `Untitled - x` to enter a plan name and description.
5. Your plan is saved and displayed in Plan View.

In Plan View, you create the objects that are part of your plan. These include:

- **Plan Schedule:** A schedule is composed of one or more triggers that determine when the plan is executed.
 - **Trigger:** Scheduling object that determines the conditions under which the plan is executed.
 - A schedule can contain one or more triggers.
- **Task:** An action that is executed when triggered.
 - You can build a sequence of one or more tasks in your plan.

Add Plan Schedule

You can add a schedule object to specify the triggers when the plan is to be executed.

NOTE: A plan's schedule cannot be executed until its schedule has been enabled. If a plan has a disabled schedule, you can still execute it via the Run Now button.

Steps:

1. When you first open Plan View, you should see an empty plan:



Figure: Plan View - empty plan

2. To begin, do one of the following:
 - a. From the Plan View context menu, click **Schedule**.
 - b. Click the big circle.
3. In the right context panel, click **Create schedule**.

4. In the Add Trigger panel, you can specify the triggers when the plan is executed. You can specify one or more triggers:

The screenshot shows the 'Trigger' configuration panel. It includes a title bar with a back arrow, the text 'Trigger', and a close button. The panel is divided into two sections for adding triggers. The first section shows a trigger set to 'Weekly' frequency, 'Sunday' day, at '12:15 AM' in the 'America/Los_Angeles' timezone. The second section shows a trigger set to 'Weekly' frequency, 'Wednesday' day, at '12:15 AM' in the 'America/Los_Angeles' timezone. A blue link 'Add another trigger' is positioned between the two sections. At the bottom right, there are 'Cancel' and 'Save' buttons.

Figure: Add trigger(s)

5. For each trigger:
- Timezone: Specify the timezone that applies to the scheduled time. For more information on timezones, see *Supported Time Zone Values*.
 - Frequency: You can specify the frequency of when the schedule is triggered.
 - In each trigger, you can specify multiple On values (e.g. Same time on Sunday and Monday).
 - As needed, you can specify the On value using a modified form of cron job syntax. For more information, see *cron Schedule Syntax Reference*.
6. To add more triggers, click **Add another trigger** and specify it.
- To delete a trigger, click the X next to it.

7. Parameter overrides:

- If the flows in your plan contain parameters, you can apply overrides to the parameter values.
- Overrides provided in this panel are applied only when the trigger is executed.

NOTE: Multiple values are ok for plan parameters, as long as the parameter values do not conflict. If you see a warning icon next to a set of multiple parameter values, then you must fix this conflict in Flow View, or the plan fails to execution.

- c. You can apply overrides through Plan View, too.
8. To save your schedule, click **Save**.
9. In the context panel, you can make changes to your schedule:
 - a. After saving, the schedule is automatically enabled. To disable the schedule, use the slider bar.

NOTE: A plan cannot be executed if the schedule for it has been disabled.

- b. To make changes to the schedule and its triggers, click **Edit**.

Add Task

Based on the schedule's triggers, you can define a sequence of one or more tasks that are executed.

- To add a new task, click the + icon below the trigger. Select the type of task in the right panel.
- To insert a task between two other objects, click the + icon between them.

Add run flow task

A **flow task** executes the recipes that produce the output objects of the flow.

Steps:

1. After you select the flow task type, use the Search bar or browse to select the flow that you wish to add as the task.
2. Select the output or outputs that you wish to generate from the selected flow.
3. Click **Create task**.
4. The task is created and added to the plan.

For more information, see *Plan View for Flow Tasks*.

Add HTTP task

An **HTTP task** is a request sent using HTTP protocol to a target URL, which could be a REST API endpoint.

NOTE: Specifying an HTTP request requires knowledge of the target endpoint and the parameters required for the request. HTTP tasks are considered developer-level objects.

Steps:

1. After you select the HTTP task type, you can specify the task in the context panel.
2. Specify the fields of the request.

Tip: If possible, you should test the HTTP task before you create it. To test for basic connection, you should use the GET method, which just returns relevant information. Some other methods are potentially destructive.

3. Click **Save**.
4. The task is created and added to the plan.

For more information, see *Create HTTP Task*.

Add Slack task

A **Slack task** is a message submitted from the Trifacta application to a specified Slack channel.

Steps:

1. After you select the Slack task type, you can specify the task in the context panel.
2. Specify the fields of the request.
3. Click **Save**.
4. The task is created and added to the plan.

For more information, see *Create Slack Task*.

Add Delete task

A **Delete task** deletes a specified set of files or folders from backend storage.

Steps:

1. After you select the Delete task type, you can specify the task in the context panel.
2. Specify the path to the file or folder to delete. This path must already exist.
 - a. **Location:** If this drop-down is available, select the file-based connection. To explore this connection, click **Browse**.
 - b. Navigate to your preferred destination. Click **Choose**.
3. Click **Save**.
4. The task is created and added to the plan.

For more information, see *Create Delete Task*.

Apply Parameter Overrides

If your plan tasks include flows in which parameters have been defined, you can review and override these parameter values. Overrides are applied when the task is triggered as part of a plan run.

Steps:

1. From the Plan View context menu, select **Parameters**.
2. Review the names, sources, and current values for all of the parameters in your plan.
3. To apply an override, click the Pencil icon and enter a new value. Click **Save**.

Subsequent runs of the plan use this new value as the override for the parameter. For more information, see *Plan View for Flow Tasks*.

Create Branching Plan

In some scenarios, you may need to branch plan execution steps based on the results of a task in the plan. For example, you may need to send separate messages using an HTTP task depending on whether a flow task succeeds or fails in execution. You can create branches in the plan graph by adding task execution rules and parallel nodes, which run based on the success and failure states of your plan runs.

To begin this simple example:

1. Create your first task, which is a flow task in the above example. For more information, see *Add Tasks* above.
2. Complete the following sections.

Add Task Execution Rule

Next, you create the first HTTP task that results from the above task and the execution rule that determines when it runs.

- This task should run based on the successful execution of the flow task.
- A **task execution rule** is a condition that is tested after a flow task has run to determine if the task that is downstream of it is executed as a result. In this case, you create an *On success* rule.

Steps :

1. Click the plus icon below the existing flow task node.
2. Select the HTTP task type and enter information in the required fields. See Add HTTP Tasks above.
3. Click the link connecting the created HTTP task node and its previous task node and select `On success`.

The HTTP task is executed only when the flow task has run successfully.

Add Parallel Task

Next, you can create the HTTP task that runs when the flow task fails.

Steps:

1. Click the plus icon below the existing flow task node and select **Add a parallel node**. A parallel node is added to the plan graph. See Example below.
2. Select the HTTP task type and enter information in the required fields. See Add HTTP Tasks.
3. Click the link connecting the new HTTP task and its previous task node and select `On failure`.

The second HTTP task is executed only when the flow task has failed to execute.

Tip: You can use parallel tasks to create separate paths through a plan when there are no dependencies between the paths.

Example - Success or Failure Tasks in a Plan



Figure: Success and Failure tasks

When the flow tasks complete successfully, the `On success` HTTP task sends a message.

When the task fails, the `On failure` HTTP task delivers a different message.

Test Plan

After you have created the triggers and tasks of your plan, you can perform a test run of the plan.

Steps:

1. To test, click **Run now**.
2. The plan run is queued for execution.

Monitor Plan Runs

1. in the upper-right corner of Plan View, click the Runs link.
2. In the Plan Runs page, you can track the progress of your plan run.
 - a. The most recently triggered plan run is displayed.
 - b. If you have executed multiple runs, you can use the angle brackets next to the timestamp for the run.
3. For tasks in progress, you can click the task to display information in the context panel.
4. To see the details for the plan run, click the Outputs tab. Then, click **Job details**.

You can monitor the progress of your plan runs and review all previous ones in the Plan Runs page.

Create Delete Task

Contents:

- *Limitations*
- *Prerequisites*
- *Steps*
- *Plan metadata references*

You can create plan tasks to delete existing files or folders through connections to which you have access. These tasks are helpful for removing files that were generated as part of intermediate steps in your plan's execution.

- A Delete task is defined as one of the tasks in a plan. For more information, see *Plan View Page*.
- This capability may need to be enabled in your environment. For more information, see *Overview of Operationalization*.

Limitations

- As a safeguard, you are prevented from deleting more than 100 files at a time. The maximum file limit for delete tasks can be modified, if needed. See *Overview of Operationalization*.
- Delete tasks are supported for the following file systems:
 - S3
 - ADLS

Prerequisites

- You must have write access to the connection, bucket, and folder where you wish to delete files.

Steps

1. Open your plan in Plan View. Click a node to create a new task.
2. In the right panel, select **Delete task**.
3. Set the following parameters:

Parameter	Description
Connection	If the drop-down is present, select the connection where the files or folders are located. If the drop-down is present, you can delete files from the backend storage environment only.
Path	<div>Specify the location where you wish to remove files. To navigate the storage environment, click Browse.</div> <div>Tip: You can paste in the Path textbox values that you have copied.</div> <div>Tip: You can insert plan metadata references in the path for tasks that have previously been executed in the plan. Enter \$ to begin exploring available references.</div> <div>You can select entire folders. These folders and files must exist at the time of creating the Delete task.</div>

NOTE: As a safety measure, you are not permitted to delete more than 100 files in a single task.

4. To add the task to the flow, click **Save**.

Plan metadata references

Within the message of your other tasks, you can reference metadata about the plan, including the Delete task. For more information, see *Plan Metadata References*.

Create HTTP Task

Contents:

- *Limitations*
 - *Prerequisites*
 - *Requirements for receiving application*
 - *Steps*
 - *Examples*
 - *Run another job*
 - *Slack channel message*
 - *Plan metadata examples*
 - *Feed metadata inputs to cloud function*
 - *Verify Signatures*
 - *Signature Header*
 - *Check Application Tools*
 - *Process Signed Requests*
-

During the execution of your plan, you can create a task to send HTTP requests to a third-party application endpoint. For example, when a flow task successfully executes, you can send an HTTP message to a designated endpoint.

- An **HTTP task** is a request between Trifacta® and another application. These requests are delivered using over HTTP and can be interpreted by the receiving application to take action.

NOTE: Your receiving application may require that you whitelist the host and port number or IP address of the platform. Please refer to the documentation for your application.

- A HTTP task is defined as one of the tasks in a plan. For more information, see *Plan View Page*.
- This capability may need to be enabled in your environment. For more information, *Overview of Operationalization*.

Limitations

- Custom security certificates cannot be used.
- HTTP-based requests have a 30-second timeout limit.

Prerequisites

NOTE: It's possible that webhook requests can be submitted back to the platform to execute API tasks within the platform. However, there are security concerns. Additional configuration is required. For more information, see *Configure Webhooks*.

Requirements for receiving application

To send an HTTP request to a target application, the application must be configured to receive the request:

- Requests from outside of the application domain must be enabled.

NOTE: Your receiving application may require that you whitelist the host and port number or IP address of the platform. Please refer to the documentation for your application.

- You must acquire the URL of the endpoint to which to send the HTTP request.
- You must acquire any HTTP headers that must be inserted with each HTTP request.
- If the request must be signed, additional configuration is required. Details are below.

Steps

1. Open your plan in Plan View. Click a node to create a new task.
2. In the right panel, select **HTTP task**.
3. Set the following parameters:

Parameter	Description
Name	User-visible name of the task.
Url	URL where the webhook message is received by the other application.
Headers	<p>Insert HTTP content headers as key-value pairs. For example, if your body is in JSON format, you should include the following header:</p> <pre>key: Content-Type value: application/json</pre> <p>NOTE: You may be required to submit an authentication token as the value for the <code>Authorization</code> key.</p> <p>Please refer to the documentation for your receiving application about the required headers.</p>
Body	<p>(POST, PUT, or PATCH methods only) The body of the request submitted to the receiving application.</p> <p>NOTE: If your request does not require a body, please insert <code>{ }</code> here. This is a known issue.</p>
Method	Select the HTTP method to use to deliver the message. The appropriate method depends on the receiving application. Most use cases require the <code>POST</code> method.
Secret key	<p>(Optional) A secret key can be used to verify the webhook payload. This secret value must be inserted in this location, and it must be included as part of the code used to process the requests in the receiving application. Insert the secret value here as a string without quotes.</p> <p>For more information on how this secret key is used to generate a signature, see Verify Webhook Signatures below.</p>
Validate SSL certificate	<p>When set to <code>true</code>, HTTPS (SSL) communications are verified to be using a valid certificate before transmission.</p> <p>NOTE: If you must send a request to an endpoint that has an expired/invalid certificate, you must disable SSL verification.</p>
Retry on failure	<p>If the returned status code is outside of the 200-299 range, then the webhook is considered to have failed. When this option is enabled, the request is retried.</p> <p>When a message is retried, the following header is submitted:</p> <pre>X-Http-Task-Guid</pre>

4. To test the connection, click **Test**. A success message is displayed.
5. To add the task to the flow, click **Save**.

Examples

Run another job

You can create a task to run another job on the successful execution of this one.

Tip: Use this method to create conditional sequences of job executions.

As needed, you can specify task overrides as part of a launching a job via API. For more information, see *API Workflow - Run Job*.

Prerequisites

NOTE: For this example, the platform must be whitelisted to receive requests from itself. Additional configuration is required. For more information, see *Configure Webhooks*.

You must acquire the recipe identifier for the next job to execute.

1. Open the flow containing the next recipe.
2. In Flow View, click the recipe whose outputs you wish to generate.
3. Review the URL for the recipe object. In the example below, the recipe Id value is 4:

```
http://www.example.com:3005/flows/1?recipe=4&tab=recipe
```

4. Retain this value for below.

Define the HTTP task

Parameter	Description
Name	This name appears in the Trifacta application only.
Url	Specify the URL as follows, replacing the example values with your own: <pre>http://www.example.com:3005/v4/jobGroups/</pre>
Headers	Insert the following two headers: <pre>key: Content-Type value: application/json</pre> <pre>key: Authorization value: Bearer <paste your access token here></pre> NOTE: The token value must be preceded by the string: Bearer.
Body	In the body, insert the recipe Id for the value for <code>wrangledDataset</code> , which is the internal platform term for recipe: <pre>{ "wrangledDataset": {</pre>

	<pre> "id": 4 } } </pre>
Method	Select the POST method.

Verify

1. Run the plan for which the HTTP task was created.
2. When the plan successfully completes, open the flow containing the other job to execute.
3. When you select the target recipe, a new job should be queued, in-progress, or completed.

Slack channel message

Tip: Slack tasks are now a supported product feature. For more information, see [Create Slack Task](#).

You can create an HTTP task to deliver a text message to a Slack channel of your choice.

Prerequisites

Set up your Slack installation to receive HTTP messages:

1. If needed, create a Slack channel to receive your messages.
2. Create an app.
3. Activate incoming HTTP messages for your app.
4. Specify the channel to receive your incoming messages.
5. Copy the URL for the incoming HTTP request from the cURL statement.

Define the HTTP task

Parameter	Description
Name	This name appears in the Trifacta application only.
Method	Select the POST method.
Url	Paste the URL that you copied from Slack.
Headers	Copy the content headers from the Slack cURL command: <pre> key: Content-Type value: application/json </pre>
Body	<pre> {"text": "Your job has completed."} </pre>

Verify

1. Click **Test** to validate that this task will work.
2. Run a job:
 - a. Check the Slack channel for a message.

Plan metadata examples

You can reference metadata information from the plan definition and the current plan run as part of the request of your HTTP task.

Notes:

- You can only insert metadata references for tasks that have already occurred in the plan run before the HTTP task begins.
- Each task in the current run is referenced using a two-letter code. Examples:

```
{{ $http_xx.name }}
{{ $flow_xy.name }}
```

Syntax

A plan metadata reference is constructed using the following syntax. In the appropriate textbox, enter one of the following values:

Tip: Start by typing \$, which provides access to a menu tree of metadata references for each of the metadata reference types. The final syntax is noted above.

Entered value	Plan metadata reference type
{{ \$plan	Metadata information from the plan definition or the current plan run.
{{ \$flow_	Metadata information for the flow tasks executed in the current plan run.
{{ \$flow_7p.['My Output Name'].	Metadata information for the outputs generated by the specific flow task. In this example: <ul style="list-style-type: none">• flow_7p is a reference to the specific flow task.• 'My Output Name ' is the display name for the underlying output.

Plan information

The following request body contains references to the Plan name, plan run identifier, and the flow that was just executed:

```
{ "text": "Plan:  {{ $plan.name }}
RunId:  {{ $plan.runId }}
Flow:  {{ $flow_7p.name }}
Success." }
```

Plan run information

The following request body contains plan execution information using timestamps:

```
{ "text": "Plan:  {{ $plan.name }}
RunId:  {{ $plan.runId }}
- plan start:  {{ $plan.startTime }}
Running time:  {{ $plan.duration }}
```

```
Times:
- last task start: {{$flow_7p.startTime}}
- last task end: {{$flow_7p.endTime}}
"
```

HTTP task information

You can reference information from an HTTP task that has already occurred:

```
{"text":">{{$http_qg.name}} returned {{$http_qg.statusCode}}."}
```

Flow task information

The following request body references information from a flow task in the plan:

```
{"text":">{{$flow_7p.name}} execution:
Duration: {{$flow_7p.duration}}
Status: {{$flow_7p.status}}

For more information, see jobIds: {{$flow_7p.jobIds}}
"}
```

Flow information

The following request body references information from the underlying output for the above flow task:

```
{"text": "Flow reference information:
Name: {{$flow_7p['2013 POS'].name}}
Favorite column: {{$flow_7p['2013 POS'].columns.Store_Nbr.name}}
Least favorite data source: {{$flow_7p['2013 POS'].sources['POS-r01.txt'].name}}
For more information, see jobIds: {{$flow_7p.jobIds}}
"}
```

Notes:

- You can reference columns from the generated results using the `.columns.` reference.
- You can reference information from datasources using the `.sources` reference.

For more information, see *Plan Metadata References*.

Feed metadata inputs to cloud function

This example demonstrates how you can use an HTTP task to deliver plan metadata to AWS lambda functions. A similar approach could be used for Google Cloud functions.

In this case, the `rowCount` value from the flow task execution is delivered via HTTP task to an AWS lambda function.

General steps:

1. Define your plan.
2. Flow task: Run the flow to generate the outputs needed for your Lambda function.
3. HTTP task: generates an HTTP request whose body includes a reference to the `rowCount` metadata variable. Request body:


```
{
  "rowCount": "{{flow_7p['My Flow Name'].output['My output name'].rowCount}}"
}
```

4. AWS Lambda functions: The following is pseudo-code for Lambda:

```
import json
def lambda_handler(event, context):
    httpTaskBody = json.loads(event["body"])
    rowCount = httpTaskBody["rowCount"]

    return {
        'statusCode': 200,
        'body': json.dumps(rowCount)
    }
```

5. Google Cloud functions: The following is pseudo-code for Google Cloud functions:

```
def get_row_count(request):
    request_json = request.get_json()
    if request_json and 'rowCount' in request_json:
        rowCount = request_json['rowCount']
        return rowCount
    return 'No rowCount attribute provided'
```

Verify Signatures

Depending on the target application, implementing signature verification may require developer skills.

Optionally, you can configure the platform to sign the HTTP requests sent for a flow. Signed requests guarantee that the requests are sent from the platform, instead of a third party.

Below, you can review how the signature is created, so that you can configure the receiving application to properly process the signature and its related request.

Signature Header

HTTP requests are signed by inserting the X-Webhook-Signature header in the request. These signatures are in the following form:

```
X-Webhook-Signature: t=<timestamp>,sha256=<signature>
```

where:

- <timestamp> - Timestamp when the signature was sent. Value is in UNIX time.
- <signature> - SHA256 signature. The platform generates this signature using a hash-based message authentication code (HMAC) with SHA-256.

More information on these values is available below.

Example:

```
X-Webhook-Signature: t=1568818215724,sha256=55fa71b2e391cd3ccba8413fb51ad16984a38edb3cccf81f381c4b8197ee07a
```

Check Application Tools

Depending on the application, you may need to complete one of the following sets of tasks to verify the task signatures:

NOTE: You may need to whitelist the platform in your application. See the application's documentation for details.

You may be required to create some custom coding for your application. Below, you can review details on how to do so, including a JavaScript example.

Process Signed Requests

Timestamp

The timestamp value (`t=<timestamp>`) appears at the beginning of the header value to prevent replay attacks, where an attacker could intercept a valid payload and its signature and re-transmit them.

- To avoid such attacks, a timestamp is included in the signature header and is also embedded as part of the signed payload.
- Since the timestamp is part of the signed payload, an attacker cannot change the timestamp value without invalidating the signature.
 - If the signature is valid but the timestamp is too old, you can then choose to reject the request.
 - For example, if you receive a request with a timestamp that corresponds to a date from one hour ago, you should probably reject the request.
- For more information on replay attacks, see https://en.wikipedia.org/wiki/Replay_attack.

Signature

The task signature includes as part of its hashed value:

- The secret key (entered above)
- The timestamp value
- Request data:
 - (POST/PUT/PATCH) - the body of the request
 - (GET/DELETE) - URL of the request

Step 1 - Extract the timestamp and signatures

Split the `X-Webhook-Signature` header:

1. Split values using the `,` character as a separator.
2. Split each of the parts using the `=` character.
3. Extract the values for the timestamp and signature. From the above example:
 - a. timestamp: 1568818215724
 - b. signature: 55fa71b2e391cd3ccba8413fb51ad16984a38edb3cccf81f381c4b8197ee07a

Step 2 - Create the expected signature

In the receiving application, you can recompute the signature to verify that the request was sent from the platform.

1. Concatenate the timestamp, the dot character `.` and the request body (POST/PUT/PATCH methods) or the url (GET/DELETE methods).
2. Suppose the above example is the signature for a `POST` request, and the request body is `test`. The concatenated value is the following:

```
1568818215724.test
```

3. You can now compute the HMAC authentication code in your receiving application. In the following JavaScript example, the secret key value is `mySecret`:

```
const crypto = require('crypto');

const message = '1568818215724.test'; // as defined above

const hmac = crypto.createHmac('sha256', 'mySecret');
hmac.update(message)
const expectedSignature = hmac.digest('hex');
```

Step 3 - Compare the signatures

The value returned by your code and the value included as the signature in the `X-Webhook-Signature` header should be compared:

- If the values do not match, reject the request.
- If the values do match, compute the difference between the current timestamp and the timestamp in the header. If the difference is outside of your permitted limit, reject the request.
- Otherwise, process the request normally in your application.

Create Slack Task

Contents:

- *Limitations*
 - *Prerequisites*
 - *Requirements for Slack*
 - *Steps*
 - *Plan metadata references*
-

You can create plan tasks to deliver messages to specific Slack channels to which you have access. These tasks are helpful for informing a set of stakeholders across your organization about the execution of your plans.

- A **Slack task** is a message from Trifacta® to a specified Slack workspace channel.
- A Slack task is defined as one of the tasks in a plan. For more information, see *Plan View Page*.
- This capability may need to be enabled in your environment. For more information, see *Overview of Operationalization*.

Limitations

- You can only post messages to Slack channels.
- HTTP-based requests have a 30-second timeout limit.
- Authentication must be made through OAuth.

Tip: You can also create HTTP tasks to deliver messages to a Slack channel. See *Create HTTP Task*.

Prerequisites

Requirements for Slack

- To send a message to Slack, you must create an app in the target workspace for the Slack channel to receive the message.
 - This Slack app must support OAuth authentication.
 - For more information, see <https://api.slack.com/apps>.

Steps

1. Open your plan in Plan View. Click a node to create a new task.
2. In the right panel, select **Slack task**.
3. In the Request tab, set the following parameters:

Parameter	Description
OAuth token	The OAuth token to use for posting the message.
Channel	Paste one of the following values from the Slack workspace for where to post the message: <ul style="list-style-type: none">• Channel Name: Name of the channel as it appears in Slack.• Channel ID: This value is available in the Settings page for the channel.• Member ID: You can post the message to a specific user instead of posting to a channel. A user's member ID can be found in the user's Profile page in Slack.
Message	The message to post.

Tip: Messages can include metadata information about the tasks in the current plan run. For more information, see *Plan Metadata References*.

4. To test the message, click **Test**. A success message is displayed.
5. You can rename the task. click **More menu > Edit** in the right panel.
6. To add the task to the flow, click **Save**.

Plan metadata references

Within the message of your Slack task, you can reference metadata about the plan that is being executed. For more information, see *Plan Metadata References*.

Share a Plan

Contents:

- *Limitations*
 - *Permissions*
 - *Steps*
-

This section provides an overview of sharing plans with other users for collaboration on the same plan.

You can share plans with one or more users to work together on the same plan. You can share the plan through the Plans page.

Limitations

- When a plan is shared, the underlying flow tasks are not shared directly.
- The plan can be executed only if the user has access to all the underlying flow tasks.
- Plan schedules cannot be shared with users.

Permissions

When a user is provided access to a plan, the user becomes a **collaborator** on the plan and is assigned a subset of the permissions assigned to the **owner** of the plan. If the user has minimal permissions for overall plans then sharing the plan as collaborator would be downgraded.

NOTE: A collaborator on a plan cannot delete the plan.

NOTE: In addition to the shared plan, you must have collaborator access to all underlying flows to execute a plan.

For more information, see *Overview of Sharing*.

Steps

1. From the context menu of the Plans page, select **Share**.
2. In the Share dialog box, add users as collaborators for the plan; start typing the name of a user or enter the email address of the user with whom you would want to share the plan.
3. Specify the privilege level of the user to whom you are sharing. For more information on sharing privileges, see *Overview of Sharing*.
4. Repeat the process to add multiple users.
5. Click **Save**.

Export Plan

This section provides an overview on how to export plans from one environment to another environment as a ZIP file.

You can export a plan from Trifacta® from one system to another system as a zip file. The exported zip file contains a JSON file for the plan and JSON files for each of its associated flows. The plan and its assets, such as nodes, edges, and tasks other than flow tasks are exported into the plan definition file .

For each flow included in the plan:

- Each flow is exported only once in a flow definition file, even if it is used in many plan flow tasks.
- If the flow contains any artifact files, they are included as `.data` files next to the plan definition file. These files should be imported with the flow, too.

When you export the plan ZIP file, a snapshot of the plan is taken at the time of export.

NOTE: When you export a plan that has flow tasks, then all the corresponding flows are also exported.

NOTE:

- You can unzip the exported plan ZIP, remove any of the flow files, and re-zip if you want to import the plan into the same workspace without replicating the flows.
- You can upload the plan as a single JSON file without re-zipping, if there are no flow files for the plan.

Plan exports are useful for:

- Backing up the work in progress on your plans
- Archiving of completed development work

Limitations

- The plan file does not include the plan schedules and their associated overrides.
- Import and export of plans is not supported in Deployment Manager.

Export from Plans page

Steps :

1. From the home page of Trifacta, navigate to **Plans**.
2. In the Plans page, select the required plan. From the context menu, select **Export**.
3. In the Export Plan window, select **Download package(.zip)**.
4. Enter any optional notes, if required.
5. The ZIP file is downloaded to the default download location of your desktop.

Tip: You can also export a plan from Plan View page.

Import Plan

An exported plan can be imported into Trifacta® into a different workspace.

Limitations

- You cannot import a plan that was exported in the earlier release.
- You cannot import schedules while importing a plan.
- You cannot modify plan definitions outside of the Trifacta application.

NOTE: After importing a plan, the objects referenced in the included flows must be connected to the corresponding resources available in the target system. For more information, see *Import Flow*.

NOTE: When you import a plan, the corresponding flow tasks and HTTP tasks in the plan are also imported.

NOTE:

- When you import a plan, you must import a ZIP file containing the JSON definition and any included flows.
- You can import the plan as a single JSON file without copying the flow files, as it reuses the existing flow files in the workspace.

Import

Steps:

1. From the home page of Trifacta, navigate to **Plans**.
2. From the context menu of the Plans page, select **Import**.

Tip: You can import multiple plans (ZIP files) through the file browser or through drag-and-drop. Press **CTRL/COMMAND** + click or **SHIFT** + click to select multiple files for import.

3. From the Import Plans window, select the exported zip file from your system and click **Open**.

The plan is imported and available for use in the Plans page.

Account Management Tasks

The topics below provide information on how to manage aspects of your account in Trifacta®.

Change Password

To recover your password, click **Forgot password?** in the login screen.

To change your password after you have logged in, select your name from the User menu. Enter a new password, confirm it, and save your changes. The new password is applied when you next try to log in. See *User Profile Page*.

The password for the administrator account should be changed immediately after installation is complete. See *Change Admin Password*.

Configure Your Access to S3

Contents:

- *Getting Started*
 - *Credential Provider*
 - *IAM Role*
 - *AWS Key and Secret*
-

If per-user access to S3 has been enabled in your Trifacta® deployment, you can apply your personal S3 access credentials through the AWS Credentials page. You can use the following properties to define the S3 buckets to use for uploads, job results, and temporary files.

Getting Started

You can access these settings through the Trifacta application.

Steps:

1. In the menu bar, click the User menu.
2. Select **Storage**. click **Edit** for AWS Credentials and Storage Settings, where you can review and modify your S3 access credentials.

Credential Provider

IAM Role

NOTE: This role must be created through AWS for you. For more information, please contact your AWS administrator.

Tip: This method is recommended for access AWS resources.

AWS Storage Settings

×

Please see [AWS Config Settings](#) for help completing this form.

Credential Provider

✓
IAM Role

AWS Key and Secret

Available IAM Role ARNs

Select Default IAM Role ARN

Default S3 Bucket

This bucket will contain uploaded files, temporary files, and job results.

Cancel

Save

Figure: Apply your IAM role and credentials

Setting	Description
Available IAM Role ARNs	You can specify the set of IAM Role ARN to use to authenticate to AWS resources.
Select Default IAM Role ARN	From the available IAM Role ARNs, you can specify the default one.
Default S3 Bucket	This bucket is used for storage, unless another bucket is explicitly selected. <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>NOTE: Specify the top-level bucket name only. There should not be any backslashes in your entry.</p> </div>
Extra S3 Buckets	You can specify a comma-separated string of additional S3 buckets that are available for storage. Do not put any quotes around the string. Whitespace between string values is ignored.

AWS Key and Secret

Per-user access must be enabled by your Trifacta administrator. See [S3 Access](#).

AWS Storage Settings

Please see [AWS Config Settings](#) for help completing this form.

Credential Provider

IAM Role

✓

AWS Key and Secret

AWS Access Key

AWS Secret Key

Default S3 Bucket

3fac-testing

This bucket will contain uploaded files, temporary files, and job results.

Cancel

Save

Figure: AWS Storage page

The following settings apply to S3 access.

NOTE: The values that you should use for these settings should be provided by your S3 administrator. If they have already been specified, do not modify unless you have been provided instructions to do so.

Setting	Description
AWS Access Key	This key defines the account to use to connect to AWS.
AWS Secret Key	The secret (or password) associated with the key.
Default S3 Bucket	<div>This bucket is used for storage, unless another bucket is explicitly selected.<div>NOTE: Specify the top-level bucket name only. There should not be any backslashes in your entry.</div></div>
Extra S3 Buckets	You can specify a comma-separated string of additional S3 buckets that are available for storage. Do not put any quotes around the string. Whitespace between string values is ignored.

Concepts

This section contains topics on concepts related to Trifacta® and the underlying principles driving its development.

Feature Overviews

These sections provide overviews of key features and capabilities of Trifacta®.

Tip: Use the links in these sections to access locations in the platform where these features appear.

Overview of Data Export

Contents:

- *How to Export*
 - *Export Job Results*
 - *Writing to Files*
 - *Writing to Tables*
 - *Parameterized Outputs*
 - *Ad-hoc Publishing*
 - *Exporting Metadata*
 - *Export flows*
 - *Export recipes*
 - *Export sample data*
 - *Export logs*
 - *Export via API*
 - *Job results*
-

This section provides an overview of exporting data from the Trifacta® application to your preferred destinations, such as file-based storage, connected datastores, or your desktop. In addition to exporting of job results, other types of exports are covered in this section.

Tip: In most cases, the source of your data does not limit the type of output that you can generate. You can create a file-based imported dataset and generate results to a database table. Some exceptions may apply.

How to Export

Job results are generated based on the specifications of an output object. An **output object** is a reference object for one or more types of outputs. This reference information includes full path to the output location, file or table name, and other settings. For more information, see *Create Outputs*.

In the Run Job page, you can specify additional settings and overrides. See *Run Job Page*.

Export Job Results

After you have executed a job, the application writes a set of results to the designated output locations. These results are the application of the recipe's transformation steps to the imported dataset, written to the location or locations specified in the output object in the specified output format.

You can export the results directly from the designated output destination. For more information, see *Job Details Page*.

Tip: Job results for your latest job may be exportable from Flow View. For more information, see *View for Outputs*.

Writing to Files

As a result of job execution, you can publish your outputs to a file-based system.

NOTE: You must have write permissions to the location where you are writing your output files. These permissions should be set up during initial configuration of the product. For more information, please contact your administrator.

Defaults for file-based outputs:

- Files are written to your designated output directory on the backend datastore. As needed, you can modify your default output directory. For more information, see *Storage Config Page*.
- Files are written in CSV format to the designated location.

You can modify the publishing action and generate results in your preferred formats.

- For more information on changing file output settings, see *File Settings*.
- For more information on supported file formats, see *Supported File Formats*.

Writing to Tables

You can export generated results directly to a connected relational database.

Tip: Some relational connection types support read-only or write-only connections.

The Trifacta application writes results to a database through an object called a connection. A **connection** is a configuration object that defines the interface between the application and the database. Among its properties are a set of credentials that provide access.

NOTE: You must have write permissions to the database where you are writing your output tables. These permissions must be enabled by a database administrator outside of the product.

NOTE: Connections can be shared among users. When a user chooses to share a connection, the user can also choose to share credentials. If credentials are not shared, other users must provide their own credentials if they wish to use the connection. For more information, see *Share a Connection*.

For relational databases, the Trifacta application passes the information in the connection definition to a third-party driver that performs the actual connection. Thereafter, the Trifacta application maintains the open connection as long as it is needed to write results. After the results are written, the connection is closed.

When you choose to write results to a table:

- Through the connection, you browse and select the database to which to write the results.
- You can choose to write to an existing table or to a new one.
- You can specify one of the following publishing actions on the table you selected:
 - **New:** Each run generates a new table with a timestamp appended to the name. For example, `myexample_test_1.csv`.
 - **Update:** Each run adds any new results to the end of the table.
 - **Truncate:** With each run, all rows columns of data in a table is removed and retain the empty table as an object.
 - **Load:** With each run, the table is dropped (deleted), and all data is deleted. A new table with the same name is created, and any new results are added to it.
 - **Merge:** Some databases may support merge (upsert) operations.

Additional options may be available, depending on the connection. For more information, see *Relational Table Settings*.

Parameterized Outputs

For file-or table-based publishing actions, you can parameterize elements of the output path. You can create parameters for your outputs of the following types:

- **Timestamp:** You can insert the timestamp of when the output was written as part of the path to the output location.
- **Variable:** Variable parameters allow you to insert values that you define as part of the output object.

Tip: You can optionally override the values of your variable parameters as part of your job definition.

For more information on parameters, see *Overview of Parameterization*.

Ad-hoc Publishing

After a job has successfully completed, you can review and download the set of generated outputs and export results. Optionally, you may be able to publish the generated results to a secondary datastore through the Job Details page.

NOTE: Additional configuration may be required.

For more information on ad-hoc publishing, see *Publishing Dialog*.

Exporting Metadata

In addition to the job results, you can export aspects of the flow definition and other objects that you have created in the Trifacta application. These exports can be useful for:

- Migrating flows to other workspaces
- Archiving data
- Taking snapshots of work in progress

Export flows

You can export a flow from application. An exported flow is stored in a ZIP file that contains references to all objects needed to use the flow in another workspace or project. Exported flows can be imported into the same workspace/project or a different one.

NOTE: Users of the imported flow must have access to the datasources and specified output locations. If not, these objects must be remapped in the new environment.

For more information, see *Export Flow*.

Export recipes

You can download a recipe in text form and reuse it in another flows.

Reuse recipes in a different environment

If you need to reuse a recipe in a different instance of Trifacta , you can do the following:

- Export the entire flow and import it into the new environment. Open the flow in the new environment.
- Convert all steps of a recipe into a macro. Export the macro and then import it into the new environment. For more information, see *Export Macro*.

Download recipes

You can download recipe in a text form of Wrangle (a domain-specific language for data transformation). For more information, see *Recipe Panel*.

Export sample data

From the recipe panel, you can download the current state of the data grid, which includes the current sample plus any recipe steps that have been applied to it.

Tip: When a sample is taken, it is tied to the current recipe step. All steps later in the recipe than the current recipe step are computed in memory using the sample as the baseline. For more information, see *Overview of Sampling*.

For example, if the sample was generated when the recipe cursor was displaying step 7 and you download the data from the recipe when the recipe cursor is on step 10, then you are downloading the state of the recipe at step 10.

NOTE: When a flow is shared, its samples are shared with other users. However, if shared users do not have access to the underlying sources that back a sample, they do not have access to the sample. These samples are invalid for the other users, who must create their own.

For more information, see *Samples Panel*.

Export logs

You can export logs of the following:

- **Download session logs:** You can download logs for your current from the Trifacta platform through the Help menu. For more information, see *Download Logs Dialog*.

Tip: Administrators can download a broader set of platform logs. For more information, see *Admin Download Logs Dialog*.

- **Job logs :** When a job fails, you can download job logs for analysis. For more information, see *Jobs Page*.
- **Sample logs:** If a sample fails to generate, you can retry or download logs for review. For more information, see *Samples Panel*.

Export via API

Job results

After a job has run, you can acquire the path to the results when you query for the job. For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup>

Overview of Data Import

Contents:

- *How to Import*
 - *Types of Import*
 - *Upload*
 - *Import of files*
 - *Import of tables*
 - *Imported Datasets*
 - *Persisted Data*
 - *Samples*
 - *Conversion*
 - *Caching*
 - *Sharing Imported Data*
-

This section provides an overview of data import and how different types of import are handled in Trifacta®.

How to Import

You import data for use in the Trifacta application through a reference object called an **imported dataset**. An imported dataset is a reference to the source of the data.

NOTE: The source data is never modified. In some cases, the source data may be copied to the base storage layer. For example, data that is uploaded from your local desktop must be copied to the base storage layer so that it is accessible to you and potentially other users of the Trifacta application.

Steps:

1. In the Trifacta application, click the Library icon in the left navigation bar.
2. In the Library, click **Import Data**.
3. The Import Data page is displayed.
 - a. Select the connection in the left nav bar through which you can access the data.
 - b. For more information, see *Import Data Page*.

After the data has been imported, you can reference it within the application as an imported dataset. For more information, see *Import Basics*.

Types of Import

You can import datasets or select datasets from sources that are stored on file-based storage, connected datastores, or your desktop. Following are the different types of import that you can perform in the Import Data page.

Upload

You can upload a variety of flat file formats from your local desktop. You can upload a file up to 1 GB in size.

- You can drag and drop files from your desktop to to upload them.
- You can select multiple files in the same directory for uploading at the same time.

Import of files

Trifacta supports multiple storage environments. You can import one or more files from any backend data storage systems. Each workspace has a default backend storage environment. Each user should be able to import files that are stored in accessible locations in this backend storage area.

NOTE: You must have read permissions for these storage environments to import the file. These permissions should be set up during initial configuration of the product. For more information, please contact your administrator.

NOTE: During import, the Trifacta identifies file formats based on the extension of the filename.

Import of tables

You can import one or more tables from relational datastores. Through the Import Data page, you can select or create the appropriate connection to the datastore, navigate to the required database and select the files to be imported.

NOTE: You must have read permissions for any database from which you want to import. These permissions must be enabled by a database administrator outside of the product. For more information, see *Using Databases*.

Imported Datasets

When you import a file or a table, the data that is imported to the platform is referenced as an imported dataset. An imported dataset is simply a reference to the original data. An **imported dataset** can be a reference to a file, multiple files, database table, or other type of data.

NOTE: Trifacta® does not modify the source data. It is only referenced as an imported dataset.

NOTE: The imported dataset may be broken if the path or the permissions change for the underlying dataset.

Persisted Data

In general, the Trifacta application does not retain data for a longer time than the data is explicitly needed. For example, when jobs are executed on Trifacta Photon, the source data is streamed to the Trifacta node and transformed, after which results are written. The transformed data is not maintained in the Trifacta node.

NOTE: Data is not persisted on the Trifacta node.

More information on persisted data is available below.

Samples

Samples can be generated within the product through the Samples panel. When a sample is created, it is stored within your storage directory on the backend datastore. You can create a new sample at any time.

- If the source data is larger than 10 MB in size, an initial sample is automatically generated when the recipe is first loaded in the Transformer page. This sample contains the first set of rows in the dataset up to 10 MB in size.
- If the source data contains multiple files, the initial sample for the dataset is generated from the first set of rows in the first filename listed in the directory.
- If that source data is a multi-sheet Excel file, the sample is taken from the first sheet in the file.

For more information on creating samples, see *Overview of Sampling*.

Conversion

For some file types, the Trifacta application must convert the source data into a format that is natively supported by the product. This process happens as part of the importing of data for use in the Trifacta application and is managed by the conversion service in the platform. In such scenarios, the data is read from the source and passed through the conversion service, which understands how to read the source format and can write it to a supported text format. This text version of the source data is written to the base storage layer.

For example, when a transformation job is executed, the original source data is passed through the conversion service, and the converted data is used for job execution. When the job results are written, conversion service removes the converted data.

During import, the Trifacta application identifies file formats based on the extension of the filename. The conversion process applies for the following type of files:

- **XLS and PDF:** These file types are stored in a proprietary binary format. Conversion service uses a set of libraries to convert files of these types to tabular CSV data and store the files in the base storage layer.
- **JSON:** JSON file through the conversion service provides considerable improvements in terms of quality and performance during ingestion of JSON data.

For more information, see *Supported File Formats*.

Caching

Caching refers to the process of ingesting and storing data sources in a temporary backend location for a specific period of time in order to perform any additional operations in a faster way.

Instead of reloading the source each time that an object is referenced, the Trifacta application checks the cache for a cached version and if the cache is still valid. Based on the results, the Trifacta application pulls data from the local cache instead.

Tip: Cached objects can be referenced later for faster performance on tasks such as sampling and job execution.

For more information, see *Configure Data Source Caching*.

Sharing Imported Data

You cannot shared an imported dataset as an object; however, you can share connections. If the user has permissions over the dataset that has been shared as a part of the connection then the imported dataset is accessible to the shared user.

NOTE: The shared user should have the connection credentials to access the imported dataset.

For more information, see *Overview of Sharing*.

For more information, see *Share a Connection*.

Overview of Storage

Contents:

- *Base Storage Layer*
 - *Uses of base storage layer*
 - *Base storage layer directories*
 - *Available base storage layers*
 - *Management of base storage layer*
- *External Storage*
 - *File-based systems*
 - *Cloud data warehouses*
 - *Relational systems*
 - *Management of external storage*

Trifacta supports different options for reading and writing data from your storage systems.

Base Storage Layer

The base storage layer is the datastore where Trifacta uploads data, generates profiles, results, and samples. By default, job results are written on the base storage layer. You can configure the base storage layer and other required settings.

Tip: The base storage layer must be a file-based system.

NOTE: The base storage layer must be enabled and configured during initial installation. After the base storage layer has been configured, it cannot be switched to another environment. For more information, see *Set Base Storage Layer*.

Uses of base storage layer

In general, all base storage layers provide similar capabilities for storing, creating, reading, and writing datasets.

The base storage layer enables you to perform the following functions:

1. **Storing datasets:** You can upload or store datasets in directories on the base storage layer. See below.
2. **Creating datasets:** You can read in from datasources stored in the storage layer. A source may be a single file or a folder of files.
3. **Storage of samples:** Any samples that you generate are stored in the base storage layer.
4. **Ingested data:** Some data like Excel and PDF are stored as binary (non-text) files. These files must be read and converted to CSVs, which are stored on the base storage layer.
5. **Cached data:** You can enable a cache on the base storage layer, which allows data that has been ingested to remain on the base storage layer for a period of time. This cache allows for faster performance if you need to use the data at a later time.
6. **Writing Results:** After you run the job, you can write the results to the storage layer.

Base storage layer directories

Trifacta creates and maintains the following directories and their sub-directories on the base storage layer:

--	--

Directory	Description
/trifacta/uploads	Storage of datasets uploaded through the Trifacta application. Directories beneath this one are listed by the internal identifier for each user of the product who has uploaded at least one file. Avoid using /trifacta/uploads for reading and writing data. This directory is used by the Trifacta application.
/trifacta/queryResults	Default storage of results generated job executions. Directories beneath this one are listed by the internal identifier for each user of the product who has run at least one job. For each user, these sub-directories are the default storage location for job results. These locations can be modified. See <i>Preferences Page</i> . Within the <code>queryResults</code> directory, you may find sub-directories labeled <code>datasourceCache</code> . When data source caching is enabled, data read into the product may be temporarily stored in this directory. For more information, see <i>Configure Data Source Caching</i> .
/trifacta/dictionaries	Storage of custom dictionary files uploaded by users. NOTE: This feature applies to Trifacta Self-Managed Enterprise Edition only. It is not often used.
/trifacta/tempfiles	Temporary storage location for files required for use of the product. NOTE: The <code>tempfiles</code> directory is reserved for use by the platform. It is the only directory of these that is actively cleaned by the platform.

Minimum Permissions

Trifacta requires the following operating system level permissions on the listed directories and sub-directories:

Directory	Owner Min Permissions	Group Min Permissions	World Min Permissions
/trifacta/uploads	read+write+execute	none	none
/trifacta/queryResults	read+write+execute	none	none
/trifacta/dictionaries	read+write+execute	none	none
/trifacta/tempfiles	read+write+execute	none	none

Available base storage layers

Trifacta supports the following base storage layers.

NOTE: In some deployments, the base storage layer is pre-configured for you and cannot be modified. After the base storage layer has been defined, you cannot change it.

NOTE: For all storage layers, the source data is untouched. Results are written to a location whenever a job is executed on a source dataset.

For more information, see *Storage Deployment Options*.

S3

Simple Storage Service (S3) is an online data storage service provided by Amazon, which provides low-latency access through web services. For more information, see <https://aws.amazon.com/s3/>.

For more information, see *External S3 Connections*.

HDFS

HDFS is a scalable file storage system for use across all of the nodes (servers) of a Hadoop cluster. Many interactions with HDFS are similar with desktop interactions with files and folders. However, what looks like a "file" or "folder" in HDFS may be spread across multiple nodes in the cluster. For more information, see https://en.wikipedia.org/wiki/Apache_Hadoop#HDFS.

NOTE: If you are using impersonated access on the base storage layer, then, the group minimum permissions must be read+write+access on all of the above directories and sub-directories.

For more information, see *Using HDFS*.

ADLS Gen 1

ADLS is a scalable file storage system for use across all of the nodes (servers) of a cluster. Many interactions with ADLS are similar with desktop interactions with files and folders. However, what looks like a "file" or "folder" in ADLS may be spread across multiple nodes in the cluster. For more information, see <https://docs.microsoft.com/en-us/azure/data-lake-store/data-lake-store-overview>.

ADLS Gen 2

Microsoft Azure deployments can integrate with the next generation of Azure Data Lake Store (ADLS Gen2). Microsoft Azure Data Lake Store Gen2 (ADLS Gen2) combines the power of a high-performance file system with massive scale and economy. Azure Data Lake Storage Gen2 extends Azure Blob Storage capabilities and is optimized for analytics workloads. For more information, see <https://azure.microsoft.com/en-us/services/storage/data-lake-storage/>.

WASB

WASB is a scalable file storage system for use across all of the nodes (servers) of a cluster. As with HDFS, many interactions with WASB are similar with desktop interactions with files and folders. However, what looks like a "file" or "folder" in WASB may be spread across multiple nodes in the cluster.

Management of base storage layer

Maintenance of the base storage layer must be in accordance with your enterprise policies.

Unless the base storage layer is managed by Alteryx, it is the responsibility of the customer to maintain access and perform any required backups of data stored in the base storage layer.

NOTE: Except for temporary files, Trifacta does not perform any cleanup of the base storage layer.

External Storage

You can create connections to external storage systems. You can integrate Trifacta with an external datastore. Depending on the type of connection and your permissions, the connection can be:

- read-only
- write-only
- read-write

You can create and edit connections between Trifacta® and external data stores. You can create either file-based or table-based connections to individual storage units, such as databases or buckets.

NOTE: In your environment, creation of connections may be limited to administrators only. For more information, contact your workspace administrator.

Tip: Administrators can edit any public connection.

NOTE: After you create a connection, you cannot change its connection type. You must delete the connection and start again.

File-based systems

In addition to the base storage layer, you may be able to connect to other file-based systems. For example, if your base storage layer is HDFS, you can also connect to S3.

NOTE: If HDFS is specified as your base storage layer, you cannot publish to Redshift.

For more information, see *Connection Types*.

Cloud data warehouses

The Trifacta application can be leveraged for loading and transforming data in data warehouses in the cloud. These integrations offer high performance access to reading in datasets from these and other sources, performing transformations, and writing results back to the data warehouse as needed.

Relational systems

When you are working with relational data, you can configure the database connections after you have completed the platform configuration and have validated that it is working for locally uploaded files.

NOTE: Database connections cannot be deleted if their databases host imported datasets that are in use by Trifacta. Remove these imported datasets before deleting the connection.

For more information, see *Using Databases*.

Management of external storage

To integrate with an external system, the Trifacta application requires:

- Basic ability to connect to the hosting node of the external system through your network or cloud-based infrastructure
- Requisite permissions to support the browsing, reading and/or writing of data to the storage system
- A defined connection between the application and the storage system.

Except for cleanup of temporary files, the Trifacta application does not maintain external storage systems.

Overview of Predictive Transformation

Contents:

- Overview
 - Phases
 - Visualizations
 - Selections
 - Predictive Model
 - Suggestions and Their Variants
 - Previews
 - Additional Steps - Modification
 - Wrangle
-

Based in academic research, **Predictive Transformation** refers to a set of design and interface principles that serve as the foundation for how Trifacta® users interact with their data. Predictive transformation is the linchpin of the platform. This section provides an overview of the concepts and links to locations where these concepts are surfaced in the interface.

Predictive Transformation is a registered trademark of Alteryx.

Overview

In essence, Predictive Transformation seeks to bring closer together:

1. the domain knowledge about the data, and
2. the technical knowledge of the sometimes complex operations required to render data into its final usable format.

In data wrangling, the former knowledge set resides with domain experts who understand the meaning of the data, while the latter often requires involvement of IT, which may have no contextual understanding of the data to inform their solution designs.

This process of rendering data from one format into another is generally called **data transformation**, which breaks down into a set of programming-type tasks, with an emphasis on structure, meaning, and the statistical properties of the data. These tasks include:

- statistical manipulation (profiling, outliers, imputation)
- restructuring (data extraction, nesting, pivot/unpivot)
- cleaning (standardization, deduplication, data removal)
- enrichment (join with other data, lookups of reference data)
- distillation (sampling, filtering, aggregation, windowing)

Across large, distributed datasets, these tasks can be technically challenging to properly execute. To move them out of the IT domain, Predictive Transformation seeks to deliver the following capabilities:

1. Features & Visualizations - innovative methods to display and select data of interest
2. Suggestions - based on user selection, suggested transforms are presented to you for selection and configuration
3. Previews - for the selected suggestion, previews of the anticipated change are available for review prior to inclusion in the transformations on the dataset

The above cycle is repeated over and over until the set of transformations is defined and executed to satisfaction.

Phases

Based on user selection, Predictive Transformation **guides** you toward possible next steps yet allows you to **decide** the step to take and (if necessary) refine the step definition. The core of the guide/decide loop of Predictive Transformation fits into the following iterative phases. When steps are selected, visualizations are updated, and the cycle repeats again.

Phase	UI Element	Description
Visualize	visualizations	A critical component of Predictive Transformation is the visual representation of the data, including items of interest for selection. In larger data sets, the visual cues around items of interest and the tools for interacting with them provide information on the meaning of each type of interaction and are critical for a productive and pleasant user experience.
Interact	selections	You interact directly with the visualizations to select values, columns, or other items of interest.
Predict	predictive model & suggestions	Automatically, user selections trigger queries into the predictive model. Data, metadata, and the selection of it effectively define queries of the predictive model. The model returns a set of suggested transforms. The suggestions guide you toward recommended actions on items that you has decided, through selection, are interesting. You can then decide which suggestion to undertake, including modification of the specific parameters around the suggestion. Or, you can define a completely different step to take.
Present	previews	Whenever the step to take is selected or subsequently modified, the anticipated results of that step are displayed as a preview overlay on top of the data. This method allows for easy development, rapid undoing, and a clearer understanding of the impacts of each step.

Visualizations

In Predictive Transformation, visualizations must be carefully designed to surface selectable data or metadata of interest. In Trifacta, the Transformer page has been designed to represent the underlying dataset while guiding you with selectable items.

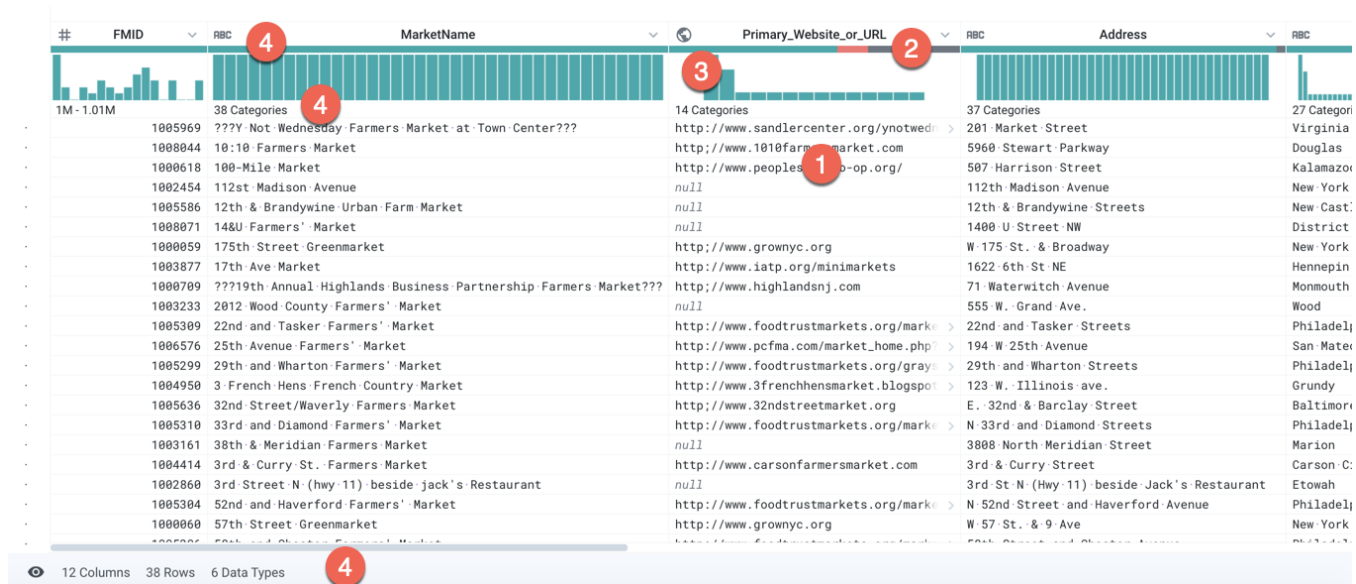


Figure: Transformer Page contains a rich overlay of information and selection cues

Specific visualization cues:

1. Data rendered into familiar grid format, regardless of underlying structure
 - a. selectable values and columns
2. Color-coded **data quality bars**:
 - a. green: valid

- b. gray: missing
 - c. red: invalid (checked against data type)
 - d. Select a color to select all corresponding values
3. **Histograms** for individual columns:
- a. Select one or more values in the histogram highlights corresponding values in other column histograms for easy visual comparisons
4. **Metadata** on entire dataset and type and statistical information for individual columns. See *Column Details Panel*.

In this manner, this visualization lifts user interaction from the domains of data and code into a more visual representation.

You must still specify via selection; the syntax of the specification is lifted into the visual domain, and the details of crafting the technical query are managed by the application.

Exploration: By design, this interaction model supports both detailed specificity and ambiguity. You select, preview the results, and then determine if the preview meets expectations. Additionally, all steps can be undone and removed from the recipe, so that you can explore different steps and entire approaches for transforming data. Solutions that demand more technical interactions often suffer from an intolerance of ambiguity, which limits your ability to express intent without significant experience and/or training. See *Transformer Page*.

Selections

As you review the visualization, a change in the cursor indicates the items that are available for selection.



Figure: Selection cursor changes on hover of selectable items

The following types of selections trigger the subsequent phases:

- cell values and values within a cell
- columns

Selecting a single column in the data grid triggers a visual profile of the column data, as well as a set of suggestions. Selecting multiple columns triggers a different set of suggestions to apply across your selected columns.

- values in a data histogram
- categories of values (valid, invalid, missing) within a data quality bar

Columns and values can be multi-selected.

You are still obligated to make selections in the data, thereby bringing domain-specific expertise to the problem of transforming it. This selection in turn triggers a more complex query through the application to the prediction service.

Predictive Model

Based on the set of selections, an inference algorithm attempts to interpret the data transformation intent of the selection and generates a ranked set of suggestions and patterns for the selections to match. For example, if you select the first three characters in a cell, the algorithm may produce two transform suggestions for data removal: one to remove the rows containing the specific text and one to keep all rows containing that pattern of text in the column.

As part of the returned results of the predictive model, matching values for the selection(s) are highlighted in the table.

The predictive model interprets selection to identify intent. Possible intentions are surfaced as one or more suggested transforms in a visual manner that minimizes exposure to the transformation language.

Suggestions and Their Variants

The set of probable next steps is computed by the predictive model from user interaction, selected data, historical information, and other sources and rendered as a set of suggestions. Since these steps are essentially predictions of user intent, they are surfaced as browsable cards, through which you can explore to disambiguate the uncertainty of intention around their data selections.

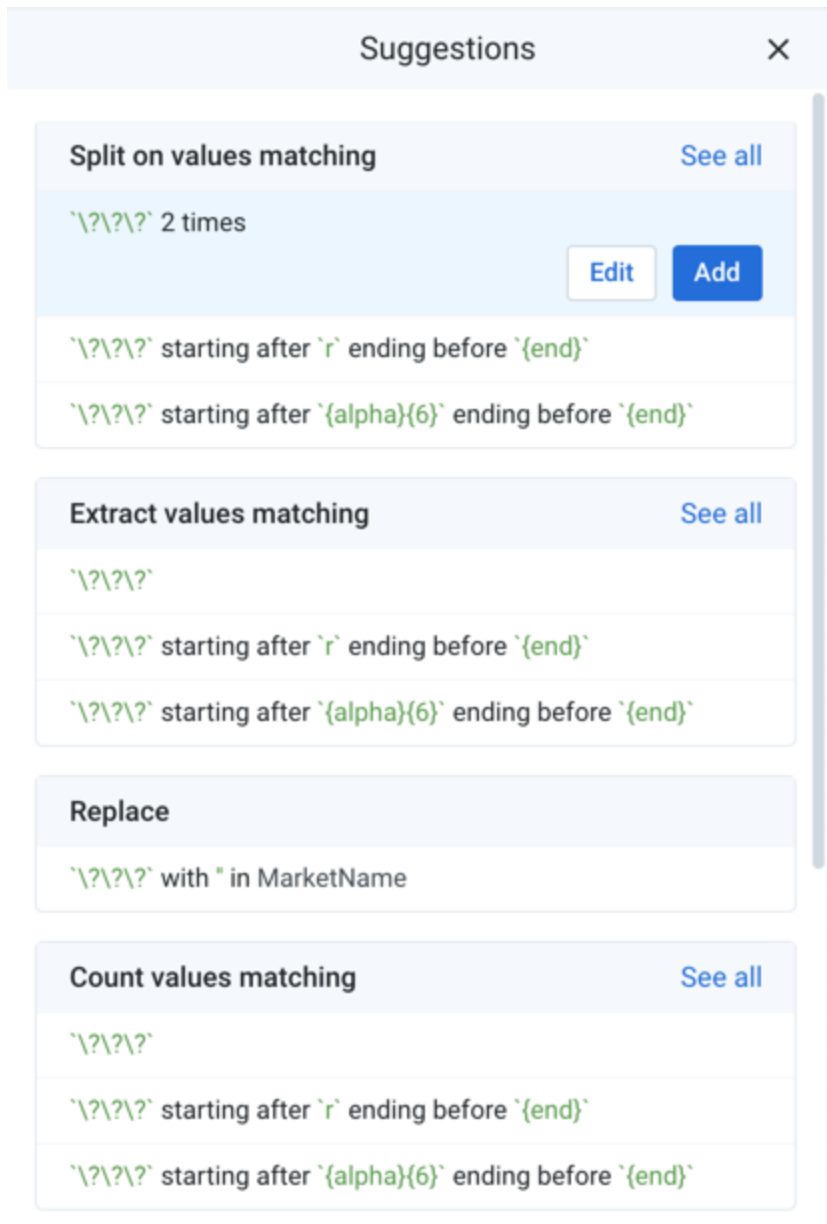


Figure: Suggestion cards - selection guides suggestion

Notes:

- Typically, pattern-based variants to the suggestion are listed first in the suggestion card.
- Pattern-based suggestions are always based on Patterns , which are easier to use than regular expressions.
- Variants using literal expressions are typically listed last. If a column has a high number of literal values, however, literal value variants may be listed first in the card.

Suggestion cards are specific enough for immediate execution. You can choose to modify the transform and its parameters, if additional specification and guidance is needed.

In a suggestion card, you may see multiple **variants** of the selected transformation.

The first variant is the most specific one applicable to the current selection in the data grid. Mouse over the variants to see different versions of the transform. As you mouse over secondary variants, the variants typically become more specific in their changes to the dataset or rarer in their usage.

When you mouse over a different transform variant in the suggestion card, the preview popup is automatically updated to reflect the variation. When you select the variant, the Preview pane is updated. You can always modify the transform to review the detailed differences.

Collaborative suggestions

Optionally, you can enable the surfacing of collaborative suggestions, which aggregate the transformation steps from users in your workspace to provide an additional category of Recently used suggestion cards. As workspace members continue to transform data that is often related, the set of Recently used suggestions become more relevant to the data on which workspace users are working. This form of data-dependent Predictive Transformation allows Trifacta to improve its understanding of the types of tasks that workspace users are trying to accomplish.

NOTE: This feature requires the machine learning service, which is enabled by default. For more information, see *Miscellaneous Configuration*.

Workspace administrators can choose to enable this feature and can configure whether data is aggregated from individual workspace users' transformations or from all workspace users' transformations. See *Workspace Settings Page*.

When this feature is enabled, collaborative suggestions appear as cards under a new **Recently used** category in the suggestions panel.

When the feature is enabled, Individual users can choose to opt-out of sharing their data with the feature. See [User Profile Page](#).

Previews

When a suggestion card is selected, the results of the selected transform are previewed in the data grid, so that you can see in advance the changes to the dataset.

Source			Preview		
	ABC	MarketName		ABC	MarketName
	<div> <div></div> <div>38 Categories</div> </div>				
969	222Y	Not Wednesday Farmers Market at Town Center	222	Y	Not Wednesday Farmers Market at Town Center
044	10:10	Farmers Market		10:10	Farmers Market
618	100-Mile	Market		100-Mile	Market
454	112st	Madison Avenue		112st	Madison Avenue
586	12th & Brandywine	Urban Farm Market		12th & Brandywine	Urban Farm Market
071	14&U	Farmers' Market		14&U	Farmers' Market
059	175th Street	Greenmarket		175th Street	Greenmarket
877	17th Ave	Market		17th Ave	Market
709	22219th Annual	Highlands Business Partnership Farmers Market	222	19th Annual	Highlands Business Partnership Farmers Marke
233	2012	Wood County Farmers' Market		2012	Wood County Farmers' Market
309	22nd and Tasker	Farmers' Market		22nd and Tasker	Farmers' Market
576	25th Avenue	Farmers' Market		25th Avenue	Farmers' Market
299	29th and Wharton	Farmers' Market		29th and Wharton	Farmers' Market
950	3 French Hens	French Country Market		3 French Hens	French Country Market
636	32nd Street/Waverly	Farmers Market		32nd Street/Waverly	Farmers Market
310	33rd and Diamond	Farmers' Market		33rd and Diamond	Farmers' Market
161	38th & Meridian	Farmers Market		38th & Meridian	Farmers Market
414	3rd & Curry St.	Farmers Market		3rd & Curry St.	Farmers Market
860	3rd Street N (hwy 11)	beside jack's Restaurant		3rd Street N (hwy 11)	beside jack's Restaurant
304	52nd and Haverford	Farmers' Market		52nd and Haverford	Farmers' Market
060	57th Street	Greenmarket		57th Street	Greenmarket
006	58th and 59th Street	Greenmarket		58th and 59th Street	Greenmarket

Figure: Previewed effects of transform

When the transform is added to the recipe, the transform is rendered into the data transformation language and applied in real-time to the dataset, so that you can immediately begin working on the next step of the process.

When a transform is selected, the selected transform and any additional guidance that you provide is translated into a specific, programmatic step in the transformation language. This step, in turn, is rendered into a complex and potentially distributed query that is applied across the dataset. In this manner, additional technical details and the knowledge required to master them are removed from user requirements.

Additional Steps - Modification

Modification via Transform Builder

As needed, any selection can be modified, such that you may tweak parameters to further refine intention to reach a specific outcome. In Trifacta, you can click **Edit** to tweak individual transformations in the Transform Builder.

The screenshot displays the Trifacta Transform Builder interface. A 'Replace text or patterns' dialog box is open on the right side. The 'Column' field is set to 'MarketName'. The 'Find' field contains a regular expression: ``\?`?\?``. The 'Replace with' field is set to 'String'. The background shows a data table with columns 'Source' and 'Preview'. The 'Source' column lists various market names, and the 'Preview' column shows the result of the transformation. The dialog box has 'Cancel' and 'Add' buttons at the bottom right.

Figure: Modifying a transform in the Transform Builder

Wrangle

The actual steps of transformation are authored in Wrangle (a domain-specific language for data transformation). Wrangle includes the following characteristics:

- Single-source transformations, with results rendered without modification to the original source data
- General cleaning and transformation operations on numerical and textual data of varying data types
- Structural transformations for managing nested data like JSON
- Multi-dataset transformations such as lookups, joins, and unions
- Transformation of data to metadata, such as pivot and unpivot operations
- Text selection patterns, including regular expressions, as a macro-type set of references. See *Text Matching*.

For a list of available transforms and functions, see *Language Index*.

For more information, see *Wrangle Language*.

Overview of the Type System

Contents:

- *How Data Types Are Used*
 - *Data Types*
 - *Logical data types*
 - *Complex data types*
 - *Logical and complex data types*
 - *Types in Source Data*
 - *Schematized files*
 - *Schematized tables*
 - *Inferred data types*
 - *Type Inference*
 - *Type inference in the application*
 - *Working with Data Types*
 - *Data types in the grid*
 - *Changing the data type*
 - *Changing the data format*
 - *Type functions*
 - *Type Conversions on Export*
 - *Type Conversions*
-

This section provides an overview of how data types are managed during import, transformation, and export of data in Trifacta®.

Terms:

A **data type** is a definition of a set of values, including:

- Possible set of values (e.g. Dates from 1/1/1400 to 12/31/2599)
- Meaning of those values (e.g. two-letter codes can represent states of the United States)
- Set of operations that can be applied to those values (e.g. functions applicable to integers)

A **data format** is a representation of the underlying type, which has the same meaning and available operations associated with the data type. For example, the following values are all valid for Datetime data type, but each is represented in a different format:

```
12/31/2021
31-12-2021
December 31, 2021
```

NOTE: Some data types can be explicitly formatted through functions. Other data types support different formats implicitly through the application.

How Data Types Are Used

In the Trifacta application, data types are used for the following:

- Anomaly detection (Is the value valid or invalid?)
- Suggestions (What are the available transformation suggestions for this column based on its data type?)

- Standardization (How can all of these valid values be standardized for the column's data type?)
- Pattern recognition (How to identify different formats in the same column?)

Data Types

Trifacta supports the following categories of data types:

Logical data types

A **logical** data type is a class of values that is understood by native system representations.

Tip: These types are recognized internally by Trifacta. Each running environment to which Trifacta connections natively supports these logical data types.

These data types have no additional specification requirements:

Data Type	Description
<i>String Data Type</i>	Any non-null value can be typed as String. A String can be anything.
<i>Integer Data Type</i>	The Integer data type applies to positive and negative numeric values that have no decimal point.
<i>Decimal Data Type</i>	Decimal data type applies to floating points up to 15 digits in length. <ul style="list-style-type: none"> • In the Trifacta application, this data type is referenced as <code>Decimal</code>. • In storage, this data type is written as <code>Double</code>.
<i>Boolean Data Type</i>	The Boolean data type expresses true or false values.
<i>Datetime Data Type</i>	Trifacta® supports a variety of Datetime formats, each of which has additional variations to it.
<i>Object Data Type</i>	An Object data type is a method for encoding key-value pairs. A single field value may contain one or more sets of key-value pairs.
<i>Array Data Type</i>	An array is a list of values grouped into a single value. An array may be of variable length; in one record the array field may contain two elements, while in the next record, it contains six elements.

Formatting differences may apply. For example, Trifacta may recognize `Y` and `N` as Boolean data type, while other systems may not.

Complex data types

A **complex** data type typically is defined by applying additional restrictions on String data type values to define the class of possible values. For example, Trifacta supports a Gender data type, which validates values such as `M` & `F` and `male` and `female` as Gender data type.

The following are the complex data types supported by Trifacta.

Data Type	Description
<i>Social Security Number Data Type</i>	This data type is applied to numeric data following the pattern for United States Social Security numbers.
<i>Phone Number Data Type</i>	This data type is applied to numeric data following common patterns that express telephone numbers and known valid phone numbers in the United States.
<i>Email Address Data Type</i>	This data type matches text values that are properly formatted email addresses.

<i>Credit Card Data Type</i>	Credit card numbers are numeric data that follow the 14-digit or 16-digit patterns for credit cards.
<i>Gender Data Type</i>	This data type matches a variety of text patterns for expressing male/female distinctions.
<i>Zip Code Data Type</i>	This data type matches five- and nine-digit U.S. zipcode patterns.
<i>State Data Type</i>	State data type is applied to data that uses the full names or the two-letter abbreviations for states in the United States.
<i>IP Address Data Type</i>	The IP Address data type supports IPv4 address.
<i>URL Data Type</i>	URL data type is applied to data that follows generalized patterns of URLs.
<i>HTTP Code Data Type</i>	Values of these data types are three-digit numeric values, which correspond to recognized HTTP Status Codes.

Complex data types are typically defined based on a regular expression applied to String values. For example, the following regex is used to define the accepted values for the Email Address data type:

```
"regexes": [
  "^[a-z0-9.!#$%&'*/+=?^_`{|}~-]+@[a-z0-9.-]+(?:\\.[a-z0-9-]+)*\\.([a-z]{2,})$"
],
```

NOTE: Regex-based data types can be modified, although in most cases, it is unnecessary. Regular expressions are considered a developer-level configuration. For more information, please contact *Alteryx Customer Success Services*.

Logical and complex data types

Data type	Category	Internal data type	Notes
<i>String Data Type</i>	logical	String	The default data type. Any non-empty/non-value is valid for String data type.
<i>Integer Data Type</i>	logical	Int	Use <i>NUMFORMAT Function</i> to format these values. Underlying values are not modified.
<i>Decimal Data Type</i>	logical	Float	Use <i>NUMFORMAT Function</i> to format these values. Underlying values are not modified.
<i>Boolean Data Type</i>	logical	Bool	
<i>Datetime Data Type</i>	logical	Datetime	Use <i>DATEFORMAT Function</i> to format these values. Underlying values are not modified.
<i>Object Data Type</i>	logical	Map/Object	
<i>Array Data Type</i>	logical	Array	
<i>Social Security Number Data Type</i>	complex	String	String data type constrained by a regular expression.
<i>Phone Number Data Type</i>	complex	String	String data type constrained by a regular expression.
<i>Email Address Data Type</i>	complex	String	String data type constrained by a regular expression.
<i>Credit Card Data Type</i>	complex	String	String data type constrained by a regular expression.
<i>Gender Data Type</i>	complex	String	String data type constrained by a regular expression.
<i>Zip Code Data Type</i>	complex	String	String data type constrained by a regular expression.
<i>State Data Type</i>	complex	String	String data type constrained by a regular expression.
<i>IP Address Data Type</i>	complex	String	String data type constrained by a regular expression.
<i>URL Data Type</i>	complex	String	String data type constrained by a regular expression.

HTTP Code Data Type	complex	String	String data type constrained by a regular expression.
---------------------	---------	--------	---

Types in Source Data

Depending on the source system, imported data may be typed to Trifacta data types according to one of the following methods.

Tip: For each method, Trifacta attempts to map the source data to one of the above data types. For schematized sources, however, you may wish to use the original data types in the source. Optionally, you can choose to disable the mapping of source to internal data type. See "Type Inference" below.

Schematized files

Some file formats, such as Avro or Parquet, are stored in a non-readable format. Part of the metadata associated with the file is information identifying the schema of the file. A **schema** represents the data types and other constraints of individual columns. It can be read independently of the data in the source table.

Schematized tables

For relational sources, schema information is typically stored with the table. This schema information defines data type validation within the datastore and can be read independently from the source table.

Tip: Database schemas can be used to define a class of tables to ensure consistency within a database.

Inferred data types

In most cases, an imported data source is assigned a data type for each column based on a review of a subset of the data. For example, a CSV file contains no information about the data types of individual columns. The data types for each column must be assigned by Trifacta. This process is called type inference. For more information, see "Type Inference" below.

Type Inference

By default, the Trifacta application applies type inference for imported data. The application attempts to infer a column's appropriate data type in the application based on a review of the first lines in the sample.

Tip: In many programming languages, a column must be explicitly "type cast" to a data type as part of a functional operation. Wrangle handles this typecasting for you through the process of type inference.

NOTE: Mapping source data types to Trifacta data types depends on a sufficient number of values that match the criteria of the internal data type. The mapping of import types to internal data types depends on the data.

- Type inference needs a minimum of 25 rows of data in a column to work consistently.
- If your dataset has fewer than 20 rows, type inference may not have sufficient data to properly infer the column type.

In some datasets, the first 25 rows may be of a data type that is a subset of the best matching type. For example, if the first 25 rows in the initial sample match the Integer data type, the column may be typed as Integer, even if the other 2,000 rows match for the Decimal data type. If the column data type is unmodified:

- The data is written out from Trifacta as Integer data type. This works for the first 25 rows.
- The other 2,000 rows are written out as null values, since they do not match the Integer data type. If the source data used decimal notation (e.g. 3.0 in the source), then those values are written out as null values, too.

In this case, it may be easier to disable type inference for this dataset.

Tip: If you are having trouble getting your imported dataset to map to expected data types, you can disable type inference for the individual dataset. For more information, see *Import Data Page*.

After data is imported, the Trifacta application provides some mechanisms for applying stronger typecasting to the data. Example:

- If all input values are double-quoted, then Trifacta evaluates all columns as String type. As a result, type inference cannot be applied.
- Since non-String data types cannot be inferred, then the first row cannot be detected as anomalous against the inferred type (String). Column headers cannot be automatically detected from double-quoted source files.

Tip: The default data type is String. If the Trifacta application is unable to evaluate a column's data type, the type is mapped to String data type. Within the application, you can use functions to remap the data type or to parse values according to a specified type.

For more information, see "Working with Data Types" below.

Disable type inference

For schematized files or tables, the Trifacta application inference of data type from the source may result in incorrect initial typing of a dataset's columns in the application. As needed, you can disable type inference for the following:

NOTE: When type inference is disabled for imported datasets, it is not disabled within the Trifacta application. For more information, see "Type inference in the application" below.

- **Disable for individual files:** In the Import Data page, select the file. In the right-hand column, click **Edit Settings**. For more information, see *File Import Settings*.
- **Disable for individual tables:** In the Import Data page, select the table. In the right-hand column, click **Edit Settings**. For more information, see *Relational Table Settings*.
- **Disable for individual connections:** In the Connections page, edit the connection. In the Edit Connection window, select **Disabled** under Default Column Data Type Inference. By default, all datasets through this connection have type inference disabled. For more information, see *Create Connection Window*.
- **Disable globally:** If desired, you can disable type inference for all users of the workspace. For more information, see *Disable Type Inference*.

Type inference in the application

Within the Trifacta application, column data types may be re-inferred based on your activities in the Transformer page:

NOTE: Disabling type inference does not disable the re-inference of types in the Transformer page.

The following general actions may result in column data types being re-inferred:

- After a sample is taken, column data types are inferred based on the first set of rows in the sample.
- If a transform or function is provided with a data type that does not match the expected input data type, the values are typecast to the expected output, so you may see changes to the data type of the output to better align with the function.
- Multi-dataset operations generally do not cause re-inferring of data types. However, if there is a mismatch of data types between two columns in a union operation, for example, the data type of the first dataset is preferred.

Working with Data Types

After data has been imported, you can remap individual column types through recipe steps. For more information, see *Change Column Data Type*.

Data types in the grid

When a sample is loaded, the data types and their formats for each column are inferred by default. Data types and formatting information is displayed for each column in the Transformer page.

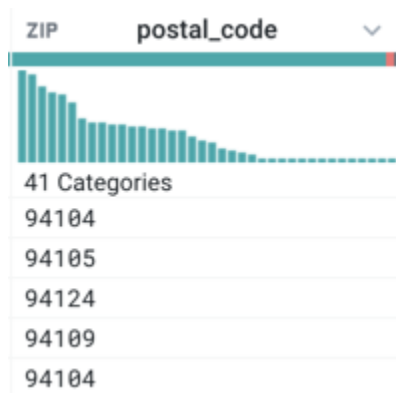


Figure: Column header example

At the top of each column, you can review graphical representations of type information:

- **Data type indicator:** To the left of the column name, you can see a graphic of the data type. In the above, the data type is set to Zip code.

Tip: Select the data type indicator to change the column to a different data type. This change is added as a step in your recipe. See "Changing the data type."

- **Data quality bar:** Below the column name, you can see a bar indicating the relative percentage of valid, invalid and empty values in the column, compared to the listed data type.
 - Green: Valid for the data type

- Red: Invalid for the data type
- Gray: empty or null

Tip: Select one of the colored bars to be prompted by a set of transformation suggestions that can be applied to the selected set of values.

- **Column histogram:** Below the data quality bar, you can see the distribution of values within the column. The column histogram may represent the data in different ways, depending on the column's data type.

Tip: Click or SHIFT-click values in the histogram to be prompted for transformation suggestions that can be applied to the selected values.

For more information, see *Data Grid Panel*.

For more information, see *Column Menus*.

Changing the data type

Change the data type through the data type menu at the top of a column.

Tip: For some types, such as Datetime type, you must select a data format when you are selecting the type. See below.

NOTE: Changing the data type may not change the underlying logical type. For example, if you change a String column to a Gender column, the underlying data is still stored as String values.

Changing type across multiple columns

To change the data type for multiple columns, you can a transformation similar to the following, which changes the data type from the `reqProdId` column to the `prodC` column and all columns in between:

Transformation Name	Change column data type
Parameter: Columns	Range
Parameter: Column list	<code>reqProdId~prodC</code>
Parameter: New type	String

Changing the data format

You can use the following functions to apply formatting on top of a column of a specified data type. For example, depending on your locale, numbers may require different formatting for use of the decimal point and the digit separator.

NOTE: When you apply a formatting function to a column, the data appears in the specified format in the Trifacta application, but the underlying data is unmodified. Formatting changes appear as a step in your recipe and are applied to the generated results.

Formatting Function	Applicable Data Type	Description
<i>NUMFORMAT Function</i>	Integer, Decimal	Formats a numeric set of values according to the specified number formatting. Source values can be a literal numeric value, a function returning a numeric value, or reference to a column containing an Integer or Decimal values.
<i>DATEFORMAT Function</i>	Datetime	Formats a specified Datetime set of values according to the specified date format. Source values can be a reference to a column containing Datetime values.
<i>UNIXTIMEFORMAT Function</i>	Datetime	Formats a set of Unix timestamps according to a specified date formatting string.

Type functions

The Trifacta application provides a set of functions for managing types.

Validation functions

These functions can be used to test for valid or invalid values against a specific data type.

Function	Description
<i>VALID Function</i>	Tests whether a set of values is valid for a specified data type and is not a null value.
<i>ISMISMATCHED Function</i>	Tests whether a set of values is not valid for a specified data type.
<i>IFVALID Function</i>	The IFVALID function writes out a specified value if the input expression matches the specified data type. Otherwise, it writes the source value. Input can be a literal, a column reference, or a function.
<i>IFMISMATCHED Function</i>	The IFMISMATCHED function writes out a specified value if the input expression does not match the specified data type or typing array. Otherwise, it writes the source value. Input can be a literal, a column reference, or a function.

Parsing functions

These functions can be used to parse String values against a specific data type.

Function	Description
<i>PARSEINT Function</i>	Evaluates a String input against the Integer datatype. If the input matches, the function outputs an Integer value. Input can be a literal, a column of values, or a function returning String values.
<i>PARSEFLOAT Function</i>	Evaluates a String input against the Decimal datatype. If the input matches, the function outputs a Decimal value. Input can be a literal, a column of values, or a function returning String values.
<i>PARSEBOOL Function</i>	Evaluates a String input against the Boolean datatype. If the input matches, the function outputs a Boolean value. Input can be a literal, a column of values, or a function returning String values.
<i>PARSEDATE Function</i>	Evaluates an input against the default input formats or (if specified) an array of Datetime format strings in their listed order. If the input matches one of the formats, the function outputs a Datetime value.
<i>PARSEARRAY Function</i>	Evaluates a String input against the Array datatype. If the input matches, the function outputs an Array value. Input can be a literal, a column of values, or a function returning String values.
<i>PARSEOBJECT Function</i>	Evaluates a String input against the Object datatype. If the input matches, the function outputs an Object value. Input can be a literal, a column of values, or a function returning String values.
<i>PARSESTRING Function</i>	Evaluates an input against the String datatype. If the input matches, the function outputs a String value. Input can be a literal, a column of values, or a function returning values. Values can be of any data type.

Managing null and empty values

These functions allow you to generate or test for missing or null values.

Function	Description
<i>ISNULL Function</i>	The <code>ISNULL</code> function tests whether a column of values contains null values. For input column references, this function returns <code>true</code> or <code>false</code> .
<i>NULL Function</i>	The <code>NULL</code> function generates null values.
<i>IFNULL Function</i>	The <code>IFNULL</code> function writes out a specified value if the source value is a null. Otherwise, it writes the source value. Input can be a literal, a column reference, or a function.
<i>ISMISSING Function</i>	The <code>ISMISSING</code> function tests whether a column of values is missing or null. For input column references, this function returns <code>true</code> or <code>false</code> .
<i>IFMISSING Function</i>	The <code>IFMISSING</code> function writes out a specified value if the source value is a null or missing value. Otherwise, it writes the source value. Input can be a literal, a column reference, or a function.

Type Conversions on Export

The Trifacta application attempts to map the data types that you have specified to match data types in the target platform.

NOTE: Values that do not match the data type of the target system for a column are subject to the method by which the target system handles mismatches. Rows could be dropped. Values can be rendered as null values. You should attempt to verify that all columns have valid values before generating results.

NOTE: Missing or null values may be treated differently between target systems. Additionally, if these systems feed downstream systems, those systems may have independent rules for managing missing or null values.

Type Conversions

Item	Description
<i>Avro Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and the Avro file format.
<i>DB2 Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and DB2 databases.
<i>Hive Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and Hive.
<i>Oracle Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and Oracle databases.
<i>MySQL Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and MySQL databases.
<i>Parquet Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and the Parquet file format.
<i>Postgres Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and PostgreSQL databases.
<i>Redshift Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and Redshift.
<i>Snowflake Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and Snowflake databases.

<i>AWS Glue Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and AWS Glue.
<i>Salesforce Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and Salesforce.
<i>SQL Server Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and SQL Server databases.
<i>SQL DW Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and SQL DW datastores.
<i>Databricks Tables Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and Databricks Tables.
<i>Tableau Hyper Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and Tableau Hyper format.
<i>Teradata Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and Teradata databases.
<i>SharePoint Data Type Conversions</i>	This section covers data type conversions between the Trifacta® application and SharePoint.

For more information, see *Type Conversions*.

Overview of Schema Management

Contents:

- *Overview of Schemas*
 - *Input type conversions*
 - *Schema Validation*
 - *Limitations*
 - *Enable*
 - *Use*
 - *Schema Refresh*
 - *Limitations*
 - *Effects of refreshing schemas*
 - *Refresh your schemas*
 - *Output Schemas*
 - *Output type conversions*
 - *Target schemas*
-

A **schema** refers to the sequence and data type of columns in a dataset. Schemas are applicable to relational tables and some file formats. This section provides an overview of how Trifacta® enables the capture and tracking of changes of input schemas as well as the methods available for transforming your data to match a target schema.

Overview of Schemas

A schema is a skeleton structure that represents the logical view of the dataset. The dataset can be a file, table, or a SQL query in a database. A schema defines how the data is structured and organized. Schema information includes:

- Column names
- Column ordering
- Column data types

Schemas may apply to relational tables and schematized file formats such as Avro and Parquet.

Input type conversions

Depending on the data source, Trifacta® can read in native data types into Trifacta data types. For more information, see *Type Conversions*.

Schema Validation

Over time, schema sources may change in major and minor ways, often without warning. From within the Trifacta application, schema changes may appear as broken recipe steps and can cause data corruption downstream. To assist with these issues, the Trifacta application can be configured to monitor schema changes on your dataset. Schema validation performs the following actions on your dataset:

- On read, the schema information from the dataset is captured and stored separately in the Trifacta database. This information identifies column names, data types, and ordering of the dataset.
- When the dataset is read during job execution, the new schema information is read and compared to the stored version, which enables identification of changes to the dataset.

Tip: This check occurs as the first step of the job execution process and is labeled as **Schema validation**.

- You can configure the Trifacta application to halt job execution when schema validation issues have been encountered.

Tip: Configuration settings can be overridden for individual jobs.

Limitations

- Schema validation applies only to sources that have published schemas (relational datasources and schematized file types).

NOTE: CSV files are not supported.

NOTE: If you attempt to refresh the schema of a parameterized dataset based on a set of files, only the schema for the first file is checked for changes. If changes are detected, the other files are contain those changes as well. This can lead to changes being assumed or undetected in later files and potential data corruption in the flow.

Enable

Schema management service

If you are not enabling schema validation, the Schema Management service can be disabled.

Steps:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
2. Locate the following parameter and set it to `false`:

```
"schema-management-service.enabled": true,
```

3. Save your changes and restart the platform.

Settings

At the project or workspace level, an administrator can set the default settings for outputs to validate schemas or not.

Tip: Workspace-level defaults can be overridden at the job level, even if the workspace-level settings are disabled. For more information, see *Run Job Page*.

Use

When schema validation is enabled and a job is launched, the schema validation check is performed in parallel with the data ingestion step. The results of the schema validation check are reported in the Job Details page in the Schema validation stage.

NOTE: Jobs may be configured to fail if schema validation checks fail. If jobs are not configured to fail, jobs may complete with warnings and publish output data to the specified targets, when schema validation fails.

For more information, see *Job Details Page*.

When schema validation detects differences in the Job Details page, those findings can be explored in detail. See *Schema Changes Dialog*.

Job-level overrides

You can override the project or workspace level settings for schema validation for individual jobs. For more information, see *Run Job Page*.

Schema Refresh

Schema refresh enables on-demand updating of your imported dataset schemas to capture changes to columns. For example, when you are working with datasets in a flow view, you can refresh your imported datasets' schemas by checking the source schema for changes. Schema refresh automatically generates a new initial sample, which allows you to gather fresh data in the Transformer page.

Schema refresh applies to:

- Relational schemas
- Schematized files
- Delimited files

NOTE: Delimiter files include CSVs and TSVs and can include other files whose delimiters can be inferred by the Trifacta application during import. Delimited files do not contain data type information; data types are inferred by the Trifacta application for these file types.

NOTE: File types that require conversion, such as Excel, PDF, and JSON, are not supported.

Key Benefits:

- Reduces the number of duplicate or invalid datasets created from the same source.
- Reduces challenges of replacing datasets and retaking samples.

Limitations

NOTE: If you attempt to refresh the schema of a parameterized dataset based on a set of files, only the schema for the first file is checked for changes. If changes are detected, the other files are assumed to contain those changes as well. This can lead to changes being assumed or undetected in later files and potential data corruption in the flow.

- You cannot refresh the schemas of reference datasets or uploaded sources.
- Schema refresh does not apply to any file formats that require conversion to native formats. These file formats include PDF, Excel, and JSON among others.

- If a column's data type is modified and other changes, such as column name changes, are not detected, this change is not considered a schema drift error.

Effects of refreshing schemas

When you choose to refresh a schema, the schema is refreshed without checking for changes, which forces the invalidation of all samples and recollection of a new initial sample. Other samples must be recreated. In some environments, this sample collection incurs costs.

When you refresh the schema in the Trifacta application:

- The source schema is applied to the imported dataset in all cases.
 - All the existing samples are invalidated.
 - A new initial sample is generated, which updates the previewed data. This may take some time.
- Addition or removal of columns may cause recipe steps to break, which can cause any transformation jobs to fail. You must fix these broken steps in the Recipe panel.

Refresh your schemas

For more information on how to refresh the schemas of your datasets, see:

- *Library Page*
- *Dataset Details Page*
- *View for Imported Datasets*

Via API:

For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/asyncRefreshSchema>

Output Schemas

Output type conversions

Depending on the output system, Trifacta can deliver your results in columns and data types native to the target. For more information, see *Type Conversions*.

Target schemas

As needed, you can import a dataset the columns of which can serve as the target of your transformation efforts. When this target schema is imported, it is super-imposed on the columns of your dataset in the Transformer page, allowing you to quickly change the naming, order, and data typing of your columns to match the target schema. For more information, see *Overview of RapidTarget*.

Overview of Standardization

Contents:

- *Standardization Methods*
- *Invalid Values*
- *Standardize Values by Clustering*
- *Standardize Formatting by Patterns*
- *Standardize Using Functions*
 - *Functions for strings*
 - *Functions for numbers*
 - *Functions for dates*
- *Custom Data Types*
 - *Custom type using a regular expression*

Trifacta® provides multiple mechanisms for reviewing your column values and identifying patterns in the data format or similar values which mean the same thing. This section summarizes the available methods of standardization, as well as their recommended uses.

Standardization Methods

Through simple visual tools, you can select the patterns or clustered value to standardize and, when prompted, the patterns or values to use as their standard. As needed, you can apply formatting or structuring functions to the data for finer grain controls.

You can use any of the following methods for standardizing values in your dataset's columns. Depending on the situation, you may choose to mix-and-match these methods. Details on these methods are below.

Method	Description	Recommended Uses	How to Use
By clustering	Trifacta can identify similar values using one of the available algorithms for comparing values. You can compare values based on spelling or language-independent pronunciation.	<ul style="list-style-type: none">• Standardize values to correct spelling differences, capitalization, whitespace, and other errors.• Values must be consistent across rows of the column.• Primarily used for string-based data types.	Available through the <i>Standardize Page</i>
By pattern	Trifacta can identify common patterns in a set of values and suggest transformations to standardize the values to a common format.	<ul style="list-style-type: none">• Standardize values to follow a consistent format, such as phone numbers or social security numbers.• Data type follows a somewhat consistent format and needs reshaping.	Available in the Patterns tab in the <i>Column Details Panel</i>
By function	You can apply one or more specific functions to cleanse your data of minor errors in formatting or structure.	<ul style="list-style-type: none">• Good method for improving the performance of pattern- or algorithm-based matching.• Some functions are specific to a data type, while others have more general application.	Edit column with formula in the <i>Transform Builder</i>
Mix-and-match	You can use combinations of the above methods for more complex use cases.	<ul style="list-style-type: none">• Combine function-based standardization for global changes to all values with cluster- or pattern-based standardization for individual value changes.	

Invalid Values

These standardization techniques assume that your column contains only valid or empty values.

Tip: Standardization may help to cut down the number of invalid values. Before you begin standardizing, however, you should select the red bar in the column histogram to review the values that are invalid for the current type and to fix them via suggestion if possible. For more information, see *Find Bad Data*.

Standardize Values by Clustering

Using one of the supported matching algorithms, Trifacta can cluster together similar column values. You can review the clusters of values to determine if they should be mapped to the same value. If so, you can apply the mapping of these values within the application.

For more information, see *Overview of Cluster Clean*.

Standardize Formatting by Patterns

For individual columns, Trifacta can analyze column values for patterns and then provide suggestions for how to normalize the patterned values into a consistent format. For example, the same US phone number can be represented in any of the following methods:

```
555-1212
415-555-1212
4155551212
(415) 555-1212
+1 (415) 555-1212
```

Tip: Pattern-based standardization is useful for confirming values in a column to a specific format. This method is applicable to data types like phone numbers, dates, social security numbers, and to a lesser extent email addresses and URLs.

You can apply pattern-based standardization through the Patterns tab. See *Column Details Panel*.

Standardize Using Functions

The following functions can be useful for standardizing values.

Functions for strings

All values can be converted to string, so these string functions can be applied to any column if its data type is converted to String data type.

Tip: The clustering algorithms may apply some of these functions to values in your column for purposes of comparison.

Category	Function	Description
String Conversion	CHAR Function	Generates the Unicode character corresponding to an inputted Integer value.
	UNICODE Function	Generates the Unicode index value for the first character of the input string.
Case Conversion	UPPER Function	All alphabetical characters in the input value are converted to uppercase in the output value.
	LOWER Function	All alphabetical characters in the input value are converted to lowercase in the output value.

	<i>PROPER Function</i>	Converts an input string to propercase. Input can be a column reference or a string literal.
Cleanse Functions	<i>TRIM Function</i>	Removes leading and trailing whitespace from a string. Spacing between words is not removed.
	<i>TRIMQUOTES Function</i>	Removes leading and trailing quotes or double-quotes from a string. Quote marks in the middle of the string are not removed.
	<i>REMOVEWHITESPACE Function</i>	Removes all whitespace from a string, including leading and trailing whitespace and all whitespace within the string.
	<i>REMOVESYMBOLS Function</i>	Removes all characters from a string that are not letters, numbers, accented Latin characters, or whitespace.
String Sizing Functions	<i>LEFT Function</i>	Matches the leftmost set of characters in a string, as specified by parameter. The string can be specified as a column reference or a string literal.
	<i>RIGHT Function</i>	Matches the right set of characters in a string, as specified by parameter. The string can be specified as a column reference or a string literal.
	<i>PAD Function</i>	Pads string values to be a specified minimum length by adding a designated character to the left or right end of the string. Returned value is of String type.
String Comparison Functions		See <i>Compare Strings</i> .

Example:

Trifacta supports nesting functions within each other. The following transformation performs some basic cleanup on all columns in your dataset that are of String type.

Transformation Name	Edit column with formula
Parameter: Columns	*
Parameter: Formula	IFVALID(\$col, 'String', LEFT(UPPER(TRIM(\$col)), 32))

- The Columns value is a wildcard, which in this case applies the transformation across all columns in the dataset (*).
- In the Formula, you see a nested expression. If the value in the column is valid against String data type, then, do the following to the column value:

NOTE: The IFVALID function tests each row value for validation against the specified data type. It does not test the column against the data type. See *IFVALID Function*.

- The TRIM function removes leading and trailing whitespace, which may register as a difference between values. See *TRIM Function*.
- The UPPER function then converts the output of the TRIM function to all uppercase. So, differences in capitalization are eliminated. See *UPPER Function*.
- The LEFT function truncates the output of the UPPER function to a maximum of 32 characters. See *LEFT Function*.

The net result of this single step applied to all columns is to eliminate whitespace, convert to uppercase, and then truncate the length of each string to only 32 characters.

For more information, see *Cleanse Tasks*.

Functions for numbers

You can use the following functions to standardize numeric values.

Function	Description
<i>ABS Function</i>	Computes the absolute value of a given numeric value. The value can be a Decimal or Integer literal or a reference to a column containing numeric values.
<i>ROUND Function</i>	Rounds input value to the nearest integer. Input can be an Integer, a Decimal, a column reference, or an expression. Optional second argument can be used to specify the number of digits to which to round.
<i>TRUNC Function</i>	Removes all digits to the right of the decimal point for any value. Optionally, you can specify the number of digits to which to round. Input can be an Integer, a Decimal, a column reference, or an expression.
<i>NUMFORMAT AT Function</i>	Formats a numeric set of values according to the specified number formatting. Source values can be a literal numeric value, a function returning a numeric value, or reference to a column containing an Integer or Decimal values.

Example:

For the NUMFORMAT function, you can specify the full format to which you want the numeric values in the column to conform. In the following example, all values that contain a decimal point and match with the Decimal (Float) type are forced to add a value before the decimal. This step converts values like .00 to 0.00, which standardizes the format of your numbers.

Transformation Name	Edit column with formula
Parameter: Columns	*
Parameter: Formula	IF(FIND(\$col, '.')>0, IFVALID(\$col, 'Float', NUMFORMAT(ROUND(\$col, 2), '0.00')), \$col)

- The Columns value is a wildcard, which in this case applies the transformation across all columns in the dataset (*).
- In the Formula, you see a nested expression, which is a bit more complicated than the preceding String example.
 - The outer IF function tests if the FIND function returns a non-zero value when searching each column value for the period (.). Values that match could possibly be decimals and require further evaluation:
 - If the value in the column is valid against the Decimal (Float) data type then do the following:
 - ROUND the value to two decimal points. For more information, see *ROUND Function*.
 - Format the value in the following manner:

0.00

- The above format includes the two decimal points to which you rounded, adding any extra zeros if they are not present in the input rounded value.
- Additionally, another zero is inserted in front of the decimal if it is missing in the output of the ROUND function.
- For more information on number formats, see *NUMFORMAT Function*.

For more information, see *Normalize Numeric Values*.

Functions for dates

Since dates are structured patterns of string-based data, the best approach is to begin by using the Patterns tab in the Column Details panel. See below.

For more detailed modifications, you can specify formatting strings that are applied as part of the DATEFORMAT function to the dates in your column.

Function	Description
----------	-------------

DATEFORMAT
Function

Formats a specified Datetime set of values according to the specified date format. Source values can be a reference to a column containing Datetime values.

For more information including examples on the DATEFORMAT function, see *Format Dates*.

Custom Data Types

You can create custom datatypes to use as a form of standardization. Values in a column that do not conform to the custom type are flagged as invalid and can be triaged accordingly.

NOTE: A custom data type does not inherently provide a means of standardizing the values. The values flagged as invalid must be converted to valid values or removed.

Custom type using a regular expression

A custom data type can be created based on a user-defined regular expression.

NOTE: Regular expressions are powerful tools for creating matching patterns. They are considered developer tools.

For more information, see *Create Custom Data Types Using RegEx*.

Overview of Cluster Clean

Contents:

- *Example - Multiple methods of clustering*
 - *Clustering Algorithms*
 - *Similar strings*
 - *Pronunciation*
 - *Job Execution*
 - *Disable*
-

Cluster clean enables users of Trifacta® to standardize values in a column by clustering similar values together. Using one of the supported matching algorithms, Trifacta can cluster together similar column values. You can review the clusters of values to determine if they should be mapped to the same value. If so, you can apply the mapping of these values within the application.

- For more information on how to apply cluster clean, see *Standardize Page*.
- For more information on other methods of standardization, see *Overview of Standardization*.

Artifacts:

When a cluster clean step is added to your recipe, the number of individual changes can be many megabytes of data. Instead of storing these objects within the recipe definition, they are stored as a set of artifacts in the artifact storage database and referenced from the recipe.

- These artifacts exist outside the scope of the recipe file.
- These artifacts must be stored in a Trifacta database for the step to be editable and exportable.

NOTE: If the artifact storage service is disabled, this feature is unusable.

- When a flow is exported, an `artifact.data` file is included as part of the export. This file must be imported with the flow definition, or the cluster clean step in the imported flow is broken. For more information, see *Export Flow*.

Example - Multiple methods of clustering

Source:

The following dataset includes some values that could be standardized:

RowId	Values
Row01	Apple
Row02	pear
Row03	apple
Row04	pair
Row05	Äpple
Row06	pare

When you standardize using a spelling-based algorithm, the following values are clustered:

--	--

Source Value	New Value
Apple	
apple	
Äpple	
	Unclustered values
pear	
pair	
pare	

After you select the cluster of values at top, you can enter `apple`, in the right context panel to replace that cluster of values with a single string.

In the above, the unclustered values are dissimilar in spelling, but in English, they sound the same (homonyms). When you select the Pronunciation-based algorithm, these values are clustered:

Source Value	New Value
pear	
pair	
pare	
	Unclustered values
Apple	apple
apple	apple
Äpple	apple

When you select the top values clustered by pronunciation, you can enter `pear` in the right context panel.

Results:

The six source values have been reduced to two final values through two different methods of clustering. See below for more information on the clustering algorithms.

Source Value	New Value
pear	pear
pair	pear
pare	pear
Apple	apple
apple	apple
Äpple	apple

You can apply cluster-based standardization through the Standardize Page.

Clustering Algorithms

The following algorithms for clustering values are supported.

Similar strings

For comparing similar strings, the following methods can be applied:

Fingerprint

The fingerprint method compares values in the column by applying the following steps to the data before comparing and clustering:

NOTE: These steps are applied to an internal representation of the data. Your dataset and recipe are not changed by this comparison. Changes are only applied if you choose to modify the values and add the mapping.

1. Remove accents from characters, so that only ASCII characters remain.
2. Change all characters to lowercase.
3. Remove whitespace.
4. Split the string on punctuation, any remaining whitespace, and control characters. Remaining characters are assembled into groups called **tokens**.
5. Sort the tokens and remove any duplicates.
6. Join the tokens back together.
7. Compare all tokenized values in the column for purposes of clustering.

Fingerprint Ngram

This method follows the same steps as those listed above, except that tokens are broken up based on a specific (N) number of characters. By default, Trifacta uses 2-character tokens.

Tip: This method can provide higher fidelity matching, although there may be performance impacts on columns with a high number of unique values.

Pronunciation

Values are clustered based on a language-independent pronunciation.

This method uses the double metaphone algorithm for string comparison. For more information, see *Compare Strings*.

Job Execution

When a job is executed, clustering that has been applied through the data grid is applied to the full dataset. Implications:

- If you have auto-standardized values, the most common value that is applied during job execution is the value that appeared most frequently in the sample that was displayed when the cluster clean step was defined. The most common value is not redetermined based on the entire dataset.
- Values that were not part of the displayed sample may not be factored in the standardization process during job execution.

Disable

This feature is enabled by default. To disable, please complete the following steps.

Steps:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`.
For more information, see *Platform Configuration Methods*.
2. Locate the following setting and set it to `false`.

```
"feature.columnStandardization.enabled"
```

3. Save changes and restart the platform.

Overview of Visual Profiling

Contents:

- *Uses*
 - *Example*
 - *Visual Profiling Interfaces*
 - *Data Grid*
 - *Column Details*
 - *Pattern Profiling*
 - *Job Details*
 - *Enable*
 - *Profiling Engine*
 - *Exact vs. Approximate Metrics in Visual Profiles*
-

In Trifacta®, **visual profiling** provides real-time interactive visualizations of your dataset to assist in the discovery, cleansing, and transformation of your data. Visual representations are required for interpreting large volumes of data, and the platform's innovative profiling techniques visualize key statistical information in a dynamic, easy-to-consume format for faster transformation.

- At the individual column level, visual profiles provide interactive statistical information visualized in an appropriate manner for the data type. For example, columns of Zip Code data type can be represented on a geographical map of the United States.
- All visual profiles are interactive, so you can dig into the details of the data. Select one or more elements in a profile, and you can take immediate action on the data, either through steps you define or through transform recommendations provided by the platform.
- The Transformer page displays a set of recommended actions to take based on the values, rows, or columns that you select in the data grid. These recommendations are motivated by platform logic and prior usage information. For more information, see *Overview of Predictive Transformation*.

Visual profiles are available while you transform your data in the Transformer page, when you dig into the detail of individual columns, and after you execute your job at scale. Each of these interfaces has different usage patterns designed to accelerate and simplify data transformation for that specific area of the process.

Uses

- **Locate anomalies.** Visual profiling surfaces missing or invalid data in individual columns. These values can then be selected and transformed as needed.
- **Identify distributions.** In the data grid, you can review value distribution for each column in your dataset. When exploring the column details, you can also identify and select statistical outliers among your column data.
- **Identify areas for further refinement.** After a job has completed, you can review its visual profile through the application and then take action on problematic data.

Example

In the following example, a dataset containing address information has been loaded in the Transformer page:

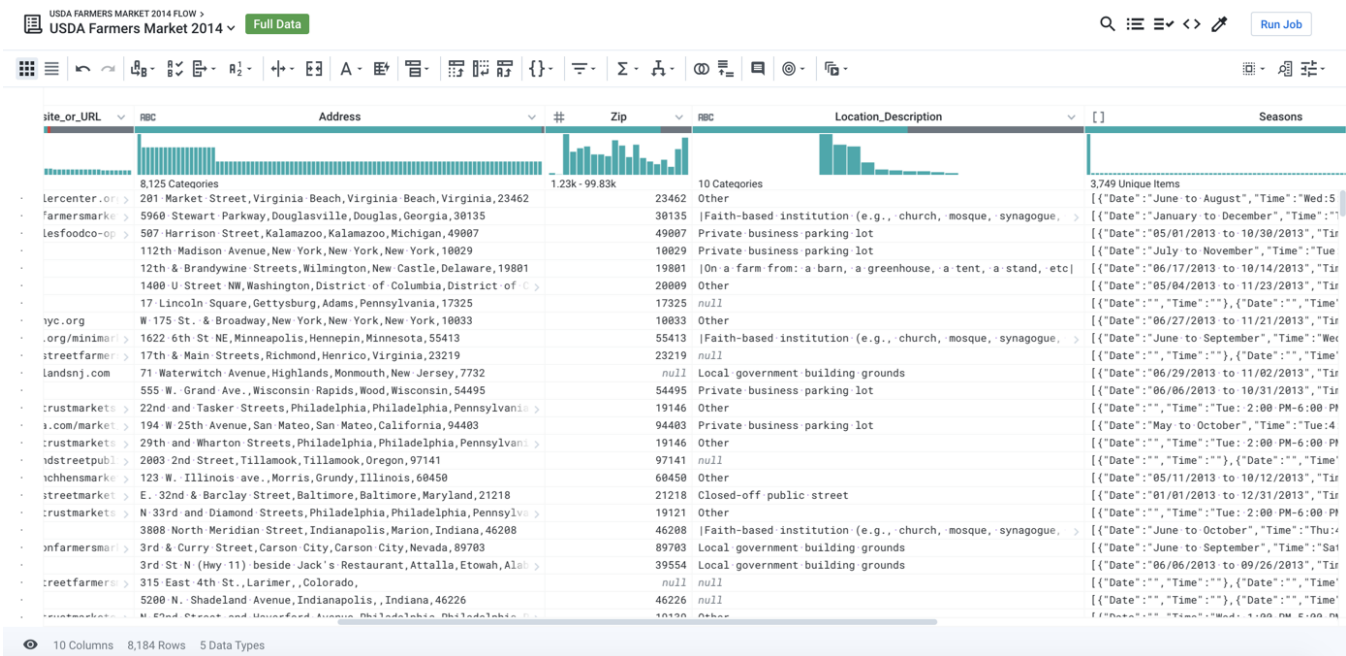


Figure: Example dataset

In this example, we are interested in exploring geographic information. From the column drop-down for the Zip column, you select **Column Details**.

Explore detail on demand. Generate visual profiles from the column drop-down.

When you explore the column details of the new column, you can see the following representation of the data:

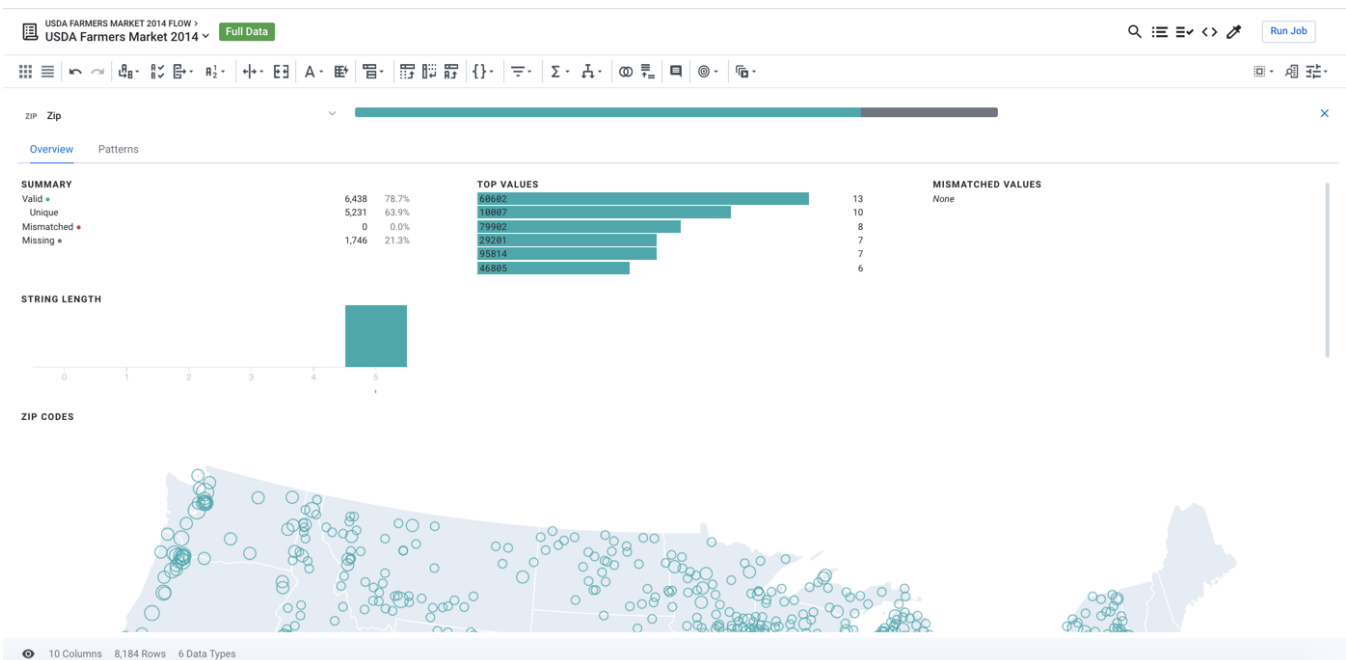


Figure: Zip Code data type represented as a U.S. map

In this case, the values in your Zip column are recognized as being of Zipcode data type. The application then represents these values as a U.S. map, which quickly renders numeric data into a format that's much easier to read and analyze.

Type-specific visualizations. The profile of the column values is represented in a type-specific visualization to assist in rapid analyzing and taking action on some or all values in the column.

Visual Profiling Interfaces

Wherever you can interact with data, visual profiling simplifies the process.

Customized visualizations. Each interface has been optimized for the scope of the data it is visualizing, whether the data is a single column, the entire sample of a dataset, or generated results.

Data Grid

In the Transformer page, the **data grid** is a tabular representation of a sample of your dataset. It is the primary interface through which you build your transformation recipes. Profiling tools:

- **Data Quality Bar:** At the top of each column, you can see graphs counting the missing, invalid, and valid values for the column's current data type. Select one of the categories, and you can take immediate action on all of the category's values in the column.
- **Column Histogram:** Individual values in the column are represented in a histogram at the top of the column. You can select one or more of these values, review relevant data, and take action.
- See *Data Grid Panel*.

Whenever a transform is selected or specified, a preview of its effects is displayed in the data grid, including any changes to the data quality bar and column histogram of affected columns. See *Transform Preview*.

For additional details on visual transformation, see *Transform Basics*.

Column Details

Through the Transformer page, you can explore statistical details about individual columns, visually represented based on the column's data type. From the drop-down for any column, select **Column Details**.

In this interface, you can review the range of values in the column and can optionally select one or more values from other columns to see which values in the current column apply. The visualizations for a column depend on the data type.

See *Column Details Panel*.

Pattern Profiling

In the Column Details panel, you can review profiling of patterns detected in the values for the selected column. These patterns can be selected, which identifies the relevant values in the column that match the pattern. You can then use these selections as the basis for building transforms that apply to the matching values.

Job Details

After the application has successfully executed a job for which profiling is enabled, you can explore a visualization of the generated dataset in the Job Details page. You can download your visual profile and results of your data quality rules on the entire dataset in PDF and JSON format.

For more information on data quality rules, see *Overview of Data Quality*.

For more information on job details, see *Job Details Page*.

Enable

Visual profiling is enabled on a per-job basis. See *Run Job Page*.

Profiling Engine

Decoupled from the user interface, the profiling engine performs the calculations required to power the visualizations before job execution and after the job results have been generated.

- In the Transformer page, the profile engine is called for incremental changes whenever a step is added to your recipe, so that you can see immediate updates to the visual profile for each column. It utilizes separate algorithms for generating the data quality bars, column histograms, value counts, frequency distributions, and other relevant statistics. When you dig into the column details, the visual profile is up-to-date and can be updated again based on your selections in that interface.
- During job execution, it is queried as a separate job when profiling is executed across the entire dataset.

NOTE: When you choose to profile your results, you are creating two distinct tasks: 1) run your transform recipe against your source and 2) profile the results. Due to the computational complexity of generating the interactive results, a profiling task often takes longer to complete than a transformation task and is therefore an optional element of a job run.

Exact vs. Approximate Metrics in Visual Profiles

Generally, visual profiles represented in the user interface, in places like column histograms and column details, are exact measurements against the current sample.

On generated results, visual profiles tend to favor approximations.

NOTE: The computational cost of generating exact visual profiling measurements on large datasets in interactive visual profiles severely impacts performance. Depending on the environment, you may choose to run profiling jobs on generated results as separate jobs. For more information on enabling this feature, see *Profiling Options*.

Below, you can review details on how metrics are calculated in visual profiling performed in different areas of the platform.

NOTE: It is not possible to calculate the accuracy of approximate calculations due to variations in dataset sizes and job execution environments.

User Interface

The UI utilizes the local running environment when displaying visual profiles on sampled data.

NOTE: Profiles are executed on the currently sampled data. Results may vary when the full transformation job is executed.

Metric Type	Measurement
Frequency (top-k)	Exact

Unique value counts	Exact
Numerical histograms	Exact
Simple statistics (mean, stdev, min, max)	Exact
Quartiles	Exact

Trifacta Photon Running Environment

Metric Type	Measurement
Frequency (top-k)	Approximate
Numerical histograms	Approximate
Simple statistics (mean, stdev, min, max)	Exact
Quartiles	Exact

Spark Running Environment

For profiling jobs, the Spark running environment is used for Spark transformation jobs.

Optionally, profiling jobs may be run on Spark for all jobs, regardless of running environment. For more information, see *Profiling Options*.

Metric Type	Measurement
Frequency (top-k)	Approximate
Numerical histograms	Approximate
Simple statistics (mean, stdev, min, max)	Exact
Quartiles	Approximate

Snowflake

For jobs executed in Snowflake, profiling jobs may also be executed in Snowflake.

NOTE: The option to pushdown profiling to Snowflake is selected for individual flows and is only applied if the job successfully executes on Snowflake. Additional limitations may apply. For more information, see *Flow Optimization Settings Dialog*.

NOTE: In Snowflake, calculations of quartiles uses a different algorithm than the same calculations in Spark. Some differences in values should be expected.

Metric Type	Measurement
Frequency (top-k)	Approximate
Numerical histograms	Approximate
Simple statistics (mean, stdev, min, max)	Exact
Quartiles	Approximate

Overview of Sampling

Contents:

- *How Sampling Works*
 - *Initial Data*
 - *Generating samples*
 - *Changing sample sizes*
 - *Important notes on sampling*
 - *Parameterization of samples*
 - *Samples management*
 - *Cancel Sample Jobs*
 - *Choosing Samples*
 - *Limitations*
 - *Sample Invalidation*
 - *Best Practices*
 - *Sampling checkpointing*
 - *Sample Types*
-

To prevent overwhelming the client or significantly impacting performance, Trifacta® generates one or more samples of the data for display and manipulation in the client application. Since Trifacta supports a variety of clients and use cases, you can change the size of samples, the scope of the sample, and the method by which the sample is created. This section provides background information on how the product manages dataset sampling.

How Sampling Works

NOTE: Generated samples are created by executing jobs on the applicable running environment. Quick Scan samples are executed in Trifacta Photon. Full Scan samples are generated in the applicable running environment on the cluster. Each running environment has a proprietary method of calculating the available volume of data in memory which is used for executing the sampling job that is launched in the running environment. As a result, the number of rows returned for the same sample type across different running environments can vary significantly.

Initial Data

When a dataset is first created, a background job begins to generate a sample using the first set of rows of the dataset. This **initial data** sample is usually very quick to generate, so that you can get to work right away on your transformations.

- The default sample is the initial sample.
- If your source of data is a directory containing multiple files, the initial sample for the combined dataset is generated from the first set of rows in the first filename listed in the directory.
 - The maximum number of files in a directory that can be read in the initial sample is limited by parameter for performance reasons.
- For more information, see *Workspace Settings Page*.
- If you are wrangling a dataset with parameters, the initial sample loaded in the Transformer page is taken from the first matching dataset.
- If the matching file is a multi-sheet Excel file, the sample is taken from the first sheet in the file.
- By default, each initial sample is either:

- 10 MB in size
- Limited by the maximum number of files
- The entire dataset
- If the source data is larger than 10MB in size, a random sample is automatically generated for you when the recipe is first loaded in the Transformer page.
 - The initial sample is selected by default. When the automatic random sample has finished generation, it can be manually selected for display.

Generating samples

Additional samples can be generated from the context panel on the right side of the Transformer page. Sample jobs are independent job executions. When a sample job succeeds or fails, a notification is displayed for you.

As you develop your recipe, you might need to take new samples of the data. For example, you might need to focus on the mismatched or invalid values that appear in a single column. Through the Transformer page, you can specify the type of sample that you wish to create and initiate the job to create the sample. This sampling job occurs in the background.

You can create a new sample at any time. When a sample is created, it is stored within your storage directory on the backend datastore.

NOTE: The Initial Data sample contains raw data from the source. Any generated sample is stored in JSONLines format with additional metadata on the sample. These different storage formats can result in differences between initial and generated sample sizes.

For more information on creating samples, see *Samples Panel*.

Sampling methods

Depending on the type of sample you select, it may be generated based on one of the following methods, in increasing order of time to create:

1. on a specified set of rows (firstrows)
2. on a quick scan across the dataset
 - a. By default, Quick Scan samples are executed on the Trifacta Photon running environment.
 - b. If Trifacta Photon is not available or is disabled, the Trifacta application attempts to execute the Quick Scan sample on an available clustered running environment.
 - c. If the clustered running environment is not available or doesn't support Quick Scan sampling, then the Quick Scan sample job fails.
3. on a full scan of the entire dataset
 - a. Full Scan samples are executed in the cluster running environment.

Sampling mechanics

When a non-initial sample is executed for a single dataset-recipe combination, the following steps occur:

1. All of the steps of the recipe are executed on the dataset on the backend cluster, up to the currently selected recipe step.
2. The generated sample is executed on the current state of the dataset.

NOTE: When a sample is executed from the Samples panel, it is launched based on the steps leading up to current location in the recipe steps. For example, if your recipe includes joining in other datasets, those steps are executed, and the sample is generated with dependencies on these other datasets. As a result, if you change your recipe steps that occur before the step where the sample was generated, you can invalidate your sample. More information is available below.

When your flow contains multiple datasets and flows, all of the preceding steps leading up to the currently selected step of the recipe are executed, which can mean:

- The number of datasets that must be accessed increases.
- The number of recipe steps that must be executed on the backend increases.
- The time to process the sampling job increases.

Implications:

- When the sample is displayed in the Transformer page, all steps after the one from which it was executed are computed in the web browser. So, if you have a lengthy series of steps or complex operations after the step where you generated a sample, you can cause performance issues of the Transformer page, including the occasional browser crash. Try generating a new sample later in your flow for better performance.
- If you have added an expensive transformation step, such as a complex union or join, you can improve performance of the Transformer page by generating and using a new sample after the transformation step.

NOTE: When a flow is shared, its samples are shared with other users. However, if those users do not have access to the underlying files that back a sample, they do not have access to the sample and must create their own.

Changing sample sizes

If needed, you can change the size of samples that are loaded into the browser your current recipe. You may need to reduce these sizes if you are experiencing performance problems or memory issues in the browser. For more information, see *Change Recipe Sample Size*.

Important notes on sampling

- Depending on the running environment, sampling jobs may incur costs. These costs may vary between Trifacta Photon and your clustered running environments, depending on type of sample and cost of job execution.
- When sampling from compressed data, the data is uncompressed and then expanded. As a result, the sample size reflects the uncompressed data.
- Changes to preceding steps that alter the number of rows or columns in your dataset can invalidate the current sample, which means that the sample is no longer a valid representation of the state of the dataset in the recipe. In this case, Trifacta automatically switches you back to the most recently collected sample that is currently valid. Details are below.

Parameterization of samples

Any parameters that are associated with your dataset can be applied to sampling:

- **Parameters:** Subsequent samples generated from the Transformer page are sampled across all datasets matched by parameter values.
- **Variables:** You can apply override values to the defaults for your dataset's variables at sample execution time. In this manner, you can draw your samples from specific sources files within your dataset with parameters.

Samples management

After you have created a sample, you cannot delete it through the application.

NOTE: Trifacta does not delete samples after they have been created. If you are concerned about data accumulation, you should configure periodic purges of the appropriate directories on the base storage layer. For more information, please contact your IT administrator.

For more information, see *Sample Jobs Page*.

Cancel Sample Jobs

Generating a sample can consume significant time, system resources, and in some deployments cost. As needed, you can cancel a sample job that is in progress in either of the following ways:

- Locate the in-progress sampling job in the Samples panel. Click X.
- Click the Jobs icon in the left nav bar. Select **Sample jobs**. For more information, see *Sample Jobs Page*.

Choosing Samples

After you have collected multiple samples of multiple types on your dataset, you can choose the proper sample to use for your current task, based on:

1. **How well each sample represents the underlying dataset.** Does the current sample reflect the likely statistics and outliers of the entire dataset at scale?
2. **How well each sample supports your next recipe step.** If you're developing steps for managing bad data or outliers, for example, you may need to choose a different sample.

Tip: You can begin work on an outdated yet still valid sample while you generate a new one based on the current recipe.

Limitations

- Some advanced sampling options are available only with execution across a scan of the full dataset.
- Undo/redo do not change the sample state, even if the sample becomes invalid.
- When a new sample is generated, any Sort transformations that have been applied previously must be re-applied. Depending on the type of output, sort order may not be preserved.
- Samples taken from a dataset with parameters are limited to a maximum of 50 files when executed on the Trifacta Photon running environment. You can modify parameters as they apply to sampling jobs. See *Samples Panel*.

Sample Invalidation

With each step that is added or modified to your recipe, Trifacta checks to see if the current sample is valid. Samples are valid based on the state of your flow and recipe at the step when the sample was collected. If you add steps before the step where it was created, the currently active sample can be invalidated. For example, if you change the source of data, then the sample in the Transformer page no longer applies, and a new sample must be displayed.

Tip: After you have completed a step that significantly changes the number of rows, columns, or both in your dataset, you may need to generate a new sample, factoring in any costs associated with running the job. Performance costs may be displayed in the Transformer page.

NOTE: If you modify a SQL statement for an imported dataset, any samples based on the old SQL statement are invalidated.

- The Transformer page reverts to displaying the most recently collected sample that is currently valid.
- You can generate a new sample of the same type through the Samples panel. If no sample is valid, you must generate a new sample before you can open the dataset.

- A sample that is invalidated is listed under the Unavailable tab. It cannot be selected for use. If subsequent steps make it valid again, it re-appears in the Available tab.

Best Practices

The data that is displayed in the data grid is based on all of the upstream samples after which all subsequent steps in each upstream recipe are performed in the browser. If you have a large number of steps or complex steps between the recipe locations for your samples in use and your current recipe location, you may experience performance slow-downs or crashes in the data grid. For more information on sampling best practices, see <https://community.trifacta.com/s/article/Best-Practices-Managing-Samples-in-Complex-Flows>.

Sampling checkpointing

All steps between the step in your current sample and the currently displayed step must be computed in the browser. As you build more complex recipes, it's a good idea to create samples at various steps in your recipe, particularly after you have executed a complex step. This type of **sample checkpointing** can improve overall performance.

For example, as soon as you load a new recipe, you should take a sample, which can speed up the process of loading.

Tip: You can annotate your recipe with comments, such as: `sample: random` and then create a new sample at that location.

Sample Types

For more information on sample types, see *Sample Types*.

Overview of Job Execution

Contents:

- *Jobs Types*
 - *Transformation job types*
 - *Other job types*
 - *Basic Process for Transformation Jobs*
 - *Job preparation*
 - *Job execution*
 - *Job monitoring*
 - *Job cleanup*
 - *Scheduled jobs*
 - *Job Execution Performance*
 - *Job logs*
 - *Running Environments*
-

This section provides an overview of how jobs of various types are initiated, managed, and executed in Trifacta®. You can also review summaries of the available running environments for your product edition.

NOTE: During job execution of any kind, Trifacta never modifies source data. All transformation is performed on requested elements of the data. If the data needs to be retained for any period of time during use or transformation, it is stored in the browser or in the base storage layer. After the data has been used for the intended purpose, it is removed from temporary storage.

When you build your recipe in the Trifacta application, you can see in real-time the effects of the transformations that you are creating. When you wish to produce result sets of these transformations, you must run a job, which performs a separate set of execution steps on the data. Job execution is a separate process for the following reasons:

- In the Transformer page, you are working with a sample of your data. For larger volumes of data, the entire dataset cannot be represented in the browser effectively. So, to apply your recipe to the entire dataset, a separate set of actions must be performed.
- When working with large datasets, you need a running environment on a multi-node cluster that has been designed for parallel processing. Modern running environments are designed to break up data transformation jobs into separate pieces, each of which can be executed on a separate node and then returned to be assembled with the other job parts into the finished result set.
- Job execution can occur asynchronously. When you launch a job, a separate lightweight process assembles the necessary pieces for the job to be executed and then distributes these pieces accordingly. You can continue to work in the Transformer page while your results are being prepared with minimal impact on Trifacta application performance or your user experience.

Other features of job execution:

- Change the format of the source to a different format in the output.
- Change the location where the results are generated.
- Change file-based source data into table-based relational data on the output.
- Write multiple versions of the output at the same time.
- Jobs can also be scheduled.
- Jobs can also be executed using REST APIs, which enables automation of job execution. For more information on job execution via API, see *API Workflow - Run Job*.

Jobs Types

The following types of jobs can be executed as part of normal operations of the product.

Job locations:

- A **local job** is one that is executed on the Trifacta node using services that are hosted on it.
- A **remote job** is executed through services that are not hosted on the Trifacta node.

Transformation job types

Informally, a "job" is considered any action that is performed on a set of data. Commonly, jobs refer to the process of transforming source data into output results. However:

- Transformation jobs are composed of a number of sub-jobs, which handle things like ingestion of data, transformation, and writing of results.
- In addition to jobs that transform data, there are other types of jobs. Discussed later.

Job groups:

For transformation job types, the following terms apply:

- Internal to the product, a job that is executed on one or more recipes in a flow is called a **jobGroup**.
- A jobGroup is composed of one or more of the job types listed below. Internal to the platform, these are called **jobs**.

The following diagram illustrates how these job types are related.

```
+ myJob jobGroup
+   Connect job
+   Request job
+   Ingest job
+   Transform job
+   Transfer job
+   Process job
```

Tip: You can have one or more of each of these job types as part of a single jobGroup.

Connect

A Connect job performs the steps necessary to connection the Trifacta application to the datastore that contains source data. These jobs use the connection objects that are native to the platform or that you have created to make the connection to your imported datasets.

NOTE: Depending on the running environment, a Connect job may time out after a period of inactivity or failure to connect, and it may be retried one or more times before the job is marked as failed.

Request

A Request job sends a query or other request to the source datastore for the assets specified in the imported datasets.

Ingest

Requested data is brought from the external source to the execution layer, which is the temporary storage location as defined for the running environment.

Convert

Some formats supported for import are not natively understood by the product. These formats must be converted to a format that the platform can quickly process. This process typically converts binary formats, such as XLS or PDF, into CSV files that are stored temporarily in the base storage layer for purposes of job execution. After the job has succeeded or failed, these converted files are removed.

Transform

After data has been requested and ingested (if needed), a Transform job converts the steps of a recipe into an intermediate scripted format (called CDF). The CDF script is then passed to the appropriate running environment for transformation of the source data. Additional details are provided later.

Prepare

If the specified job is publishing results to a connection other than the base storage layer, the results are initially prepared on the base storage layer, after which they are written to the target datastore.

This job type does not apply when the base storage layer is the final destination for the results.

Transfer

A Transfer job writes the results to the appropriate output location, as specified by the output objects referenced when the job was launched.

Process

When the transfer is complete, a Process job performs final cleanup, including removal of temp files such as intermediate results written to the base storage layer.

Other job types

Profiling

When you execute a transformation job, you can optionally choose to create a visual profile of the results of that job. Visual profiling is a separate job that sometimes takes longer to execute than the job itself, but a visual profile can be useful in highlighting characteristics of your data, including metrics and errors on individual columns.

Visual profiles are available for review in the Job Details page. You can also download PDF or JSON versions of your visual profile.

For more information on visual profiling, see *Overview of Visual Profiling*.

Sampling

When you are interacting with your source data to transform it through the browser, you are working on a sample of the data. As needed, you can take new samples of the data to provide different perspectives on it. Also, for longer and more complex flows, you should get in the habit of taking periodic samples, which can improve performance in the browser.

Through the Samples panel, you can launch a job to collect a new sample of your data. There are multiple types of sampling, which can be executed using one of the following methods:

- **Quick scan:** These sample types are performed based on a scan a limited number of rows of your data.

- These samples are based on the first set of rows that are accessible and are quick to execute. However, they cannot pick up in the sample any rows that are deeper in your datasets. For example, if your source data contains multiple files, quick scan samples might not contain any data from the second or later files.
- These samples are executed in Trifacta Photon.
- **Full scan:** A full scan sample is executed across the entire available dataset.
 - Depending on the size of your dataset, this scanning and sample process can take a while to execute on a large dataset.
 - These samples are executed on the clustered running environment with which the Trifacta application is connected.

For more information, see *Overview of Sampling*.

Basic Process for Transformation Jobs

A transformation job is run based on the outputs that you are trying to generate. For a selected output, the executed job runs the transformations for all of the recipes between the output and all of its imported datasets. For example, generation of a single output could require the transformation of five different recipes that use 13 different imported datasets.

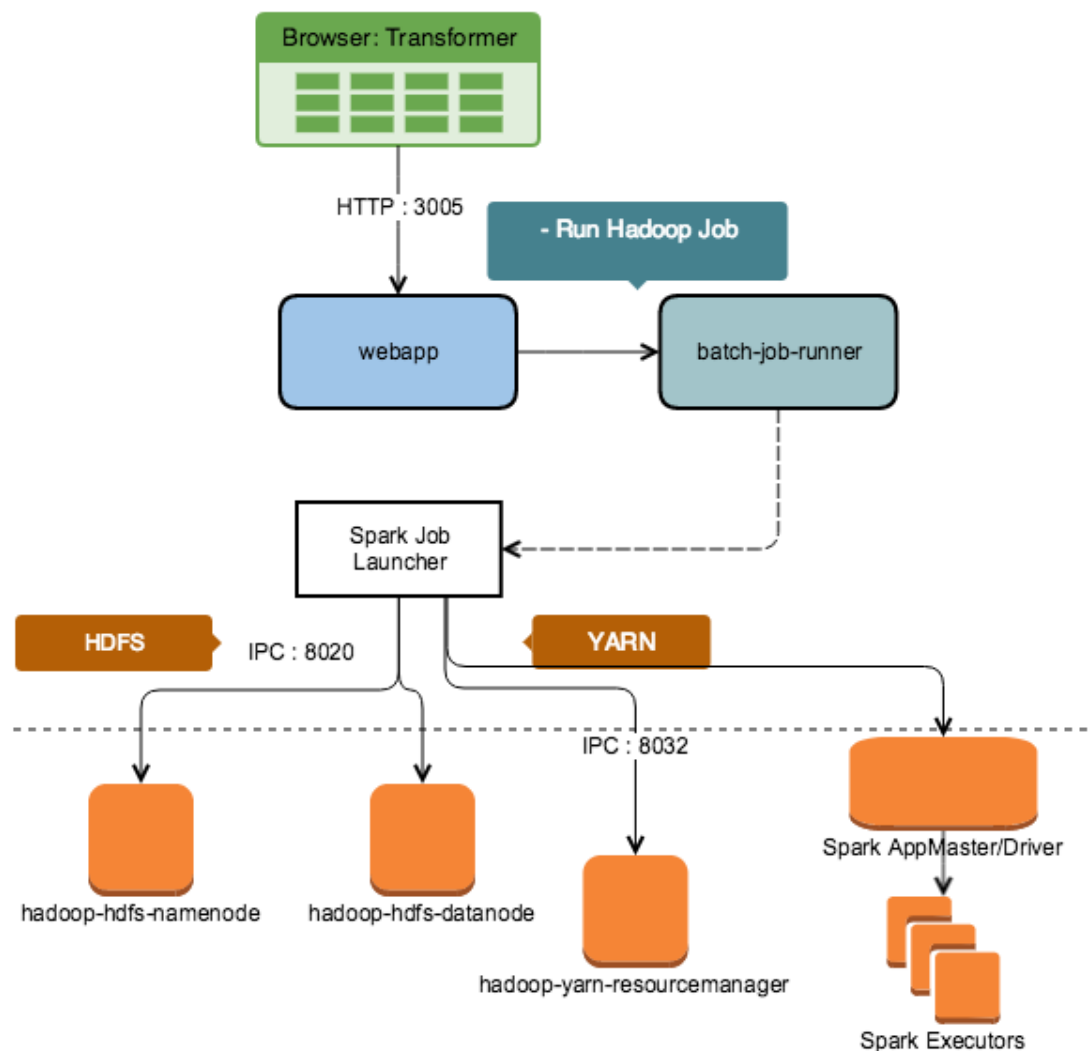


Figure: Flow chart on how a Spark-based job is executed on a Hadoop cluster

Steps:

1. **Job preparation:** A jobGroup entry is created in the database, and a job execution graph is created for each job within the jobGroup and submitted to the batch job runner service. This service requests an expanded version of the recipe in CDF format from the Trifacta application. This and other assets are placed in a queue for processing by the batch job runner.
2. **Job preparation:** When resources are available, the job is pulled from the queue and submitted to the resource coordinator of the batch job runner for execution on Trifacta Photon or the remote running environment. The job is placed in another queue for execution on the appropriate running environment.
3. **Job execution:** When cluster resources are available, the job definition, CDF script and other resources are submitted to the resource coordinating process for the selected running environment (In the above diagram, this is the Spark Job Launcher, which coordinates with YARN). This process submits parts of the job to separate nodes on the cluster for execution. Periodically, batch job runner polls this process for status. When all nodes of the cluster have completed their execution, the job results are written to the designated location, and batch job runner finishes the job execution by updating the Jobs database.

Job preparation

When you initiate a job through the Trifacta application, the following steps occur:

1. A jobGroup is created in the database. It consists of the specification of one or more jobs, as described above.
2. The recipe whose output is being executed is requested from the Trifacta database. This recipe is expanded from storage format and later is stored temporarily in the database for reference.
3. The Trifacta application verifies access to data sources and output locations.
4. A job execution graph (flow chart) is created for the various jobs required to complete execution of the jobGroup.
 - a. This graph includes jobs for ingest, transformation, conversion, and other steps, as described above.
5. The graph is sent to the batch job runner service in the platform. This service manages the submission, tracking, and completion of all jobs to supported running environments.
6. Batch job runner requests to the Trifacta application to return a Common Dataflow Format (CDF) version of the expanded recipe.
 - a. CDF is a domain-specific language for data transformation that runs anywhere that supports Python execution.
 - b. Wrangle is compiled into CDF format at execution time. This CDF script is delivered to the running environment for execution.
 - c. CDF scripts are internal to the platform and are not accessible to users of the platform.
7. Depending on the running environment, additional modifications to the CDF script may be made before the job is submitted.
8. The batch job runner places the job in a queue for submission to the running environment.

Job execution

When the job is ready to be pulled from the queue, the following tasks are completed:

1. The job definition, CDF script, and associated resources are submitted to the resource coordinating process of the running environment.
 - a. This coordinator is the batch job runner for local jobs or a dedicated service on remote running environments.
 - b. For example, for EMR execution, which is a remote running environment, the job is submitted to the YARN service, which manages the delegation of work tasks to the various nodes in the cluster.
 - c. In the resource coordinator, jobs from the product are labeled as `Trifacta Transformer` or `Trifacta Profiler` (for profiling jobs).
2. Periodically, batch job runner polls the running environment for status on job execution.
 - a. This status information is stored and updated in the Jobs database.
3. The Trifacta application queries the Jobs database for updated information.

- a. These updates are stored in the Trifacta databases for internal services to access to present updates.
 - b. Updates can appear in Flow View page and also in the Jobs and Job Details page, so that you can track progress.
4. During execution, the resource manager arranges for the delivery of data and CDF script objects to nodes of the cluster.
 - a. On these individual nodes, portions of the data are processed through the CDF script.
 - b. The results of this processing is messaged back to the resource manager.
 - c. When all of the nodes have reported back that the job processing has been completed, results are written to the location or locations as defined in the output object that was selected during job execution.
5. Batch job runner updates any available job logs as needed based on the results of the job execution. These logs may be available through the Trifacta application.

Job monitoring

Transformation jobs: After a transformation job has been launched, you can monitor the state of the job as it passes through separate stages in the process.

- In Flow View, click the output object. Then, click the Jobs tab. See *Flow View Page*.
- In the Jobs page, you can hover over the status of the job to gather more information. See *Jobs Page*.
- Additional information may be available in the Job Details page. See *Job Details Page*.

Sample jobs: In-progress sampling jobs can be tracked through the following locations:

- After you have initiated a sample job through the Samples panel, you can track progress there. See *Samples Panel*.
- All of your sample jobs are available through the Trifacta application. See *Sample Jobs Page*.

Plan runs: When you have launched jobs as part of a plan run, you can track progress through the Trifacta application. A plan run may consist of flow-based transformation jobs, as well as other tasks.

See *Plan Runs Page*.

Job cleanup

After the results have been written, the following tasks are completed:

1. Applicable job logs are updated and written to the appropriate location.
2. The expanded recipe stored in the database is removed.
3. Any temporary files written to the base storage layer are removed.

Scheduled jobs

You can also schedule the execution of jobs within your flows. This process works as follows:

1. In Flow View, you define the outputs that you wish to deliver when the flow is executed according to a schedule. These outputs are different objects that the outputs you create from your recipes, but you can define them to write to the same locations.
2. You specify the schedule for when the job is to be executed. Date and time information, as well as frequency of execution, can be defined within the flow.

When the specified time is reached, the job is queued for execution, as described above. For more information, see *Overview of Automator*.

Job Execution Performance

Job execution is a resource-intensive and multi-layered process that transforms data of potentially limitless size. The following factors can affect performance in the Trifacta application and during job execution:

- **Long or complex recipes**
 - Consider breaking recipes into smaller steps. You can change recipes together.
- **Number of columns in your data**
 - The entire width of a dataset must be represented in the sample.
 - Delete unnecessary columns early in your recipe.

Tip: If your data is sourced in relational systems, you can apply optimizations to your imported datasets to pre-filter out columns in your dataset before they are ingested into the system. See *Flow Optimization Settings Dialog*.

You can also use custom SQL statements to collect only the columns that are needed from source tables. See *Create Dataset with SQL*.

- **Complexity of transformations**
 - Transformations that blend datasets (join and union) or that perform complex transformations on your dataset (aggregate, window, pivot, etc.) can be expensive to process.
 - If your recipe contains too many of them, it can negatively impact job processing. Consider breaking these across multiple recipes instead.

Job logs

Separate log files are maintained for each jobGroup. As needed, you can acquire these logs from the Trifacta application. In the Job Details page, select **Download logs** from the context menu for a job entry. For more information, see *Job Details Page*.

If there are issues with job execution that cannot be resolved by reviewing the job log, workspace administrators can download a support bundle, which contains additional log information from the platform. For more information, see *Support Bundle Contents*.

Running Environments

Trifacta Photon Running Environment

Trifacta Photon is an in-memory running environment that is hosted on the Trifacta node. This environment is initialized only when a job is queued for execution on it. Designed for small- to medium-sized jobs, it offers superior performance due to its location on the Trifacta node and its in-memory processing.

When you choose to run a job in the Trifacta application, Trifacta Photon is selected as the default running environment if it is available and the job size is estimated to small or medium.

Tip: Trifacta Photon is also used for sampling jobs that are configured to use the Quick Scan method. For more information, see *Overview of Sampling*.

Tip: In the Run Job page, select **Photon** to run the job on this running environment.

Trifacta Photon is enabled by default but can be disabled as needed.

NOTE: Trifacta Photon cannot process numeric values with more than 16 digits. Columns containing such values are converted to String values, and the digits beyond 16 are converted to 0.

NOTE: When a recipe containing a user-defined function is applied to text data, any null characters cause records to be truncated during Trifacta Photon job execution. In these cases, please execute the job on Spark.

NOTE: For more information on configuring Trifacta Photon, see *Configure Photon Running Environment*.

EMR Running Environment

Elastic Map Reduce is a service of Amazon Web Service (AWS) for processing large volumes of data using open source technologies such as Spark. EMR integrates easily with other AWS-based services such as S3, IAM, Glue, and more.

When Trifacta is installed in an EC2 instance on AWS, the Trifacta application can be integrated with either pre-existing or new EMR clusters for supported versions of EMR. Additional configuration and limitations apply. For more information, see *Configure for EMR*.

Tip: In the Run Job page, select **Spark** to run the job on this running environment when the Trifacta application has been integrated with it.

For more information, see <https://aws.amazon.com/emr>.

Snowflake Running Environment

Contents:

- *Requirements*
 - *General*
 - *For Trifacta Self-Managed Enterprise Edition*
 - *Requirements across multiple Snowflake connections*
 - *Limitations*
 - *Enable*
 - *Workspace Settings*
 - *Flow Optimizations*
 - *Run Job*
 - *To execute a job in Snowflake in the Trifacta application:*
 - *Unsupported Wrangle for Snowflake Execution*
 - *General limitations*
 - *Unsupported input data types*
 - *Unsupported Trifacta data types*
 - *Unsupported transformations*
 - *Unsupported functions*
 - *Verify Execution*
-

Snowflake provides cloud-based data storage and analytics as a service. Among other infrastructures, Snowflake runs on Amazon S3. If all of your source datasets and outputs are in Snowflake locations and other conditions are met, then the entire execution of the transformations can occur in Snowflake.

Transferring the execution steps from the Trifacta node to Snowflake yields the following benefits:

- A minimum of data (recipe steps and associated metadata) is transferred between systems. Everything else remains in Snowflake.
- Recipe steps are converted into SQL that is understandable and native to Snowflake. Execution times are much faster.
- Depending on your environment, total cost of executing the job may be lower in Snowflake.

In this scenario, the recipe steps are converted to SQL, which is sequentially executed your source data in temporary tables, from which the results that you have defined for your output are written.

Tip: When running a job in Snowflake, your data never leaves Snowflake.

Tip: Execution on datasets created with custom SQL is supported.

If the requirements and limitations are met, the Trifacta application automatically executes the job in Snowflake.

Requirements

General

- This feature must be enabled by the workspace admin. See below.
- Trifacta application must be integrated with Snowflake. See *Snowflake Connections*.
 - The permission to execute jobs in Snowflake must be enabled.
- All sources and outputs must reside in Snowflake.
- Spark + Snowflake must be selected as running environment. See *Run Job Page*.

- Jobs are executed in the virtual warehouse that is specified as part of the Snowflake connection.

NOTE: Job execution requires significantly more resources than ingest or publish jobs on Snowflake. Before you begin using Snowflake, you should verify that your Snowflake virtual warehouse has sufficient resources to handle the expected load. For more information, see *Snowflake Connections*.

- In your flow, you must enable all general and Snowflake-specific flow optimizations. When all of these optimizations are enabled, the job can be pushed down to Snowflake for execution. See "Flow Optimizations" below.

For Trifacta Self-Managed Enterprise Edition

For customer-managed deployments, the following additional requirements apply:

- **S3:** Base storage layer must be S3. See *Set Base Storage Layer*.
- **AWS running environment:** The Trifacta node must be integrated with a running environment that is compatible with AWS.
 - For more information, see *Configure for EMR*.
 - For more information, see *Configure for AWS Databricks*.

Requirements across multiple Snowflake connections

If you are executing a job on Snowflake that utilizes multiple connections, the following requirements must also be met for execution of the job on Snowflake:

- All Snowflake connections used in the job must utilize to the same Snowflake account.
- All Snowflake connections used in the job must be backed by the same Snowflake primary role. For more information, see <https://docs.snowflake.com/en/user-guide/security-access-control-overview.html#enforcement-model-the-primary-role-and-secondary-roles>

Limitations

Snowflake as a running environment requires that pushdowns be enabled for the workspace and for the specific flow for which the job is executed. If the flow and the workspace are properly configured, the job is automatically executed in Snowflake.

NOTE: Snowflake is not a running environment that you explicitly select or specify as part of a job. If all of the requirements are met, then the job is executed in Snowflake when you select EMR.

- All datasources and all outputs specified in a job must be located within Snowflake.
- All recipe steps, including all *Wrangle* functions in the recipe, must be translatable to SQL.

NOTE: When attempting to execute a job in Snowflake, Trifacta application executes each recipe in Snowflake, until it reaches a step that cannot be executed there. At that point, data is transferred to EMR, where the remainder of the job is executed.

- If the schemas have changed for your datasets, pushdown execution on Snowflake is not supported. Trifacta falls back to submitting the job through another running environment.
- Some transformations and functions are not currently supported for execution in Snowflake. See below.
- Sampling jobs are not supported for execution in Snowflake.
- If your recipe includes data quality rules, the job cannot be fully executed in Snowflake.
- Visual profiling is supported with the following conditions or requirements.
 - Visual profiles are unloaded to a stage in an S3 bucket.

- If a stage is named in the connection, it is used. This stage must point to the default S3 bucket in use.
- If no stage is named, a temporary stage is created in the `PUBLIC` schema. The connecting user must have write access to `PUBLIC`.

NOTE: Creating a temporary stage requires temporary credentials from AWS. These credentials are valid for 1 hour only. If a job is expected to run longer than one hour, you should define a named stage.

- For more information, see *Snowflake Connections*.

Enable

Workspace Settings

The following setting must be enabled in the workspace. Select **User menu > Admin console > Workspace settings**.

Optimization	Description
Logical and physical optimization of jobs	When enabled, the Trifacta application attempts to optimize job execution through logical optimizations of your recipe and physical optimizations of your recipes interactions with data.

For more information, see *Workspace Settings Page*.

Flow Optimizations

You must enable the Snowflake optimizations in your flow. In Flow View, select **More menu > Optimization settings**.

NOTE: All general optimizations must be enabled for your flow, as well as the following optimizations, which are specific to Snowflake.

Optimization	Description
Snowflake > Column pruning from source	When enabled, job execution performance is improved by removing any unused or redundant columns from the source database.
Snowflake > Filter pushdown	When this setting is enabled, the Trifacta application optimizes job performance on this flow by pushing data filters directly on the source database.
Snowflake > Full pushdown	When this setting is enabled, all supported pushdown operations, including full transformation and profiling job execution, is pushed down to Snowflake, where possible.

For more information, see *Flow Optimization Settings Dialog*.

Run Job

To execute a job in Snowflake in the **Trifacta application**:

- Your job must meet the requirements listed above.
- Your job must not include the functions, transformations, or other unsupported elements that are listed below.
- You must select `Snowflake + Spark` as your running environment in the Run Job page.

NOTE: If this running environment option does not appear in the Run Job page, then all required optimization settings have not been enabled for the workspace or the flow (see above) or the data or recipes do not meet the criteria for execution.

See *Run Job Page*.

Tip: After launching the job, you can monitor job execution through the Job Details page, which includes a link to the corresponding job in the Snowflake console.

Unsupported Wrangle for Snowflake Execution

The following transformations and functions are not currently supported for execution in Snowflake.

NOTE: If your recipe contains any of the following transformations or functions, full job execution in Snowflake is not possible at this time. These transformations are expected to be supported and removed from this list in future releases.

General limitations

For more information on limitations on specific push-downs, see *Flow Optimization Settings Dialog*.

Unsupported input data types

The following Snowflake data types are not supported for input into Trifacta:

- BINARY
- VARBINARY
- GEOGRAPHY

Unsupported Trifacta data types

None.

Unsupported transformations

The following Wrangle functions are not currently supported for execution in Snowflake.

- Standardize

Unsupported functions

The following Wrangle functions are not currently supported for execution in BigQuery.

Aggregate functions

KTHLARGEST
KTHLARGESTIF
KTHLARGESTUNIQUE
KTHLARGESTUNIQUEIF
APPROXIMATEMEDIAN
APPROXIMATEPERCENTILE
APPROXIMATEQUARTILE
QUARTILE

For more information, see *Aggregate Functions*.

Math functions

LCM
NUMVALUE

Partially supported:

NUMFORMAT: Only supported when used for rounding.

For more information, see *Math Functions*.

Date functions

NETWORKDAYS
NETWORKDAYSINTL
WORKDAY
WORKDAYINTL
KTHLARGESTDATE
KTHLARGESTUNIQUEDATE
KTHLARGESTUNIQUEDATEIF
KTHLARGESTDATEIF
EOMONTH
SERIALNUMBER

String functions

DOUBLEMETAPHONEEQUALS
TRANSLITERATE

For more information, see *String Functions*.

Type functions

Partially supported:

IFMISSING

NOTE: When the IFMISSING function immediately follows the PREV function in your recipe steps, Snowflake generates an incorrect value. This is a known issue and will be fixed in a future Snowflake release.

Window functions

SESSION

For more information, see *Window Functions*.

Verify Execution

To verify execution in Snowflake, please do the following:

Steps:

1. In the left nav bar, click the Jobs link.
2. In the Jobs page, select the job that you executed.
3. In the Overview tab, the value for Environment under the Execution summary should be: `Snowflake`.

For more information, see *Job Details Page*.

AWS Databricks Running Environment

Databricks provides the combination of data lakehouse storage, analytics processing, and artificial intelligence capabilities in a single unified platform. For job execution, the Databricks running environment can be hosted in the Azure or AWS ecosystems.

NOTE: This running environment is available only if you install Trifacta on AWS.

Tip: In the Run Job page, select **Spark (Databricks)** to run the job on this running environment when the Trifacta application has been integrated with it.

Additional configuration is required.

NOTE: Use of AWS Databricks is not supported on Marketplace installs.

NOTE: When executing a job on the AWS Databricks running environment using a relational source, the job fails if one or more columns has been dropped from the underlying source table. As a workaround, the recipe panel may show steps referencing the missing columns, which can be used to either fix the recipe or the source data.

For more information, see *Configure for AWS Databricks*.

For more information on Databricks, see <https://databricks.com/>.

Azure Databricks Running Environment

Databricks provides the combination of data lakehouse storage, analytics processing, and artificial intelligence capabilities in a single unified platform. For job execution, the Databricks running environment can be hosted in the Azure or AWS ecosystems.

NOTE: This running environment is available only if you install Trifacta on Azure.

Tip: In the Run Job page, select **Spark (Databricks)** to run the job on this running environment when the Trifacta application has been integrated with it.

Additional configuration is required.

NOTE: Use of Azure Databricks is not supported on Marketplace installs.

NOTE: When executing a job on the Azure Databricks running environment using a relational source, the job fails if one or more columns has been dropped from the underlying source table. As a workaround, the recipe panel may show steps referencing the missing columns, which can be used to either fix the recipe or the source data.

For more information, see *Configure for Azure Databricks*.

For more information on Databricks, see <https://databricks.com/>.

Hadoop Spark Running Environment

When Trifacta is installed on a supported version of Cloudera, the Trifacta application can be configured to execute larger jobs on the cluster instance of Spark. Spark leverages in-memory capabilities on individual nodes for faster processing of distributed analytics tasks, with spillover to disk as needed.

Tip: In the Run Job page, select **Spark** to run the job on this running environment when the Trifacta application has been integrated with it.

Spark requires a backend distributed storage layer:

- On AWS-based deployments, this storage layer is S3.
- On Hadoop-based deployments, this storage layer is HDFS.

Additional configuration is required.

NOTE: When executing a job on the Spark running environment using a relational source, the job fails if one or more columns has been dropped from the underlying source table. As a workaround, the recipe panel may show steps referencing the missing columns, which can be used to fix to either fix the recipe or the source data.

NOTE: The Spark running environment does not support use of multi-character delimiters for CSV outputs. You can switch your job to a different running environment or use single-character delimiters. This issue is fixed in Spark 3.0 and later. For more information on this issue, see <https://issues.apache.org/jira/browse/SPARK-24540>.

For more information, see *Configure for Spark*.

Overview of TBE

Contents:

- *Limitations*
 - *Enable*
 - *Column by Example*
 - *CBE for Datetime*
 - *Alternatives*
-

Transformation by Example (TBE) enables you to build recipe objects by mapping example output values for source values. Trifacta® then interprets the differences between the inputs and outputs to determine the transformation required to map them.

TBE leverages pattern-based matching and predictive transformation to derive transformations. When you provide explicit mappings of input value to output, the mapping is passed through predictive transformation to determine the best possible matching pattern.

- For more information on patterns, see *Overview of Pattern Matching*.
- **Predictive Transformation** is a core component of Trifacta. Based upon user input, the platform provides one or more suggestions of ways in which to transform the data.
 - In TBE, these suggestions are rendered as elements of the transformation in progress.
 - For more information, see *Overview of Predictive Transformation*.

Use cases:

Tip: TBE simplifies the process of defining patterns to match all values in your source column. Since you know and can specify the exact desired output, you can leave the details of defining the pattern or patterns required to match input to output to the product.

Transformation by Example works well in the following use cases:

- You are just getting started with the product and would like to get productive quickly to transform your data into known outputs.
- Your data has groups of values, each of which needs transformation in a different way. In a single recipe step, you can perform these transformations across all groups.
- Your data has special-case exceptions that must be transformed.

Tip: You can use this feature as a final cleanup for other transformations. If you have a transformation that handles 90% of the cases in a column, you can use this transformation to handle the remainder.

Artifacts:

When a TBE step is added to your recipe, the number of individual changes can be many megabytes of data. Instead of storing these objects within the recipe definition, they are stored as a set of artifacts in the artifact storage database and referenced from the recipe.

- These artifacts exist outside the scope of the recipe file.
- These artifacts must be stored in a Trifacta database for the step to be editable and exportable.

NOTE: If the artifact storage service is disabled, this feature is unusable.

- When a flow is exported, an `artifact.data` file is included as part of the export. This file must be imported with the flow definition, or the TBE step in the imported flow is broken. For more information, see *Export Flow*.

Limitations

- TBE works best for inputs that are text-based data types (e.g. String, State, URL, etc.).
 - Non-text inputs are treated as String type and may result in unexpected outputs (Integer, Decimal, etc.).
 - You cannot use multi-value inputs, such as Arrays or Objects, or use the feature to create them.

Tip: If you have Array or Object input columns, convert them to String type before using TBE.

- TBE bases its transformations on the currently displayed sample.
 - Even if you accurately map all values in your sample, some other values in the full dataset may not be mapped by the transformation.
 - You may need to take additional samples of other parts of the entire dataset to generate a more accurate transformation.
- Arithmetic operations or other numeric functions are not supported.
- You cannot create multiple columns from a single TBE step.

Enable

This feature can be enabled and disabled through the Settings page in the Admin console.

Locate the following setting:

Create examples

Set this value to `Enabled`.

Column by Example

In column-by-example transformations, you create a new column from an existing one by mapping input to output values.

General workflow:

1. Select the column to use as input data.
2. Change the column to String data type, if needed.
3. From the column menu, select **Create column from examples**. See *Transformation by Example Page*.
4. Transform by example:
 - a. Locate a row containing an example value to transform.
 - b. In the corresponding row in the Preview column, you can enter in the new value to which the input is mapped.
 - c. The transformation in development is updated to accurately capture the mapping you just performed. Additional rows in the output column may be accurately mapped, as well.
5. Repeat the above steps until all values in the output column appear to be accurately mapped.
6. When satisfied, add the transformation to your recipe.
7. Change the data type of the target and the source columns, if needed.
8. Remove the source column, if needed.

For more information, see *Create Column by Example*.

CBE for Datetime

Column-by-example also works on Datetime columns. When you use a Datetime column as your input, you specify the output values in the date/time format that you wish to use. That input value and all similarly formatted inputs should be converted to the output format. You can then specify additional example outputs for input values in a different format to standardize all of the values in the output column.

NOTE: For Datetime formatting to work properly, the input column must be specified as Datetime data type.

Alternatives

For string-based inputs, the following options in `Wrangle` may assist in performing the same functions that you are trying to do in TBE:

Wrangle	Description
<i>Extract Transform</i>	You can use the extract transform to retrieve sub-strings from a column and insert into a new column.
<i>String Functions</i>	Wrangle supports a variety of string manipulation functions, which can be used to gather data from a string.

Overview of Data Quality

Contents:

- *Data Quality Characteristics*
 - *Schema Validation*
 - *Assign targets*
 - *Identify Anomalies*
 - *Data quality bar*
 - *Column histogram*
 - *Column details*
 - *Standardization*
 - *Data Quality Functions*
 - *Type functions*
 - *Count functions*
 - *Aggregation functions*
 - *Statistical functions - single column*
 - *Statistical functions - multi-column*
 - *Data Quality in Job Details*
 - *Visual profiling*
-

Trifacta® provides multiple mechanisms to transform and standardize data to meet usage needs, including profile visualizations and type-based quality bars to identify potential anomalies and quality problems. **Data quality** checks can be applied during data import, transformation, or export in the form of visual profiling.

Broadly speaking, data quality identifies the degree to which data is usable and responsive to your use case. When you assess data quality, you are designing tests to assess its suitability for generic usage and for your specific uses.

Data Quality Characteristics

Data quality covers the following characteristics:

- **Completeness:** values are present where they are needed and expected
- **Accuracy:** data is substantively free of errors
- **Consistency:** a dataset can be matched across different data sources of the enterprise
- **Timeliness:** data values are up-to-date
- **Uniqueness:** aggregate data are free from any duplication via filters or other transformations of source data
- **Validity:** data are structured based on an adequate and rigorous classification system
- **Availability / Accessibility:** data are made available to the relevant stakeholders
- **Traceability:** the history, processing and location of the data under consideration can be easily traced

Schema Validation

Type inference

When data is imported, the Trifacta attempts to infer the data types in the source and to type columns in the dataset accordingly. Type inference uses the first 20-25 rows of the initial sample to assess the appropriate data type to apply to the column. For more information, see *Type Conversions*.

Some imported data, such as relational tables, may include schema information to identify the data type of each column. In some cases you can disable type inferencing on imported data:

- **Global:** Trifacta administrators can disable type inferencing for all imported schematized sources. In this manner, the Trifacta platform uses the schema of the source to define the initial types assigned to the columns of the dataset.
- **Connections:** As part of the definition of a connection, you can optionally choose to disable type inference. For more information, see *Create Connection Window*.
- **Per-dataset:** When you import a dataset, you can modify the import settings for the selected source to disable type inference. See *Import Data Page*.

Assign targets

To assist in your transformation efforts, you can assign a target schema for each recipe. This target schema is super-imposed on the columns of your data. Using visual tools to review differences and select changes, you can rapidly convert the structure of your dataset in development to meet the expected target schema. For more information, see *Overview of RapidTarget*.

Identify Anomalies

In the Transformer page, you can use the available visual tools to review the data quality characteristics of the columns in your data. These data visualizations and type-based quality bars can assist in identifying potential anomalies and quality problems.

Data quality bar

At the top of each column, you can see a data quality bar, which uses the following color coding to validate the column values against the selected column type.

Color bar	Description
green	Values that are valid for the current data type of the column
red	Values that are mismatched for the column data type
gray	Missing or null values

Tip: Click any of the color bars to receive suggestions for transformations to add to your recipe.

Tip: You can change a column's data type in the column header. See *Column Menus*.

For more information, see *Data Quality Bars*.

Column histogram

In the column header, you can review the count and distribution of values in the column. A column's histogram can be useful for identifying anomalies or for selecting specific sets of values in the column for further exploration.

Tip: Click and drag over any set of values to receive suggestions for transformations to add to your recipe.

See *Column Histograms*.

Column details

Through the Column Details panel, you can explore the quality and distribution of the values in the column. The contents of the panel vary depending on the data type. For example, if the column is typed for Datetime values, then the Column Details panel includes information on the distribution of values across the days of the week and days of the month.

For all data types, you can review useful statistics on statistical quartiles, the uniqueness of values, mismatches, and outliers.

Tip: The Column Details panel is very useful for acquiring statistical information on column values in a visual format. Click any data quality bar to be prompted for suggestions of transformation steps. See *Overview of Predictive Transformation*.

For more information, see *Column Details Panel*.

Standardization

You can use the Standardization tool to standardized clustered sets of column values to values that are common and consistent throughout your enterprise's data. For more information, see *Overview of Standardization*.

Data Quality Functions

The following functions are available for assessing data quality.

Type functions

Item	Description
NULL Function	The NULL function generates null values.
IFNULL Function	The IFNULL function writes out a specified value if the source value is a null. Otherwise, it writes the source value. Input can be a literal, a column reference, or a function.
IFMISSING Function	The IFMISSING function writes out a specified value if the source value is a null or missing value. Otherwise, it writes the source value. Input can be a literal, a column reference, or a function.
IFMISMATCHED Function	The IFMISMATCHED function writes out a specified value if the input expression does not match the specified data type or typing array. Otherwise, it writes the source value. Input can be a literal, a column reference, or a function.
IFVALID Function	The IFVALID function writes out a specified value if the input expression matches the specified data type. Otherwise, it writes the source value. Input can be a literal, a column reference, or a function.
ISNULL Function	The ISNULL function tests whether a column of values contains null values. For input column references, this function returns true or false.
ISMISSING Function	The ISMISSING function tests whether a column of values is missing or null. For input column references, this function returns true or false.
ISMISMATCHED Function	Tests whether a set of values is not valid for a specified data type.
VALID Function	Tests whether a set of values is valid for a specified data type and is not a null value.
PARSEINT Function	Evaluates a String input against the Integer datatype. If the input matches, the function outputs an Integer value. Input can be a literal, a column of values, or a function returning String values.
PARSEBOOLEAN Function	Evaluates a String input against the Boolean datatype. If the input matches, the function outputs a Boolean value. Input can be a literal, a column of values, or a function returning String values.

PARSEFLOAT Function	Evaluates a String input against the Decimal datatype. If the input matches, the function outputs a Decimal value. Input can be a literal, a column of values, or a function returning String values.
PARSEARRAY Function	Evaluates a String input against the Array datatype. If the input matches, the function outputs an Array value. Input can be a literal, a column of values, or a function returning String values.
PARSEOBJECT Function	Evaluates a String input against the Object datatype. If the input matches, the function outputs an Object value. Input can be a literal, a column of values, or a function returning String values.
PARSESTRING Function	Evaluates an input against the String datatype. If the input matches, the function outputs a String value. Input can be a literal, a column of values, or a function returning values. Values can be of any data type.

Count functions

The following functions measure counts of values within a column, optionally counted by group.

Item	Description
<i>COUNT Function</i>	Generates the count of rows in the dataset. Generated value is of Integer type.
<i>COUNTA Function</i>	Generates the count of non-null rows in a specified column, optionally counted by group. Generated value is of Integer type.
<i>COUNTDISTINCT Function</i>	Generates the count of distinct values in a specified column, optionally counted by group. Generated value is of Integer type.
<i>UNIQUE Function</i>	Extracts the set of unique values from a column into an array stored in a new column. This function is typically part of an aggregation.

Aggregation functions

Item	Description
<i>AVERAGE Function</i>	Computes the average (mean) from all row values in a column or group. Input column can be of Integer or Decimal. See also: <ul style="list-style-type: none"> <i>AVERAGEIF Function</i>
<i>SUM Function</i>	Computes the sum of all values found in all row values in a column. Input column can be of Integer or Decimal.
<i>MIN Function</i>	Computes the minimum value found in all row values in a column. Input column can be of Integer, Decimal or Datetime.
<i>MAX Function</i>	Computes the maximum value found in all row values in a column. Inputs can be Integer, Decimal, or Datetime.
<i>MODE Function</i>	Computes the mode (most frequent value) from all row values in a column, according to their grouping. Input column can be of Integer, Decimal, or Datetime type.
<i>MINDATE Function</i>	Computes the minimum value found in all row values in a Datetime column.
<i>MAXDATE Function</i>	Computes the maximum value found in all row values in a Datetime column.
<i>MODEDATE Function</i>	Computes the most frequent (mode) value found in all row values in a Datetime column.

Statistical functions - single column

Variations in these functions:

- Some of these functions have variations that use the sample population method of computation.
- IF conditional functions can be used to compute statistical computations based on a condition.

General statistics

Item	Description
------	-------------

<i>VAR Function</i>	Computes the variance among all values in a column. Input column can be of Integer or Decimal. If no numeric values are detected in the input column, the function returns 0.
<i>STDEV Function</i>	Computes the standard deviation across all column values of Integer or Decimal type.
<i>MEDIAN Function</i>	Computes the median from all row values in a column or group. Input column can be of Integer or Decimal.
<i>QUARTILE Function</i>	Computes a specified quartile across all row values in a column or group. Input column can be of Integer or Decimal.
<i>PERCENTILE Function</i>	Computes a specified percentile across all row values in a column or group. Input column can be of Integer or Decimal.

Item	Description
<i>APPROXIMATEMEDIAN Function</i>	Computes the approximate median from all row values in a column or group. Input column can be of Integer or Decimal.
<i>APPROXIMATEQUARTILE Function</i>	Computes an approximation for a specified quartile across all row values in a column or group. Input column can be of Integer or Decimal.
<i>APPROXIMATEPERCENTILE Function</i>	Computes an approximation for a specified percentile across all row values in a column or group. Input column can be of Integer or Decimal.

Statistical functions - multi-column

Item	Description
<i>COVAR Function</i>	Computes the covariance between two columns using the population method. Source values can be of Integer or Decimal type.
<i>CORREL Function</i>	Computes the correlation coefficient between two columns. Source values can be of Integer or Decimal type.

Data Quality in Job Details

When you run a job and generate results, you can review the the quality of the data of the generated output.

Visual profiling

In parallel with executing the job, you can generate a visual profile of the generated results. This visual profile provides graphical representations of the valid and mismatched values against each column's data type, as well as indications about missing values in the output.

Tip: Visual profiles can be downloaded in PDF or JSON format for offline analysis.

Visual profiling is selected as part of the job definition process. See *Run Job Page*.

For more information, see *Overview of Visual Profiling*.

For more information, see *Job Details Page*.

Overview of Sharing

Contents:

- *Enable*
- *Sharing Model*
 - *Owners and collaborators*
 - *Role by object type*
 - *Fine-grained sharing privileges for individual shared objects*
- *Shareable Objects*
 - *Sharing Flows*
 - *Share Connections*
 - *Share Plans*

In a collaborative environment, it can be helpful to be able to have multiple users work on the same assets or to create copies of good quality work to serve as templates for others. Trifacta® enables users to collaborate on the same flow objects or to create copies for others to use for independent work.

This section provides an overview of sharing principles, limitations, and approaches.

Enable

Sharing can be enabled and disabled through Workspace settings by a workspace administrator. To enable, set the following to `Enabled`.

Sharing

For more information, see *Workspace Settings Page*.

Sharing Model

NOTE: You cannot share with users outside of your current project or workspace, including any account that you may have in a different project or workspace.

NOTE: You may not be permitted to share objects with users who have not yet logged into the product.

Owners and collaborators

The following are the basic types of users of a shared object:

User Type	Description
Owner	<div>Typically, the owner is the original creator of the shared object. This user has maximum permissions on the object.</div> <div>NOTE: There can be only one owner on an object. Only the owner or a workspace admin can delete a shared object.</div>
Workspace admin	All workspace admins have owner rights on all objects in the workspace.
Collaborator	

Any user who has been shared an object is a **collaborator**. A collaborator can have the one of the following permissions on the object:

- Editor
- Viewer

See below.

Role by object type

Individual users can be assigned one or more roles. A **role** is a set of privileges (permissions).

For each type of shareable object, an administrator can define within a role the privileges that users have on the object type. Below, you can review the basic privilege levels and the implications on sharing:

Privilege	Description	If object shared, default privileges on the object
Author	Assigned user can create and delete new objects of this type.	Editor
Editor	Assigned user can modify objects of this type with limitations. See below.	Editor
Viewer	Assigned user has read-only access to this type of object.	Viewer

For more information, see *Roles Page*.

Fine-grained sharing privileges for individual shared objects

When an object is shared, the user who is sharing the object can specify the privilege level for the target user on the shared object, which provides finer-grained access controls on individual objects:

- The high-level privileges define the maximum set of privileges that you can share on an object with the target user.
- Project- or workspace-level privileges on object types can be overridden for individual objects.
- For example, a user with Viewer privileges on flows at the project or workspace level cannot be given Editor privileges on any individual flow.

Limitations:

- Fine-grained sharing privileges apply to flows, plans, and connections only.
- Users who have received changes in privileges on individual objects should log out and log in again to see those changes.

Shareable Objects

The following types of objects can be shared with other users:

- Flows
- Connections
- Plans

Sharing Flows

In the collaborative approach, two or more users can work on the same flow. When a flow is shared, all flow objects are shared, including:

- Imported datasets

NOTE: A dataset that is created with parameters cannot be modified by a collaborator. It can only be modified by the owner.

- Recipes
- Output objects
 - If available, any output SQL scripts are also shared.
- Job results
- Webhook tasks

NOTE: Sharing of data is managed at the flow level. You cannot share individual recipes or datasets from within a flow.

NOTE: You cannot share a flow with yourself.

All collaborators have access to the above objects, as long as they have access to the underlying sources. See below.

Use cases:

- Distribute the work on a flow with multiple recipes among team members for faster throughput.
- Pass recipes to others for commenting, editing, and general review.
- When stuck, share the flow with the team expert to provide guidance.

Privileges

Underlying datasets: Sharing a flow does not change the permissions to the underlying data. If a user with whom a flow has been shared does not have access to the data on the datastore, the user cannot work with the flow's datasets.

- Datasets that are accessed through private connections cannot be shared, unless the connection is also shared.
- Stricter permissions sets on the datastore can adversely affect users' ability to access shared flows.

Sharing samples: A flow's samples are not necessarily available to all users who have been shared the flow. In some cases, if a user who has been shared a flow does not have access to a recipe's sample, the user may have to collect a separate sample to view data or edit the recipe associated with the sample. To enable universal access to shared samples, you can use either of the following permissions schemes:

1. The default output directories for any user can be accessed by any other user. This configuration must be managed in the base storage layer.
2. When the sample is executed, an individual user must set his or her default output directory to a location that shared users of the flow can access.

When flows are shared with you, you can access them through the Shared with Me tab in the Flows page. See *Flows Page*.

Editor privileges:

- Datasets
 - Use the imported datasets and references as sources in other flows accessible to the collaborator.
 - Add new imported datasets.
 - Remove existing imported datasets.
 - Change the source of datasets.
 - Edit dataset names and descriptions.
- Recipes
 - Add new recipes.
 - Edit the existing recipes, including multi-dataset operations such as union or join.
 - Delete recipes.
 - Copy recipes within the shared flow.

- Move recipes to the shared flow.
- Move recipes out of the shared flow.
- Run jobs.
- Schedules
 - Create new schedules.
 - Edit schedules.

Viewer privileges:

- User can access the flow and run jobs.
- User cannot modify the flow.
- Schedules
 - Create new schedules.
 - Edit schedules.

Collaborator (Editor and Viewer) limitations:

Collaborators do not have the following privileges on a flow shared with them:

- Flow
 - Delete the flow
 - Edit the name and description of the flow
 - Remove the flow owner's access to the flow
- Datasets
 - Delete imported datasets
 - Modify imported datasets

NOTE: Collaborators cannot modify datasets created with custom SQL.

For more information on the privileges for Viewer and Editor roles, see *Privileges and Roles Reference*.

Editing recipes

Owners and Editors have the same privileges to edit recipes in the shared flow. In the Edit History, edits appear under the usernames of the individual contributors.

NOTE : Multiple editors cannot make changes to the same recipe at the same time.

NOTE: When a column is hidden from a dataset, it is hidden for all users.

Tip: You can review the history of changes to a recipe through the Edit History for a recipe. See *Recipe Panel*.

Removing access

You can remove sharing access to a flow. When a flow is no longer shared with a user, that user:

- Cannot see the flow or its objects
- Cannot access them, if the user knows the location of the objects

NOTE: If a dataset from a shared flow is referenced in another flow, when sharing access is removed from the flow, the referenced dataset is still available in the other flow.

NOTE: If a flow is unshared with you, you cannot see or access the datasources for any jobs that you have already run on the flow, including any PDF profiles that you generated. You can still access the job results. This is a known issue.

Share Connections

When initially created, a connection is **private**. It is accessible only to the user who created it.

Through the Connections page, you can share your connections with other users:

- **Share connection with individual users:** You can share your connection with specified users.
 - You can also share connections that have been shared with you.
- **Make connection public:** Public connections are available for use by all users.

NOTE: Only an admin can make connections public. After a connection has been made public, it cannot be made private again. You must delete and recreate the connection.

When connections are shared with you, you can access them through the Shared with Me tab in the Connections page. See *Connections Page*.

Sharing credentials:

When shared, private connections can be shared with or without credentials. If credentials are not shared, new users of the shared connection must supply their own credentials. Those credentials must be permitted access if access to any datasets previously imported through the connection is required.

NOTE: A workspace admin has owner-level access to all connections. However, a workspace admin cannot access or use a connection's credentials if those credentials were not shared by the owner of the connection. For more information, see *Workspace Admin Permissions*.

NOTE: Password values for credentials are always masked in the user interface.

NOTE: For SSO connections, credentials are never shared. Instead, the Kerberos principal of the user with whom the connection is shared is used to connect. That principal must have the appropriate permissions to access the data.

For more information, see *Connections Page*.

Sharing connections through flows:

When a flow is shared, any connections associated with it are automatically shared to the specified users. If the connection is configured to do so, credentials are included, so that the new users can immediately begin using the flow.

For more information, see *Flow View Page*.

For more information on the privileges for Viewer and Editor roles, see *Privileges and Roles Reference*.

Share Plans

Plans that you create can be shared with other users. In the Plans page, select **Share** from a plan's context menu.

Depending on whether you created the plan, you may have the following set of privileges:

You are	Privileges
Owner	The owner created the plan and can schedule the plan and has all editor privileges.
Collaborator	A collaborator has been shared the plan as a Viewer or Editor. Privileges to the plan that are limited in the following ways: <ul style="list-style-type: none">• Collaborators cannot delete plans that have been shared with them.• Collaborator access to the plan may be further filtered based on assignments at the project or workspace level. See below.

When a plan is shared with you, you are a collaborator on the plan. A collaborator has the following capabilities based on the plan privileges assigned to your role:

Plan Privilege	Description
Author	<ul style="list-style-type: none">• Create plans.• Delete plans that you create.• All Editor privileges.
Editor	<ul style="list-style-type: none">• Edit parameters in entitled plans• Manage email notifications on entitled plans• Update entitled plans names and descriptions• Share entitled plans• All Viewer privileges.
Viewer	<ul style="list-style-type: none">• View and run entitled plans• View runs and jobs from entitled plans• Export entitled plans

For more information on the privileges for Viewer and Editor privileges, see *Privileges and Roles Reference*.

For more information, see *Share a Plan*.

Overview of Job Monitoring

Contents:

- *Monitoring Phases*
 - *Connect*
 - *Request*
 - *Transfer*
 - *Prepare*
 - *Process*
 - *Enable*
 - *Configure*
 - *Enable phases in Data sources tab*
 - *Enable phases in Outputs tab*
 - *Monitoring Jobs in the Application*
 - *Flow View*
 - *Import*
 - *Job Details Page*
-

Trifacta® supports detailed monitoring of a job throughout each phase of its execution.

Limitations:

- Applies only to ingest and publishing jobs
- Applies only to JDBC datasets

Monitoring Phases

These phases apply to ingest and publishing jobs. Information on them is surfaced in the application.

Connect

In the Connect phases, Trifacta uses the specified connection for the flow to connect to the source of the job.

NOTE: Errors in this phase typically involve issues in the connection definition or in the network configuration or availability.

Request

After the platform has been able to connect to the datastore, the Request phase entails the submission of the request to the datastore for the assets. For example, for JDBC-based datasets, this phase covers the SQL query of the database through the response that the query was successfully executed.

NOTE: Errors in this phase typically reflect errors in the SQL query, which can include renaming or moving of assets in the datastore.

NOTE: If assets are retrieved via custom SQL query, you may need to review the query and validate it through the Trifacta application. For more information, see *Create Dataset with SQL*.

Transfer

This phase covers the transfer of assets from the datastore to the platform.

NOTE: Errors in this phase typically indicate issues with permissions.

Prepare

NOTE: This phase applies to publishing jobs only.

Depending on the destination, the Prepare phase includes the creation of temporary tables, generation of manifest files, and the fetching of extra connections for parallel data transfer.

Process

After the data has been transferred to the platform, this phase covers the processing of cleanup after data transfer, including the dropping of temporary tables or copying data within the instance.

Enable

The base feature is enabled by default.

```
"feature.enableJobMonitoring": true,
```

Configure

Optionally, you can enable the following capabilities in the Trifacta application. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.

Enable phases in Data sources tab

To display separate columns in the Data sources tab of the Job Details page for each phase on an ingest job, set the following parameter to `true`:

```
"jobMonitoring.enablePhasesInDatasourcesTable": true,
```

Enable phases in Outputs tab

To display separate columns in the Outputs tab of the Job Details page for each phase for a publish job, set the following parameter to `true`:

```
"jobMonitoring.enablePhasesInOutputsTable": true,
```

Save your changes and restart the platform.

Monitoring Jobs in the Application

When the base feature is enabled, you can monitor jobs in the following locations.

Flow View

- Track phases in the Jobs panel in Flow View. Hover the mouse over the link to the job.
- See *Flow View Page*.

Import

NOTE: This feature may require enablement in your deployment. See *Configure JDBC Ingestion*.

Import Data:

For long-loading datasets, you can track the progress of the import through the Import Data page as you specify the import. See *Import Data Page*.

Library:

After specifying the import, if the data is continuing to be ingested, you can track progress through the Library page. See *Library Page*.

Dataset Details Page:

In the Dataset Details page, you can monitor the ingest progress. Hover over the Status link.

SQL Dataset 1

Last updated: 06/15/2020

Connection: postgres

Status: Completed < 10 sec

Custom SQL [Edit](#)

Used in 1 Flow

Name	Owner	Objects	Last Updated
Untitled Flow	Administrator	1 Dataset, 1 Recipe	Last Thursday at 9:17 PM

Connect

Ran for <1 sec

Request

Ran for <1 sec

Transfer

Ran for <1 sec

0.0017 MB

0.0017 MB/sec

Created: 06/15/2020

Column Data Type Inference: Disabled

Job ended: 06/15/2020

Wrangle in new Flow

Preview

Edit custom SQL

...

Figure: Dataset Details Page - Job Monitoring

Job Details Page

- Track phases of progress by hovering over the job in progress in the Job Details page.
- Review new and better detail in the Job Details page. Click **View Details** for the job listing.
- For more information, see *Job Details Page*.

Datasources tab - Phased ingest monitoring

If job monitoring phases have been enabled for the Datasources tab, the tab looks like the following:

<div>Monitoring > Farmers_Market - 2</div> <div>Job 7</div> <div>Started Today at 12:36 PM</div> <div>Cancel job</div> <div>...</div>						
Overview Output Destinations Profile Dependencies <u>Data sources</u>						
Name	Queued	Connect	Request	Transfer	Process	Status
Farmers_Market	3 sec	✓ <1 sec	✓ <1 sec	✓ 9 sec	-	✓ Completed • 13 sec


Figure: Job monitoring in the Datasources tab

View details:

If an ingest job succeeds or fails, you can click **View details** in the status column for additional information on each phase of the ingest job:

Details

×

 Farmers_Market

Details

SQL

Type

oracle

Created

June 24th 2019, 11:38 am

Ingestion

✓

Completed • 13 sec

Ingestion details

Queued

In queue for 3 sec

Connect

✓

Ran for <1 sec

Request

✓

Ran for <1 sec

Transfer

✓

Ran for 9 sec
76.8760 MB
8.5418 MB/sec

Figure: View details on monitoring ingest jobs

Output destinations tab - Phased publishing monitoring

If job monitoring phases have been enabled for the Output Destinations tab, the tab looks like the following:

Monitoring > Farmers_Market - 2

Job 7

Finished Today at 12:37 PM

Publish results
...

Overview
Output Destinations
Profile
Dependencies
Data sources

You can download the generated results locally or [publish](#) to another storage location.







Name	Queued	Connect	Prepare	Transfer	Process	Status
 test_20190624_193621	2 sec	 <1 sec	 <1 sec	 <1 sec	 <1 sec	 Completed • 15 sec


Figure: Job monitoring in the Datasources tab

View details:

If a publishing job succeeds or fails, you can click **View details** in the status column for additional information on each phase of the publishing job:

Details

×

 test_20190624_193621

Data preview

#	FM_ID	RBC	Market_Name
1005627			Montgomery Farmers' Market
1002686			Hines Veterans Hospital Farmers Market
1004509			Montgomery Village Farmers Market
1001779			Farmstand at CHSAS
1008852			Monticello Farmers' Market
1007417			Farmville Farmers Market
1002538			Monticello Old Jail Market

Type

ORACLE

Location

MKOHLLI_TEST/test_20190624_193621


Size

3 columns · 3 types

Start Time

June 24th 2019, 12:36 pm

Publish


 Completed · 51 sec

Publish details

Queued

In queue for 38 sec

Connect

 Ran for <1 sec

Prepare


 Ran for <1 sec
5 connections

Figure: View details on monitoring ingest jobs

Overview of Automator

Contents:

- *Limitations*
 - *Data Management*
 - *Flows for scheduling*
 - *Schedule a Job*
 - *Job Execution*
 - *Tracking*
 - *Configure*
-

As needed, you can use the **Automator** to schedule the execution of recipes in your flows on a recurring basis. For example, if the source file of your flow is updated outside of the application on a weekly basis, you can define a schedule to execute the recipe associated with the related imported dataset after the data has been refreshed. When the scheduled job successfully executes, you can collect the wrangled output in the specified output location, where it is available in the published form that you have specified.

- This feature was formerly known as, "scheduling."

To schedule a job, you must create the following configuration objects:

1. **Define a schedule** - For each flow you can define a schedule. A **schedule** specifies one or more recurring times (**triggers**) when scheduled jobs for the flow are executed. For example, in a single schedule, you can specify daily trigger times for incremental updates and monthly execution times for rollups.

Tip: The scheduler supports a modified form of cron job syntax. For more information, see *cron Schedule Syntax Reference*.

2. **Define one or more scheduled destinations** - When you specify a **scheduled destination** for a recipe, the recipe is executed whenever one of the schedule's execution times occurs. Scheduled destinations are specified like regular destinations in flow view.

NOTE: When a schedule for a flow is triggered, all of recipes to generate the scheduled destinations are executed. Manual destinations are not generated. You cannot create schedules for individual outputs.

For more information on the scheduling objects, see *Object Overview*.

Limitations

- One schedule cannot be applied to multiple flows.
- You cannot create separate schedules for individual recipes within a flow. A schedule defined at the flow level applies to all recipes in the flow.
- Only a flow owner can create or modify a flow's schedule.

Data Management

NOTE: Since scheduled destinations are re-populated with each scheduled execution, you must determine how you wish to manage the data that is published to each location. Data management should be done outside of Trifacta®.

- **Import:** Before each scheduled execution, you should refresh the source of the imported dataset with new data outside of Trifacta.
- **Execution:** Please verify that the publishing settings for your scheduled destination are consistent with how you are using the results. For example, if the scheduled destination creates a new file with the same name for each execution (replace), you must move the generated file out of the output location before the next scheduled execution.
- **Output:** You must collect the generated results. While you can export the job's results through the Jobs page, you may find it easier to use an external scheduler to gather the results and forward to the downstream consumer of them.

Flows for scheduling

Tip: When a schedule is executed, all outputs in a flow are generated, even if they are unused. For better performance on larger flows, you can create a separate flow that contains only the references back to the objects in the source flow that you wish to have scheduled. As an additional benefit, this separation keeps development and scheduled execution in separate flows.

Schedule a Job

Schedules and scheduled destinations are defined through Flow View.

Tip: You can create schedules for datasets with parameters and apply overrides through Flow View at runtime. See *Flow View Page*.

For more information, see *Schedule a Job*.

Job Execution

Tracking

You can monitor a scheduled job like any other job in the application. See *Jobs Page*.

Configure

See *Configure Automator*.

Overview of Parameterization

Contents:

- *Environment Parameters*
 - *Limitations*
 - *Example - parameterized bucket names*
 - *Export and Import*
 - *Datasets with Parameters*
 - *Example*
 - *Parameter Types*
 - *Guidelines for Sources*
 - *Mismatched Schemas*
 - *Limitations*
 - *Creating Dataset with Parameters*
 - *Managing Datasets with Parameters*
 - *Flow Parameters*
 - *Limitations*
 - *Example*
 - *Upstream flow parameters*
 - *Creating flow parameters*
 - *Managing flow parameters*
 - *Flow parameters in plans*
 - *Output Parameters*
 - *Parameter Types*
 - *Example*
 - *Creating output parameters*
 - *Using output parameters*
 - *Bucket Name Parameters*
 - *Parameter Overrides*
 - *Order of Parameter Evaluation*
 - *Run Jobs with Parameters*
 - *Runtime Parameter Overrides*
 - *Scheduling jobs on datasets with parameters*
 - *Parameters in Job Details*
 - *Operationalization with Parameters*
 - *APIs*
 - *Configuration*
 - *Disable*
-

In Trifacta®, **parameterization** enables you to apply dynamic values to the data that you import and that you generate as part of job execution. Parameter types:

- **Environment Parameters:** A workspace administrator or project owner can specify parameters that are available across the environment, including default values for them.
- **Dataset Parameters:** You can parameterize the paths to inputs for your imported datasets, creating datasets with parameters. For file-based imported datasets, you can parameterize the bucket where the source is stored.
- **Flow Parameters:** You can create parameters at the flow level, which can be referenced in any recipe in the flow.
- **Output Parameters:** When you run a job, you can create parameters for the output paths for file- or table-based outputs.

These parameters can be defined by timestamp, patterns, wildcards, or variable values that you specify at runtime.

Environment Parameters

Project owners or workspace administrators can define parameters that apply across the project or workspace environment. These parameters can be referenced by any user in the environment, but only a user with admin access can define, modify, or delete these parameters.

Tip: Environment parameters are a useful means of ensuring that all users of the project or workspace share common reference values to buckets, output locations, and more. Environment parameter definitions can be exported and then imported into other projects or workspaces to ensure commonality across the enterprise. The values assigned to environment parameters can be modified after they have been imported into a new project or workspace.

NOTE: You must have admin access to the project or workspace to define environment parameters.

- Names of environment parameters must begin with `env.`

Limitations

- You cannot use environment parameters in recipes.
- You cannot use environment parameters in plans.
- Environment parameter names are unique within the environment.
- You cannot use environment parameters in Deployment Manager. For more information, see *Overview of Deployment Manager*.

Example - parameterized bucket names

In this example, you have three Trifacta workspaces, each of which has a different set of resources, although the only difference between them is the name of the S3 bucket in which they are stored:

Environment Name	S3 Bucket Name
Dev	myco-s3-dev
Test	myco-s3-test
Prod	myco-s3-prod

In your Dev workspace, you can create an environment parameter called the following:

```
env.bucket-source
```

The default value for this parameter is set to:

```
myco-s3-dev
```

When creating imported datasets in this workspace, you insert the environment parameter for the source bucket for each one.

For your Test and Prod environments:

- Export your environment parameters from Dev.

2. Import them into Test and Prod. During import, the importing user can map the imported parameters to existing parameters in the environment.
3. In the imported environments, an administrator can manage the imported parameters and values as needed.

When you later export your flows from Dev and move them to Test and Prod, the imported flows automatically connect to the correct bucket for the target environment, since the bucket name is referenced by an environment parameter.

Export and Import

You can export environment parameters from one environment and import them to another. For example, you may be building your flows in a Dev workspace before they are exported and imported into a Prod workspace. If your flows make use of environment parameters from the Dev space, you may want to export the parameters and their values from the Dev workspace for migration to the Prod workspace.

NOTE: As part of the import process, you must reconcile name conflicts between imported environment parameters and the parameters that already exist in the workspace.

For more information, see *Manage Environment Parameters*.

Datasets with Parameters

In some cases, you may need to be able to execute a recipe across multiple instances of identical datasets. For example, if your source dataset is refreshed each week under a parallel directory with a different timestamp, you can create a variable to replace the parts of the file path that change with each refresh. This variable can be modified as needed at job runtime.

Example

Suppose you have imported data from a file system source, which has the following source path to weekly transactions:

```
<file_system>:///source/transactions/2018/01/29/transactions.csv
```

In the above, you can infer a date pattern in the form of 2018/01/29, which suggests that there may be a pattern of paths to transaction files. Based on the pattern, it'd be useful to be able to do the following:

- Import data from parallel paths for other weeks' data.
- Sample across all of the available datasets.
- Execute jobs based on runtime variables that you set for other transaction sets fitting the pattern.
- Pass in parameterized values through API to operationalize the execution of jobs across weeks of transaction data.

In this case, you would want to parameterize the date values in the path, such that the dynamic path would look like the following:

```
<file_system>:///source/transactions/YYYY/MM/DD/transactions.csv
```

The above example implements a Datetime parameter on the path values, creating a **dataset with parameters**.

Parameter Types

You can use the following types of parameters to create datasets with parameters:

- **Datetime parameters:** Apply parameters to date and time values appearing in source paths.
 - When specifying a Datetime parameter, you must also specify a **range**, which limits the range of the Datetime values.
- **Variables:** Define variable names and default values for a dataset with parameters.
 - Variable parameters can be applied to elements of the source path or to the bucket name, if applicable.
 - Modify these values at runtime to parameterize execution.
- **Pattern parameters:**
 - Wildcards: Apply wildcards to replace path values.
 - Regular Expressions: You can apply regular expressions to specify your dataset matches. Please see the limitations section below for more information.
 - Patterns : The platform supports a simplified means of expressing patterns.
 - For more information on Patterns , see *Text Matching*.

For more information, see *Create Dataset with Parameters*.

Guidelines for Sources

The source files or tables for a dataset with parameters should have consistent structures. Since the sources are parsed with the same recipe or recipes, variations in schema could cause breakages in the recipe or initial parsing steps, which are applied based on the schema of the first matching source.

NOTE: All datasets imported through a single parameter are expected to have exactly matching schemas. For more information on variations, see *Mismatched Schemas* below.

Tip: If there have been changes to the schema of the sources of your dataset with parameters, you can edit the dataset and update the parameters. See *Library Page*.

Parameters in paths for imported datasets are rendered as regular expressions. Depending on the number of parameters and the comparative depth of them in a parameterized dataset, the process of performing all pattern checks can grow large, impacting import performance.

Tip: When specifying an imported dataset with parameters, you should attempt to be as specific as possible in your parameter definitions.

NOTE: When importing one or more Excel files as a parameterized dataset, you select worksheets to include from the first file. If there are worksheets in other Excel files that match the names of the worksheets that you selected, those worksheets are also imported. All worksheets are unioned together into a single imported dataset with parameters. Pattern-based parameters are not supported for import of Excel worksheets.

Mismatched Schemas

Trifacta expects that all datasets imported using a single parameter have schemas that match exactly. The schema for the entire dataset is taken from the first dataset that matches for import.

If schemas do not match:

- When the first dataset contains extra columns at the end, the subsequent datasets that match should import without issues.
- If the subsequent datasets contain extra columns at the end, the datasets may import. Depending on the situation, there may be issues.
- If the subsequent datasets have additional or missing columns in the middle of the dataset, results of the import are unpredictable.

- If there are extra columns in the middle of the dataset, you may see extra data in the final column, in which the spill-over data has not been split.
- Ideally, you should fix these issues in the source of the data. But if you cannot, you can try the following:

Tips:

- After import of a dataset with parameters, perform a full scan random sample. When the new sample is selected:
 - Check the last column of your imported to see if you have multiple columns of data. See if you can perform split the columns yourself.
 - Scan the column histograms to see if there are columns where the number of mismatches or anomalous or outlier values has suddenly increased. This could be a sign of mismatches in the schemas.
- Edit the dataset with parameters. Review the parameter definition. Click **Update** to re-infer the data types of the schemas. This step may address some issues.
- You can use the union tool to import the oldest and most recent sources in your dataset with parameters. If you see variations in the schema, you can look to modify the sources to match.
 - If your sources have variation in structure, you should remove the structure from the imported dataset and create your own initial parsing steps to account for the variations. See *Initial Parsing Steps*.

Limitations

- You cannot create datasets with parameters from uploaded data.
- You cannot create dataset with parameters from multiple file types.
 - File extensions can be parameterized. Mixing of file types (e.g. TXT and CSV) only works if they are processed in an identical manner, which is rare.
 - You cannot create parameters across text and binary file types.
- For datasources that require conversion, such as Excel, PDF, or JSON files, you can create a dataset with parameters from a maximum of 500 converted files.
- Parameter and variable names can be up to 255 characters in length.
- For regular expressions, the following reference types are not supported due to the length of time to evaluate:
 - Backreferences. The following example matches on `axa`, `bxb`, and `cxc` yet generates an error:

```
(([a-c])x\1
```

- Lookahead assertions: The following example matches on `a`, but only when it is part of an `ab` pattern. It generates an error:

```
a(?:=b)
```

- For some source file types, such as Parquet, the schemas between source files must match exactly.
- You cannot define import mapping rules for datasets with parameters. If the imported dataset with parameters is still accessible, you should be able to run jobs from it.

Creating Dataset with Parameters

From file system

When browsing for data on your default storage layer, you can choose to parameterize elements of the path. Through the Import Data page, you can select elements of the path, apply one of the supported parameter types and then create the dataset with parameters.

NOTE: Matching file path patterns in a large directory can be slow. Where possible, avoid using multiple patterns to match a file pattern or scanning directories with a large number of files. To increase matching

speed, avoid wildcards in top-level directories and be as specific as possible with your wildcards and patterns.

Tip: For best results when parameterizing directories in your file path, include the trailing slash (/) as part of your parameterized value.

Options:

- You can choose to search nested folders for files that match your specified pattern.

Tip: If your imported dataset is stored in a bucket, you can parameterize the bucket name, which can be useful if you are migrating flows between environments or must change the bucket at some point in the future.

For more information, see *Create Dataset with Parameters*.

From relational source

If you are creating a dataset from a relational source, you can apply parameters to the custom SQL that pulls the data from the source.

NOTE: Avoid using parameters in places in the SQL statement that change the structure of the data. For example, within a SELECT statement, you should not add parameters between the SELECT and FROM keywords.

For more information, see *Create Dataset with SQL*.

Matching parameters

When a dataset with parameters is imported for use, all matching source files or tables are automatically unioned together.

NOTE: Sources for a dataset with parameters should have matching schemas.

The initial sample that is loaded in the Transformer page is drawn from the first matching source file or table. If the initial sample is larger than the first file, rows may be pulled from other source objects.

Managing Datasets with Parameters

Datasets with parameters in your flows

After you have imported a dataset with parameters into your flow:

- You can review any parameters that have been applied to the dataset through the Parameterization in Flow view.
- When the dataset with parameters is selected, you can use the right panel to review and edit the parameters that are applied to it.
- You can override the default value applied to the parameter through Flow View. See *Manage Parameters Dialog*.

For more information, see *Flow View Page*.

Tip: You can review details on the parameters applied to your dataset. See *Dataset Details Page*.

Sampling from datasets with parameters

When a dataset with parameters is first loaded into the Transformer page, the initial sample is loaded from the first found match in the range of matching datasets. If this match is a multi-sheet Excel file, the sample is taken from the first sheet in the file.

With parameters:

To work with data that appears in files other than the first match in the dataset, you must create a new sample in the Transformer page. Any sampling operations performed within the Transformer page sample across all matching sources of the dataset.

With variables:

If you have created a variable with your dataset, you can apply a variable value to override the default at sampling time. In this manner, you can specify sampling to occur from specific source files from your dataset with parameters.

For more information, see *Overview of Sampling*.

Scheduling for datasets with parameters

Schedules can be applied to a dataset with parameters. When resolving date range rules for scheduling a dataset with parameters, the schedule time is used.

For more information, see *Add Schedule Dialog*.

Sharing for datasets with parameters

By default, when a flow containing parameters is copied, any changes to parameter values in the copied flow also affect parameters in the original flow. To separate these parameters, you have the following options:

1. Optionally, when the flow is copied, you can copy the underlying datasets.
2. As a workaround, you can export and import the flow into the same system and replace the datasets in the imported flow.

NOTE: For copying flows using parameterized datasets, you should duplicate the datasets, which creates separate copies of parameters and their values in the new flow. If datasets are not copied, then parameter changes in the copied flow modify the values in the source flow.

For more information, see *Overview of Sharing*.

Housekeeping

Since Trifacta never touches the source data, after a source that is matched for a dataset with parameters has been executed, you should consider removing it from the source system or adjusting any applicable ranges on the matching parameters. Otherwise, outdated data may continue to factor into operations on the dataset with parameters.

NOTE: Housekeeping of source data is outside the scope of Trifacta. Please contact your IT staff to assist as needed.

Flow Parameters

You can specify flow parameters and their default values, which can be invoked in the recipe steps of your flow. Wherever the flow parameter is invoked, it is replaced by the value you set for the parameter. Uses:

- Dynamically affect recipe steps
- Improve flow usability; build fewer flows and recipes to maintain
- Parameters are evaluated at design time in the Transformer page and at runtime during job execution
- All parameter values can be overridden, as needed.

Flow parameter types:

- Literal values: These values are always of String data type.

Tip: You can wrap flow parameter references in your transformations with one of the `PARSE` functions.

NOTE: Wildcards are not supported.

- Patterns . For more information, see *Text Matching*.
- Regular expressions.

Limitations

- Flow parameters are converted to constants in macros. Use of the macro in other recipes results in the constant value being applied.
- A flow parameter cannot be used in some transformation steps or fields.

Example

Suppose you need to process your flow across several regions of your country. These regions are identified using a region ID value: `pacific`, `mountain`, `central`, or `eastern`.

From the Flow View context menu, you select **Manage parameters**. In the Parameters tab, you specify the parameter name:

```
paramRegion
```

You must specify a default value. To verify that this critical parameter is properly specified before job execution, you set the default value to:

```
##UNSPECIFIED##
```

The above setting implies two things:

- If the above value appears in the output, then an override value for the parameter was not specified when the job was executed, which prevents the default value being used erroneously.
- Before the job is executed, you must specify an override value. You can specify an override:
 - At the flow level to assist in recipe development.
 - At run time to insert the proper region value for the job run.

After the flow parameter has been created, you can invoke it in a transformation step using the following syntax.

```
$paramRegion
```

Where the parameter is referenced, the default or applicable override value is applied. For more examples, see *Create Flow Parameter*.

Upstream flow parameters

If your flow references a recipe or dataset that is sourced from an upstream flow, the flow parameters from that flow are available in your current flow. That value of the parameter at time of execution is passed to the current flow.

NOTE: Downstream values and overrides of parameters that share the same name take precedence. When you execute the downstream flow, the parameter value is applied to the current flow and to all upstream objects. For more information, see "Order of Evaluation" below.

Creating flow parameters

Flow parameters are created at the flow level from the context menu in Flow View. See *Manage Parameters Dialog*.

Managing flow parameters

Flow parameters can be edited, deleted, and overridden through the Flow View context menu. See *Manage Parameters Dialog*.

Flow parameters in plans

You can also apply overrides to your flow parameters as part of your plan definition. For more information, see *Plan View Page*.

Output Parameters

You can specify variable and timestamp parameters to apply to the file or table paths of your outputs.

NOTE: Output parameters are independent of dataset parameters.

Parameter Types

You can create the following types of output parameters:

- **Datetime parameters:** Insert date and time values in output paths based on the job's start time.
- **Variables:** Define variable names and default values for an output parameter. Modify these values at runtime to parameterize execution.

Tip: These types of parameters can be applied to file or table paths. An output path can contain multiple parameters.

Example

Suppose you are generating a JSON file as the results of job execution.

```
/outputs/myFlow/myOutput.json
```

Since this job is scheduled and will be executed on a regular interval, you want to insert a timestamp as part of the output, so that your output filenames are unique and timestamped:

```
/outputs/myFlow/myOutput_<timestamp>.json
```

In this case, you would create an output parameter of timestamp type as part of the write settings for the job you are scheduling.

Creating output parameters

When you are creating or editing a publishing action in the Run Jobs page, you can click the **Parameterize destination** link that appears in the right panel.

Tip: For outputs that are stored in buckets, you can parameterize the name of the bucket.

For more information, see *Create Outputs*.

Using output parameters

Whenever you execute a job using the specified publishing action, the output parameters are applied.

After specifying variable parameters, you can insert new values for them at the time of job execution in the Run Job page.

For more information, see *Run Job Page*.

Bucket Name Parameters

In addition to parameterizing the paths to imported datasets or outputs, you can also apply parameters to the buckets where these assets are stored. For example, if you are developing flows in one workspace and deploying them into a production workspace, it may be useful to create a parameter for the name of the bucket where outputs are written for the workspace.

Bucket names can be parameterized for the buckets in the following datastores:

- S3
- ADLS Gen2

Tip: You can parameterize the user info, host name, and path value fields as separate parameters.

Bucket names can be parameterized as variable parameters or as environment parameters. For more information on examples of parameterized bucket names, see "Environment Parameters" above.

For more information:

- *Parameterize Files for Import*
- *Create Outputs*

Parameter Overrides

For each of the following types of parameter, you can apply override values as needed.

Override Type	Description
dataset parameters	When you run a job, you can apply override values to variables for your imported datasets. See <i>Run Job Page</i> .
flow parameters	At the flow level, you can apply override values to flow parameters. These values are passed into the recipe and the rest of the flow for evaluation during recipe development and job execution.

	<p>NOTE: Overrides applied at the flow level are passed into all recipes and other objects in the flow. Wherever there is case-sensitive match between the name of the overridden parameter and a parameter name in the flow, the override value is applied. These values can be overridden by ad-hoc values. See "Order of Precedence" below.</p>
output parameters	When you define your output objects in Flow View, you can apply override values to the parameterized output paths on an as-needed basis when you specify your job settings. See <i>Run Job Page</i> .

Order of Parameter Evaluation

Wherever a parameter value or override is specified in the following list, the value is applied to all matching parameters within the execution tree. Suppose you have created a parameter called `varRegion`, which is referenced in your imported dataset, recipe, and output object. If you specify an override value for `varRegion` in the Run Job page, that value is applied to the data you import (dataset parameter), the recipe during execution (flow parameter), and the path of the output that you generate (output parameter). Name matches are case-sensitive.

NOTE: Override values are applied to upstream flows, as well. Any overrides specified in the current flow are passed to downstream flows, where they can be overridden as needed.

Parameter values are evaluated based on the following order of precedence (highest to lowest):

NOTE: The following does not apply to environment parameters, which cannot be overridden.

1. **Run-time overrides:** Parameter values specified at run-time for jobs.

NOTE: The override value is applied to all subsequent operations in the platform. When a job is submitted to the job queue, any overrides are applied at that time. Changes to override values do not affect jobs that are already in flight.

NOTE: You can specify run-time override values when executing jobs through the APIs. See *API Workflow - Run Job*.

See *Run Job Page*.

2. **Flow level overrides:** At the flow level, you can specify override values, which are passed into the flow's objects. These values can be overridden by overrides set in the above locations. See *Manage Parameters Dialog*.
3. **Default values:** If no overrides are specified, the default values are applied:
 - a. Imported datasets: See *Create Dataset with Parameters*.
 - b. Flow parameters: See *Manage Parameters Dialog*.
 - c. Output parameters: See *Run Job Page*.
4. **Inherited (upstream) values:** Any parameter values that are passed into a flow can be overridden by any matching override specified within the downstream flow.

Run Jobs with Parameters

When running a job based on datasets with parameters, results are written into separate folders for each parameterized path.

NOTE: During job execution, a canary file is written for each set of results to validate the path. For datasets with parameters, if the path includes folder-level parameterization, a separate folder is created for each parameterized path. During cleanup, only the the canary files and the original folder path are removed. The parameterized folders are not removed. This is a known issue.

Runtime Parameter Overrides

When you choose to run a job on a dataset with parameters from the user interface, any variables are specified using their default values.

Through the Run Job page, you can specify different values to apply to variables for the job.

NOTE: Override values applied to a job are not validated. Invalid overrides may cause your job to fail.

NOTE: Values applied through the Run Job page to variables override the default values for the current execution of the job. Default values for the next job are not modified.

NOTE: When you edit an imported dataset, if a variable is renamed, a new variable is created using the new name. Any override values assigned under the old variable name for the dataset must be re-applied. Instances of the variable and override values used in other imported datasets remain unchanged.

For more information, see *Run Job Page*.

Scheduling jobs on datasets with parameters

You can schedule jobs for datasets with parameters.

NOTE: When a job is executed, the expected time of execution is used during execution. For scheduled jobs, this value is the scheduled time. For example, if a job scheduled for 08:00 begins execution at 08:05, any parameters that reference "now" time use 08:00 during the job run.

For a scheduled job:

- Parameter values are evaluated based on the scheduled time of execution. Relative times are evaluated based on the scheduled time of execution.
- If there are interruptions in service due to maintenance windows or other reasons, scheduled jobs are queued for execution on restart. These queued jobs are attempted only once.

See *Schedule a Job*.

Parameters in Job Details

In the Job Details page:

- **Data sources tab:** For file-based parameterized datasets, you can review the files that were matched at runtime for the specified parameters.
- **Parameters tab:** View the parameter names and values that were used as part of the job, including the list of matching datasets.

See *Job Details Page*.

Operationalization with Parameters

APIs

Through the API, you can apply runtime parameters to datasets with parameters during job execution. For more information, see <https://api.trifacta.com/ee/es.t/index.html#operation/runJobGroup> For more information on working with parameters and the APIs, see *API Workflow - Run Job on Dataset with Parameters*.

Use of parameters to create imported datasets through the API is not supported.

For other parameter types, you can apply overrides as key-value pairs in the API request to execute a new job. See *API Workflow - Run Job*.

Configuration

Disable

By default, parameterization is enabled. This feature is covered by this setting: **Parameterization**.

For more information on disabling, see *Workspace Settings Page*.

Overview of Authorization

Contents:

- *Resource Roles and Privileges*
 - *Standard roles*
 - *Custom role(s)*
 - *Privileges*
 - *Example model*
-

Authorization governs how Trifacta® users can access platform features and user-defined objects in the Trifacta application.

NOTE: Authorization manages access to object types. It does not cover access to individual objects of a specified type. For example, access to a specific flow is governed by ownership of the flow (owner) and sharing of the flow by the owner (to a collaborator). If a flow is shared with a user who is not permitted to access flows, then the user cannot access the flow.

Resource Roles and Privileges

Access to Trifacta objects is governed by roles in the user account.

- A **role** is a set of zero or more privileges. A user may have one or more assigned roles.

NOTE: Roles are additive. If a user has multiple roles, the user has access at the highest level of privileges from each role.

- A **privilege** is an access level for a type of object. A role may have one or more privileges assigned to it.
- All accounts are created with the `default` role, which provides a set of basic privileges.

Standard roles

default role

All new users are automatically assigned the `default` role. By default, the `default` role enables full access to all types of Trifacta objects.

- If you have upgraded from a version of the product that did not support authorization, the `default` role represents no change in behavior. All existing users can access Trifacta objects as normal.

Since roles in a user account are additive, you may choose to reduce the privileges on the `default` role and then add privileges selectively by creating other roles and assigning them to users. See the example below.

NOTE: You can modify the `default` role. You can also remove it from a user account. You cannot delete the role.

NOTE: In future releases of the software, additional objects may be made available. A level of access may be defined in the `default` role. No other roles will be modified.

Workspace admin role

This admin role is a super-user. The admin role enables all capabilities of the `default` role, plus:

- access to all Trifacta application objects, unless specifically limited. See Resource Roles and Privileges below.
- administration functions and settings within the Trifacta application.

NOTE: This role enables for the user owner-level access to all objects in the project or workspace and access to all admin-level settings and configuration pages in the admin console. This role should not be assigned to many users. At least one user should always have this role.

NOTE: A platform administrator is automatically granted the admin role.

Custom role(s)

As needed, administrators can create custom roles for users of the project or workspace. For more information, see *Create Role*.

Privileges

For a complete list of privileges for each type of object, see *Privileges and Roles Reference*.

Example model

In the following model, three separate roles have been created. Each role enables the highest level of access to a specific type of object.

The `default` object has been modified:

- Since all users are automatically granted the `default` role, the scope of its permissions has been reduced here to view-only.
- There is no `viewer` privilege for Plans (`none`, `author`).

NOTE: Depending on your product edition, some of these privileges may not be applicable.

Privilege/Role	default	Role A	Role B	Role C	Notes
Flows	viewer	author	none	none	
Connections	viewer	none	author	none	Paid product editions only
Plans	none	none	none	author	Premium product editions only
User defined functions	viewer	none	none	author	Dataprep by Trifacta product editions only

User 1:

Roles: `default`

- User can see flows in Flows page. User cannot schedule, modify, or create new ones.
- User can see connections in the Connections page. User cannot schedule, modify, or create new ones.
- User cannot access the Plans page.
- User can invoke UDFs but cannot create, modify or delete them.

User 2:

Roles: `default`, `Role A`

- User can create, schedule, modify, run jobs, and delete flows (full privileges).
- User can see connections in the Connections page. User cannot schedule, modify, or create new ones.
- User cannot access the Plans page.
- User can invoke UDFs but cannot create, modify or delete them.

User 3:

Roles: Role A, Role B, Role C

- User can create, schedule, modify, run jobs, and delete flows (full privileges).
- User can create, modify, and delete connections (full privileges).
- User can create, schedule, modify, run jobs, and delete plans (full privileges).
- User can create, modify, and delete UDFs.

Overview of Operationalization

Contents:

- *Single-Flow Operations*
 - *Parameterization*
 - *Scheduling*
 - *Job monitoring*
 - *Email notifications*
 - *Webhooks*
 - *Deployment Manager*
 - *Orchestration*
 - *Terms*
 - *Task types*
 - *Limitations*
 - *Basic workflow*
 - *Plan scheduling*
 - *Plan execution*
 - *Enable*
 - *Logging*
-

Operationalization refers to a general class of platform features that enable repeated application of Trifacta® on production data. Whether deployed in a single flow or across all flows in your environment, operationalization features broaden the scope of wrangled data, simplify job execution, and enable these processes on a repeated or scheduled basis.

In the following sections, you can review short summaries of specific features and explore more detailed information on them.

Single-Flow Operations

These features can be applied to individual flows to simplify job execution.

Parameterization

Parameterization enables you to specify parameters that capture variability in your data source paths or names. For example, you can parameterize the names of folders in your filepaths to capture files within multiple folders. Or, you can parameterize your inputs to capture datasets named within a specific time range. Nested folders of data can be parameterized, too.

Parameter types:

- dataset parameters: Parameterize the input paths to your data, allowing you to process data in parallel files and tables through the same flow.
- output parameters: Parameterize the output paths for your results.
- flow parameters: Define parameters that can be applied in your flows, including recipe steps.

Tip: You can apply overrides to any parameter at the flow level. These parameter override values are applied to any parameter that is referenced within the flow for any supported parameter type.

Parameter formats:

NOTE: Some of the following may not be available in your product edition.

Parameter Type	Description
Pattern	Use regular expressions or Patterns in your paths or queries to sources to capture a broader set of inputs.
Wildcard	Replace parts of your paths or queries with wildcards.
Datetime	You can specify parameterized Datetime values in one of the supported formats.
Variable	Variable values can be specified as overrides during import, job execution, and output.

Parameterization is available for the following:

File systems

Input	Output
Date/time	Timestamp
Pattern	Variable
Variable	

Relational sources

Input	Output
Timestamp	Timestamp
Variable	Variable

NOTE: For relational data, parameterization is applied to custom SQL queries used to import the data. For more information, see *Enable Custom SQL Query*.

For more information, see *Overview of Parameterization*.

Scheduling

The scheduling feature, also known as **Automator**, enables you to schedule the execution of individual flows on a specified frequency. Frequencies can be specified through the Trifacta application through a simple interface or, if needed, in a modified form of cron syntax.

Tip: Automator is often used with parameterization to fully automate data preparation processes in Trifacta.

For more information, see *Overview of Automator*.

Job monitoring

After a job has been launched, detailed monitoring permits you to track the progress of your job during all phases of execution. Status, job stats, inputs, outputs and a flow snapshot are available through the Trifacta application. For more information, see *Overview of Job Monitoring*.

Email notifications

After a job has completed, you can send email notifications to stakeholders based on the success or failure of the job.

NOTE: This feature must be enabled. See *Workspace Settings Page*.

These notifications are defined within Flow View. See *Email Notifications Page*.

Webhooks

Webhook notifications let you define outgoing HTTP messages to any REST API. The message form and body can be customized to include job execution metadata. For more information, see *Create Flow Webhook Task*.

Deployment Manager

The **Deployment Manager** is a separate environment that can be enabled for the execution of production flows under limited access. Flows in development are exported from your default (Dev) instance and then imported to the Production instance, the Deployment Manager, where you can configure the periodic execution of the flow. For more information, see *Overview of Deployment Manager*.

Orchestration

Orchestration is a set of functionality that supports the scheduled execution of jobs across multiple flows. These jobs could be external processes, other flows, or even HTTP requests.

Terms

Term	Description
plan	A plan is a sequence of tasks that are executed on one or more flows to which you have access. To orchestrate tasks, you build a plan. A plan can be scheduled for execution, triggered manually, or invoked via API.
trigger	A task is executed based on a trigger. A trigger is a condition under which a task is executed. In many cases, the trigger for a task is based on the schedule for the plan.
task	A task is a unit of execution in the platform. For example, one type of task is the execution of a flow, which executes all recipes in the flow, as well as the flow's upstream dependencies.
snapshot	A snapshot of the plan is captured, and the plan is executed against this snapshot. For more information on snapshots, see "Plan execution" below.

Task types

The following types of tasks are available.

Type	Description
flow task	An ad-hoc or scheduled execution of the transformations required to produce one or more selected outputs from a flow.
HTTP task	A request submitted to a third-party server as part of a plan run.
Slack task	Send a message with information about the plan run to a specified Slack channel.
Delete task	Delete files and folders from backend data storage.

Limitations

- You cannot specify parameter overrides to be applied to plans specifically.
 - Plans inherit parameter values from the objects referenced in the plan's tasks.
 - If overrides are applied to flow parameters, those overrides are passed to the plan at the time of flow execution.

Tip: Prior to plan execution, you can specify parameter overrides at the flow level. These values are passed through to the plan for execution. For more information, see *Manage Parameters Dialog*.

Basic workflow

You create a plan and schedule it using the following basic workflow.

1. Create the plan. A **plan** is the container for definition of the tasks, triggers, and other objects. See *Plans Page*.
2. In Plan View, you specify the objects that are part of your plan. See *Plan View Page*.
 - a. **Schedule:** The schedule defines the set of triggers that queue the plan for execution.
 - i. **Trigger:** A trigger defines the schedule and frequency at which the plan is executed. A plan can have multiple triggers (e.g. monthly versus weekly executions).
 - b. **Task(s):** Next, you specify the tasks that are executed in order.
 - i. **Flow task:** A flow task includes the specification of the flow to run and the outputs from the flow to generate.

NOTE: You can select the outputs from the recipe that you wish to generate. You do not need to generate all outputs.

NOTE: When a flow task is executed, the execution plan works back from the selected outputs to execute all of the recipes required to generate the output, including the upstream dependencies of those recipes.

See *Plan View for Flow Tasks*.

- ii. **HTTP task:** An HTTP task is a request issued when it is triggered from the application to a target endpoint. This request supports a variety of API methods. See *Plan View for HTTP Tasks*.
 - iii. **Slack task:** A Slack task is a message between Trifacta and a specified Slack channel that is triggered within the plan. See *Plan View for Slack Tasks*.
 - iv. **Delete task:** A Delete task deletes specific files or folders from backend storage. See *Plan View for Delete Tasks*.
 - v. Continue building tasks in a sequence.
3. As needed, you can apply override values to any flow parameters that are included in the tasks of your recipe. These overrides are applied during a plan run. For more information, see *Manage Parameters Dialog*.
4. To test:
 - a. Click **Run now**.
 - b. To track progress, click the Runs link.
 - c. In the Run Details page, you can track the progress.
 - d. The first task is executed and completes, before the second task is started.
 - e. Individual tasks are executed as separate jobs, which you can track through the Jobs page. See *Jobs Page*.
 - f. When the plan has completed, you can verify the results through the Job details page. See *Job Details Page*.
5. If you are satisfied with the plan definition and your test run, the plan will execute according to the scheduled trigger.

Plan scheduling

Through the Plan View page, you can configure the scheduled executions of the plan. Plan schedules are defined using triggers.

- These schedules are independent of schedules for individual flows.
- You cannot create schedules for individual tasks.

Plan execution

When a plan is triggered for execution, a **snapshot** of the plan is taken. This snapshot is used to execute the plan. Tasks are executed in the sequence listed in Plan View.

Important notes:

NOTE: Any subsequent changes to the flows, datasets, recipes, and outputs referenced in the plan's tasks can affect subsequent executions of the plan. For example, subsequent removal of a dataset in a flow referenced in a task can cause the task to fail to execute properly.

At the flow level, you can define webhooks and email notifications that are triggered based on the successful generation of outputs. When you execute a plan containing an output with one of these messages, the message is triggered and delivered to stakeholders.

NOTE: Webhook messages and email notification cannot be directly triggered based on a plan's execution. However, you can create HTTP-based tasks to send messages based on a plan task's execution.

Tip: When a flow email notification is triggered through a plan, the internal identifier for the plan is included in the email.

See "Webhooks" and "Email notifications" above.

Enable

Enable the following setting:

Plans feature

Plan sharing, import, and export must also be enabled. For more information, see *Workspace Settings Page*.

The following flags must be enabled for the orchestration service to correctly function.

Steps:

- 1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
- 2. Locate the following settings. Verify that they are set to `true`:

```
"webapp.orchestrationWorkers.enabled": true,  
"orchestration-service.enabled": true,  
"orchestration-service.autoRestart": true,
```

- 3. You can choose to enable the following task types:

Task type	Setting	Description
HTTP task	<code>feature.orchestration.httpTasks.enabled</code>	

		When <code>true</code> , you can configure plan tasks to deliver a REST request over HTTP or HTTPS to a specified endpoint, including endpoints in the Trifacta platform.
Slack task	<code>feature.orchestration.slackTask.enabled</code>	When <code>true</code> , you can configure plan tasks to deliver messages to a specified Slack channel.
Delete task	<code>feature.orchestration.deleteFileTask.enabled</code>	When <code>true</code> , you can configure plan tasks to delete files or folders from backend data storage.
	<code>feature.orchestration.deleteFileTask.maxFiles</code>	By default, the maximum number of files that can be matched for deletion is 100. You can modify this value if needed. NOTE: This setting is intended as a safety measure to prevent runaway deletion of a large number of files. Modify this value only if necessary.
Email task	<code>feature.orchestration.emailTask.enabled</code>	This feature is not yet available.
Flow task	This feature is automatically enabled when <code>Plans</code> feature is enabled. See above.	These tasks execute a specific output on a selected flow.

4. Save your changes and restart the platform.

Logging

Logging information on plan execution is captured in the `orchestration-service.log`. This log file can be downloaded as part of a Support Bundle. For more information, see *Support Bundle Contents*.

You can configure aspects of how this log captures service events. For more information, see *Configure Logging for Services*.

Overview of Macros

Contents:

- *Limitations*
 - *Enable*
 - *Examples*
 - *Example 1 - Reformat headers*
 - *Example 2 - Redact data for sensitive column data types*
 - *Create*
 - *Macro inputs*
 - *Apply*
 - *Sharing*
 - *Import/Export*
 - *Manage*
-

In Trifacta®, a **macro** is a saved sequence of one or more recipe steps that can be reused in other recipes. As needed, values in the recipe steps can be modified, so that instances of the macro can be configured for the recipe requirements.

Limitations

- You cannot create macros from steps that contain the following:
 - Multi-dataset operations like join, union, and lookup
 - Data-dependent transformations like header, valuestocols, and pivot.
 - Other macros

NOTE: In macros, Rename Columns transformations do not work. This is a known issue.

- You cannot create macros in flows that you do not own.
- Macro input limitations on the following types:
 - limits
 - enums
 - arrays
- Sharing of macros is not supported.
 - When working with a flow that was shared with you, you can only use the macros that belong to the flow's owner.
- When a flow containing a macro is imported, the macro steps are expanded.

Enable

This feature is enabled by default. To disable this feature, please complete the following steps.

Steps:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
2. Locate the following setting:

```
"feature.macros.enabled": true,
```

3. Export and import of macros is controlled by a separate setting. Locate the following setting and set it to true:

```
"feature.macros.exportable": true,
```

4. Save your changes and restart the platform.

Examples

Example 1 - Reformat headers

Suppose one of your downstream systems has the following requirements for column headers:

- No spaces. Underscore is ok.

You can do the following:

1. For the recipe on which you are working, create a new recipe.
2. In this new empty recipe, add the steps to configure your headers according to the above requirements.
 - a. No spaces. Underscores are ok:

Transformation Name	Rename columns based on a pattern
Parameter: Option	Find and replace
Parameter: Columns	All
Parameter: Find	' '
Parameter: Replace with	'_'
Parameter: Match all occurrences	true

3. Select the above step. In the context menu for it, select **Create or replace macro**.
 - a. Enter a Name and optional Description value. Click **Next**.
 - b. In the Create Macro dialog, you can review the step and its specified field values.
 - c. To save the macro, click **Save**.
4. For any recipe that must generate results for this downstream system, you can insert this macro as the last step before publication. For example, you can delete the recipe where you made the macro and insert the macro reference in the preceding recipe.

Example 2 - Redact data for sensitive column data types

For security reasons, you may decide that sensitive information must be redacted before it is delivered as an output for downstream consumption. For the following data types, you may wish to remove the sensitive information at the end of your transformation process:

- Credit card numbers
- Social Security numbers

1. For the recipe on which you are working, create a new recipe.
2. In this new empty recipe, add the following steps.
 - a. Redact social security numbers:

Transformation Name	Edit formula
Parameter: Columns	All

Parameter: Formula	IF (ISVALID (\$col , 'SSN') , '##REDACTED##' , \$col)
---------------------------	--

- b. Redact credit card numbers: For this one, you can use the following transformation to mask the numbers except for the last four digits using Patterns :

Transformation Name	Replace text or patterns
Parameter: Columns	All
Parameter: Find	`{start}{digit}{4}{any}{digit}{4}{any}{digit}{4}{any}({digit}{4}){end}`
Parameter: Replace with	XXXX-XXXX-XXXX-\$1

NOTE: The above transformation matches values based on the structure of the data, instead of the data type. If for some reason, you have values that are not credit card numbers yet follow the credit card pattern, those values will be masked as well by this transformation.

3. Select the above steps. In the context menu, select **Create or replace macro**.
 - a. Enter a Name and optional Description value. Click **Next**.
 - b. In the Create Macro dialog, you can review the step and its specified field values.
 - i. You may wish to parameterize the Find and Replace with values. For example, for some uses of the macro, you may wish to replace with an empty string or a value like ##REDACTED ## like the previous macro.
 - c. To save the macro, click **Save**.
4. For any recipe that must generate results for this downstream system, you can insert this macro as the last step before publication. For example, you can delete the recipe where you made the macro and insert the macro reference in the preceding recipe.

Create

A macro is created from a sequence of steps inside a recipe.

- The steps do not have to occur consecutively in the recipe.
- Recipe steps are added to the macro in the order that they are listed in the recipe.
- Some recipe steps cannot be added to a macro, so the option to create a macro with these types of steps is not available.

For more information, see *Create or Replace Macro*.

Macro inputs

When you create a macro, you can define macro inputs to contain values to be used in the macro's steps. Values for these inputs can be specified with each instance of a macro. For example, if you use MacroA at Step 2 and Step 37 of your recipe, you can specify different values for inputs to MacroA at the Step 2 and Step 37 instance of it.

- **Create macro inputs:** Macro inputs can be defined when you create a new macro.
- **Reuse or replace macro inputs:** When you replace a macro, you can reuse or replace the existing macro inputs in the new version of a macro.
 - If you are reusing the existing macro inputs, you must map them to the new steps in the new version of the macro.
 - If you are replacing macro inputs, instances of the macro that were added to your recipes under the old definition must be updated.

Apply

After a macro is created, you can apply an instance of it anywhere in your recipes. See *Apply a Macro*.

Sharing

Macros cannot be independently shared.

Copy a flow:

All macros are included. Steps are not expanded.

Share a flow:

When a flow is shared, the flow owner's macros are available for use by any collaborator in the recipes of the shared flow.

Import/Export

NOTE: Exported macros can be imported into a release that is later than the source release of the product. Exported macros cannot be imported into earlier releases.

Export:

- You can export individual macros from the Macros page. See *Export Macro*.
- When a flow containing a recipe that references macros is exported, macros are exported as expanded steps.

Import:

- Exported macros can be imported into a new environment through the Macros page. See *Import Macro*.
- When a flow containing macros is imported, the expanded steps are imported normally.

Manage

After macros have been created, you can manage them through the Library. For more information, see *Macros Page*.

Overview of Deployment Manager

Contents:

- *Dev/Test and Prod Deployments*
 - *Implementation in the platform*
 - *Terminology*
 - *Production environment terminology changes*
 - *Deployment Objects*
 - *Enable Deployment*
 - *User management*
 - *Import/Export*
 - *Exported Flows*
 - *Connections*
 - *Import*
 - *Value and Object Mapping Rules*
 - *Production Environment*
 - *Version Management*
 - *Flow View Page*
 - *Example Workflow*
 - *Recommended Practices*
 - *Job Execution*
 - *On-demand jobs*
 - *Scheduled jobs*
 - *Automation*
-

You can deploy flows that you have created into a separate, production environment where jobs for those flows can be executed on a periodic or scheduled basis. In this manner, you can create separation between your development and production environments and their flows. The **Deployment Manager** includes the tools to migrate your software between environments, manage releases of it, and separately control access to development and production flows.

- - Deployment Manager enables the transfer of flows between development and production instances of the platform. A customer may have one or more instances of the platform.
- For managing user access to flows within the same development instance, you can use sharing. See *Overview of Sharing*.
- This feature was formerly known as, "deployment management."

Key Features:

- Development environment:
 - Export of flows and all dependent objects
 - Import back into Development deployments for further development
- Production environment:
 - Import of flows
 - Import global and object-level mapping rules
 - Manage releases of flows
 - Rollback to previous versions as needed
- APIs to manage deployments

Dev/Test and Prod Deployments

In a typical environment, deployments may be segmented between Development (Dev), Testing (Test), and Production (Prod) environments. With respect to the Trifacta platform, these deployments break down into the following:

NOTE: In some cases, Dev and Test may be the same instance.

NOTE: Multiple browser tabs or windows open to different versions of the product is not supported.

Platform instance	Description
Development (Dev)	<p>New flows and recipes are created in a Development instance of the platform. Experiments can be undertaken without concern that production use of the recipe or flow is affected.</p> <div><p>Tip: You should do all of your recipe development and testing in Dev/Test. Avoid making changes in a Prod environment.</p></div> <p>Rules should be established on how flows, datasets, and recipes are organized and structured. Where are these assets stored? Where are shared versions of them made available? What are the rules by which items in Dev can be moved to Test /Prod?</p>
Testing (Test)	<p>In the Testing deployment, the objects in development are subjected to various stress tests. In the Trifacta platform, this testing can include load testing, malformed inputs, and changes to any parameters affecting the use of the object. For example, scheduled executions of flows should be thoroughly tested in this deployment.</p> <p>When errors are detected, they can be corrected in Dev or Test. Ideally, they are first applied in Test to address the issue at hand. Changes should then be applied back into the Dev deployment, so that future versions can consume the fix.</p>
Production (Prod)	<p>In the Production deployment, flows and their objects are presumed to be ready for regular, read-only use. After imported flows are reconfigured for the environment, they are ready for immediate use and require no further modification.</p> <ul style="list-style-type: none">• Management of flows and jobs is typically handled via API.• The UI should be used for checking and modifying settings and perform on-demand job executions to verify operations. <p>When errors are detected, you can:</p> <ul style="list-style-type: none">• Revert to a previous version of the flow• Apply any fixes in the Dev/Test instance for refinement and eventual updating back to the Prod instance.

Implementation in the platform

In the Trifacta platform, deployment management can be addressed in either of the following ways.

Implementation type	Description
Separate environments: Multiple instances of the platform	<p>Dev, Test, or both environments are separate instances of the Trifacta platform from the Production environment.</p> <p>Flows are migrated between environments using the export/import mechanisms.</p> <div><p>NOTE: Each platform instance is configured to be either a Dev instance or a Prod instance.</p></div>
All-in-one: Single instance of the platform, separate roles	<p>Dev, Test, and Prod are contained in a single instance of the Trifacta platform. This scenario can apply to cloud-based environments as well.</p>

A user can access either Dev/Test or Prod, but not both at the same time. In this scenario, a user can access Production deployments by having the Deployment account role.

Tip: Access to the Production environments should be tightly controlled to prevent inadvertant changes to Production jobs.

Terminology

Production environment terminology changes

A Prod environment focuses on management of the following objects. Differences between how these objects are used in a Dev environment are noted below.

NOTE: Some objects are available only in the Production environment. These objects are described later.

Object	Differences
Flows	<p>In a Prod environment, you can review a flow through Flow View.</p> <div>NOTE: Avoid making changes to your flows in a Prod environment. Any changes in the Prod version should be exported and then imported to the Dev version. Otherwise, when the next release is imported as a package into the Prod environment, those changes are lost.</div>
Jobs	<p>In the Prod environment, you can execute jobs against Prod flows. For the version of the flow that is active, you trigger a job for its overall deployment. Details are below.</p> <p>These jobs are accessible through an interface that is very similar to a Dev environment.</p>

The following flow objects from the Dev environment must be replaced in the Prod environment:

Dev Object	Replacement
Connections	Any connections used in the Dev system must be recreated or replaced with connections in the Production system.
Output Objects	Output objects from the Dev flow must be recreated or replaced in Flow View in the Prod environment.
Imported Datasets	If the Prod environment is not using the same sources as the Dev environment, you must create import rules to remap the point the flow to use imported datasets that are stored in a different location for the Prod environment.

Deployment Objects

In a Prod environment, you can explore the following objects, which are organized in a hierarchy:

Level	Item	Description
1	deployment	A deployment is a versioned set of releases that have been uploaded to the Prod instance for use. You can think of it as a production instance of your primary flow and its dependencies.
2	release	<p>A release is a specific instance of a package that has been imported to the Prod instance. Each time you import, you create a new release within the deployment where you imported.</p> <p>A release is created whenever you import a package into a deployment. A package is a ZIP file containing a flow definition that has been exported from an instance of the Trifacta platform.</p>
3	flows	<p>Within a release, you can explore the primary flow and any upstream flows that were included in the package. Each flow can be explored through a version of the Flow View page.</p> <ul style="list-style-type: none">The primary flow is the flow that you chose to export in the Dev instance.

- A **secondary flow** is any flow that is included with the package for the primary flow because the primary one depends on it.

Enable Deployment

This feature must be enabled through configuration. When enabled, the user experience of the product changes significantly, and a number of features are no longer available, including the Transformer page and its ability to modify recipes.

Tip: When you initially set up a platform instance, you should decide whether it is a Dev instance, a Prod instance or both.

User management

For more information on how to configure user accounts for Deployment Manager, see *Configure Deployment Manager*.

Import/Export

To transfer your flows between instances, you must export the flow from one instance of the platform and import it into the other instance of the platform.

NOTE: If Dev and Prod are in the same instance, you must export the flow and import it into a deployment. These are separate processes.

NOTE: As part of the import process, you must define rules for how objects and values contained in the imported flow definition are remapped in the Prod environment. See below.

Exported Flows

Through Flow View or the Flows page, you can export the flow through the context menu. The export is a ZIP file called a **package**.

NOTE: You must be the owner of a flow to export it.

A package ZIP contains all objects required to reconstruct and use the flow in a new environment.

- It includes the exported flow and any flows on which it depends.
- It does not include data, samples, or jobs.

Upstream dependencies

If the outputs of an exported flow require imported datasets or recipes from another flow, that entire flow is included as part of the export package. This package includes objects that may not be required to run the primary exported flow.

Connections

In the target instance, connections must be created prior to import. You may need to create import mapping rules to use this connections. See *Connections Page*.

Import

How a flow is imported depends on the environment into which you are importing it and how you intend to use it.

NOTE: If a flow is imported into an instance that is different from the instance where it was created, you must first create remapping rules for values and objects contained in the flow definition. More information is provided below.

For more information, see *Import Flow*.

Value and Object Mapping Rules

When objects are moved between environments, paths and other object-related references may require updating to point to the new environment.

NOTE: Import mapping rules do not work for parameterized datasets. If the imported dataset with parameters is still accessible, you should be able to run jobs from it.

For example, a dataset in the Dev environment may be pointing to the following location:

```
hdfs:///mydata-dev/1/00005a1a-81b0-4e4d-9c9b-f42ce55e1dde/Open_Order.csv
```

For the Prod version, the flow may need to be changed to the following:

```
hdfs:///mydata-prod/1/11115z4a-92f5-9f91-7v7f-g22fk99f2rru/Open_Order.csv
```

To support this kind of remapping, you can specify import rules at the level of individual deployments.

NOTE: For each deployment that you create, you must define new import remapping rules.

These rules can be specified using literal values, Trifacta patterns, or regular expressions. For more information, see *Define Import Mapping Rules*.

Production Environment

When a user accesses a Production environment, the UI is changed to include only the following pages:

NOTE: You cannot modify recipes within a Prod instance because the Transformer page is not available. The Prod flow must be exported and re-imported into a Dev instance.

Page	Description
<i>Deployment Manager Page</i>	On this page, you create deployments, for which you manage import of packages, activation of releases, and rollback to previous release as needed. For more information on the deployment objects, see below.

<i>Flow View Page</i>	<p>Within a specific release, you can review and update the flow definition, including specification of outputs and schedules. Flow View for a Prod instance has some restrictions.</p> <div> <p>NOTE: Use of scheduling through Flow View of a Prod instance is not supported. When a new release of a flow is imported, the schedule still points to the older release and is orphaned until the old release is reactivated or the schedule or release is removed.</p> </div>
<i>Jobs Page</i>	Same as Dev instance. No changes.
<i>Connections Page</i>	Connections that have been included as part of imported packages are available for review through the Production environment.
<i>Admin Settings Page</i>	<p>Same as Dev instance. No changes to the interface.</p> <div> <p>NOTE: In a multi-instance environment, some settings do not apply to the Prod environment.</p> </div>

Version Management

When you explore a deployment, you can see the list of releases pertaining to the deployment, with the active release listed at the top of the list. The **active** release is the one that is triggered for execution when a job is run.

You can roll back to using previous releases. Select **Activate** from the context menu for the desired release.

NOTE: Do not use scheduling features available through the user interface in a Production instance. If you have defined schedules through Flow View in the Prod instance and then add a new release, the schedules in the previous release are still available. You must remove them to prevent scheduled executions of outdated flows.

Flow View Page

In a Prod instance, you can drill into a release to review its flows through Flow View page.

NOTE: Avoid making modifications to the flow in a Prod instance.

Example Workflow

In this example, your environment contains separate Dev and Prod instances, each of which has a different set of users.

Item	Dev	Prod
Environment	http://wrangle-dev.example.com:3005	http://wrangle-prod.example.com:3005
User	User1 <div> <p>NOTE: User1 has no access to Prod.</p> </div>	Admin2
Source DB	devWrangleDB	prodWrangleDB
Source Table	Dev-Orders	Prod-Orders
Connection Name	Dev Redshift Conn	Prod Redshift Conn

Example Flow:

User1 is creating a flow, which is used to wrangle weekly batches of orders for the enterprise. The flow contains:

- A single imported dataset that is created from a Redshift database table.
- A single recipe that modifies the imported dataset.
- A single output to a JSON file.
- Production data is hosted in a different Redshift database. So, the Prod connection is different from the Dev connection.

Steps:

1. **Build in Dev instance:** User1 creates the flow and its steps.
2. **Export:** When User1 is ready to push the flow to production, User1 exports the flow from the Flows page and delivers the export package ZIP to Admin2. See *Export Flow*.
3. **Deploy to Prod instance:**
 - a. Admin2 creates a new deployment in the Prod instance. See *Deployment Manager Page*.
 - b. Admin2 creates a new connection (Prod Redshift Conn) to the Redshift database ProdWrangleDB. See *Create Connection Window*.
 - c. Admin2 creates an import rule to map the old connection (Dev Redshift Conn) to the new one (Prod Redshift Conn). See *Define Import Mapping Rules*.
 - d. Admin2 uploads the export ZIP package provided by User1. See *Import Flow*.
 - e. The deployment now contains a single release.
4. **Test deployment:**
 - a. Through Flow View in the Prod instance, Admin2 runs a job.
 - b. In reviewing the profile results of the job, Admin2 discovers a problem with the recipe. One column contains a number of mismatched values.
 - c. Admin2 chooses to fix in Dev and re-import into Prod.

NOTE: Any changes made in Production that must appear in future releases must be applied back in the Dev environment, too. You can either 1) export the flow from Prod and import back into Dev, or 2) manually apply all Prod changes back to the Dev environment and export/import into Prod when ready.

5. **Fix in development:** Back in the Dev environment, Admin2 opens the recipe for the flow.
 - a. Admin2 adds a step to the recipe to delete the rows containing mismatched values for the column.
 - b. Admin2 runs a job and verifies that the problem is fixed. In the visual profile for the dataset, the mismatched rows are removed from the dataset.
6. **Deploy again:** Admin2 exports the flow and imports it again as a new release in the deployment.
 - a. Since import rules have already been created for this deployment, the connection is automatically re-mapped for this second import.
 - b. Admin2 runs a job. The results look fine.
 - c. Admin2 removes profiling from the output object, since profiling takes time and is unnecessary in this production environment.
7. **Set schedule:** Using cron, Admin2 sets a schedule to run the active release for this deployment once per week.
 - a. Each week, the Prod-Orders table must be refreshed with data.
 - b. The dataset is now operational in the Prod environment.

Recommended Practices

- If possible, you should maintain separate instances of the platform for Dev and Prod.

- If you must use the All-in-One method of managing Dev and Prod instances, you should maintain a small number of non-admin accounts that are specifically used for Deployment Manager.
- Avoid scheduling Prod executions through Flow View. While possible, these schedules continue to exist even if the version of the flow has been replaced by another. Consequently, schedules that were specified through the application continue to execute, even though the flow itself is outdated. Instead, scheduled executions should be specified at the command line through cron jobs pointing at the latest release of each at all times.
- Do not modify Flow View settings through a Prod instance. These settings are not applied back to the Dev version and are lost when the next release package is imported.

Job Execution

On-demand jobs

You can configure jobs on-demand through the Flow View page of a Production instance. See *Flow View Page*.

Scheduled jobs

In Dev:

When your flow is exported from a Dev instance, all scheduling-related data is removed from the export package.

In Prod:

In a Prod instance, an imported flow contains no schedules. You must configure schedules through the REST APIs to execute on the currently active release for each deployment.

NOTE: Do not schedule executions through Flow View in a Prod instance.

- Schedules defined in Flow View are applied to Active and Non-Active releases in Production environments.
- If the scheduled release is deactivated, the schedule still exists, and the jobs are executed on an flow that is now out-of-date.

Automation

Automation of Deployment Manager is supported through the APIs.

NOTE: When you run a deployment, you run the primary flow in the active release for that deployment. Running the flow generates the output objects for all recipes in the flow.

NOTE: Scheduled execution of jobs in a deployment environment must be managed through external tools such as cron. For more information on the endpoint to schedule, see <https://api.trifacta.com/ee/es.t/index.html#operation/runDeployment>

For more information on the APIs for Deployment Manager, see *API Reference*.

For more information on an API-based method for deploying flows, see *API Workflow - Deploy a Flow*.

Overview of Pattern Matching

Contents:

- Overview
 - Example Patterns
 - Patterns in the Platform
 - Column Profiling
 - Machine Learning
 - Pattern Matching by Data Type
 - Using Patterns
 - Selecting Data
 - Patterns in Column Details
 - Advanced Uses
 - User-Defined Patterns
-

Trifacta® utilizes columnar pattern matching to identify data patterns of interest to you and to surface them in the interface for use in building your recipes. Additionally, in your recipe steps, you can apply regular expressions or Patterns to locate patterns and transform the matching data in your datasets.

Overview

A **pattern** is a combination of abstracted character sets and literal characters that can summarize data patterns in a column. Patterns can be applied through one of two methods:

- **Regular expressions** are a standardized method of matching data. The syntax of regular expressions is both powerful and not easy to understand.
- **Trifacta patterns** are pattern-matching widgets that provide a layer of abstraction on top of regular expressions. Instead of having to specify the sometimes complex underlying regular expression, you can specify a simple token to represent the underlying expression.

Tip: While regular expressions are a widely used standard, Patterns are powerful simplifications that can limit the sometimes "greedy" matching issues in regular expressions.

- For more information on the supported patterns, see *Text Matching*.

This section provides an overview of the pattern matching features of the platform.

Example Patterns

Within a row, multiple patterns may be applied at different levels of abstraction to describe the data in all fields (columns) of the row. Suppose you have two records like the following:

```
[cz.laping@gmail.com,3987,1446319063821]
[ajuneauk@gmail.com,5289,1447275151508]
```

The above records can be described by any of the following patterns:

```
[{alpha-numeric}+,{4-digits},{13-digits}]
[{email},{4-digits},{13-digits}]
[{alpha-numeric}+@gmail.com,{4-digits},{13-digits}]
```


NOTE: The above patterns utilize the syntax of Patterns . Regular expressions can be used to describe them as well.

In the above case, all three pattern sets capture the data completely. However, please note the differences between the patterns for column 1:

Pattern	Description
{alpha-numeric}+	This pattern captures alpha-numeric values of one or more characters. So, entries that match on this pattern do not need to be valid email addresses.
{email}	This pattern ensures matching only on valid email addresses. So, values that do not match this pattern are likely to be flagged as mismatched within the platform.
{alpha-numeric}+@gmail.com	This partial pattern ensures that the only matches are from gmail.com.

Depending on the specific meaning of the data for your use, any of the above may apply.

Patterns in the Platform

Column Profiling

Pattern matching applied to columns can permit users to see the most common patterns and anomalous patterns of data in a column across the entire sample. Since patterns presented to the user encompass the entire set of values in the sample, you can gather detailed information about the consistency of data in the column across the column.

Tip: Column pattern profiling is especially useful after you have addressed the mismatched values in the column.

Based on the patterns surfaced for the column, you can take any of the following actions:

- **Filtering a subset of records.** For example, you can review patterns for a column of addresses and filter the rows of data where no street number is provided, based on patterns you select.
- **Standardize values.** You can make selections of patterns for the different patterns for phone numbers. See Pattern Matching by Data Type below.
- **Extract values.** You can break apart column values based on mismatches in structure. For example, apartment numbers from an address field can be extracted into a new column.
- **Variable levels of abstraction.** As demonstrated in the previous example, you may be able to select from multiple matching patterns to determine which one is the best fit for the row values of interest.

Machine Learning

Additionally, Trifacta collects aggregated information about patterns applied by all users. These patterns are given weight in the set of suggested patterns presented to each user.

Pattern Matching by Data Type

As part of pattern matching, the platform evaluates the data against the specified data type for the column. Type-specific pattern matching applies to the following data types:

- Datetime
- Phone

See *Standardize Using Patterns*.

Using Patterns

In the application, patterns can be used as the starting point in building your next recipe step, and you can modify or iterate on a pattern definition to preview the results of the specified transformation. Patterns are used in the following actions:

- Select text to trigger a pattern-based suggestion or suggestions
- Select patterns of varying level of abstraction to modify column data

Selecting Data

When you select a value in the data grid, your options include pattern-based suggestions. In this manner, you indicate something of interest and enable the platform to interpret your specific interest or broader goal for the selected data. These broader changes are surfaced as pattern-based suggestions in the context panel.

- See *Explore Suggestions*.
- See *Selection Details Panel*.
- For more information on how the platform predicts suggestion cards based on selection, see *Overview of Predictive Transformation*.

Patterns in Column Details

In the Column Details panel, you can review sets of patterns that describe subsets of the values in the column. When you select one of the patterns, you are prompted with a set of suggested transform steps to apply to the data. See *Column Details Panel*.

Advanced Uses

In addition to the above basic uses, patterns can be used as the basis for the following advanced uses and more.

Use	Description
Standardize records	Match values based on a pattern and then change values to fit this pattern. See <i>Standardize Using Patterns</i> .
Filter records	Keep or delete records based on patterns of values found in row data. See <i>Filter Data</i> .
Extract values	Extract values matching a pattern from one column and insert them into a new column of data. See <i>Extract Values</i> .
Generate function outputs	Use patterns to generate function outputs in new columns.

User-Defined Patterns

In your recipe steps, you can specify patterns using either of the following methods.

Regular expressions

Regular expressions (regexes) are sequence of characters that can be used to define a pattern. This pattern can be used in the transformations that support regex to identify patterns in your data of interest to you. Example:

```
replace col: myCol with:/$1/ on:/^((\d\d\d\d)\)/ global: false
```

In the above step, the matching pattern expressed in the `on` clause evaluates in the following manner:

- The forward slashes around the pattern indicate that it is a regular expression.

- `^` indicates the start of the value in the `myCol` column. So, the matching is only made at the beginning of the column.
- `\(` and `\)` are representations in regular expressions of the literal values for parentheses. So, matches are made on those specific characters.
- The interior set of parentheses are used to define a capture group of values. These values, which correspond to three digits, are captured and inserted as the replacement.

So, the net effect is to search the beginning of a field for values like `(555)` and replace them with just the digits: `555`. This replacement removes the parentheses from the area code part of a phone number.

NOTE: Regular expressions are very powerful tools for matching patterns. They can also cause unexpected results. Use of regular expressions is considered a developer-level skill. You should use the `Patterns` described below instead.

Trifacta implements a version of regular expressions based off of *RE2* and *PCRE* regular expressions.

Patterns

Use `Patterns` to quickly assemble sophisticated patterns to match in your data. The following example includes the equivalent `Pattern` as the previous regular expression:

```
replace col: myCol with:`$1` on:`^\\(({digit}{3})\\)` global: false
```

- The back-ticks around the pattern indicate that it is a `Pattern`.

Overview of RapidTarget

Contents:

- Overview
 - *Targets in the platform*
 - *Known Limitations*
 - *Creating Targets*
 - *Sources for creating a target*
 - *Creating a target for a recipe*
 - *Using a target*
 - *Running jobs on recipes with assigned targets*
 - *Configure*
 - *Configure fuzzy matching threshold*
 - *Disable*
-

In Trifacta®, a **target** is the set of columns, their order, and their formats to which you are attempting to wrangle your dataset. This target can be defined through imported or created datasets and must be assigned to an existing recipe. After it is assigned to a recipe, a target appears in the Transformer page to assist in your wrangling efforts. You can also apply changes to selected columns based on the target.

- This feature was formerly known as, "target matching."

Overview

In general, a target consists of the set of information required to define the expected data in a dataset. Often referred to as a "schema," this target schema information can include:

- Names of columns
- Order of columns
- Column data types
- Data type format
- Example rows of data

A dataset associated with a target is expected to conform to the requirements of the schema. Where there are differences between target schema and dataset schema, a validation indicator (or **schema tag**) is displayed.

Targets in the platform

In Trifacta, a target is created from the information in a dataset and can be applied to a recipe in a flow. When you are working with the flow, the target information is available for your wrangling activities, so that you can match up columns in your dataset (source) with their corresponding columns in the target. As you make changes in your recipe through the Transformer page, the target schema is available as a reference to see if your latest changes get you closer to matching the dataset to the target.

Known Limitations

- Targets are applied only after initial type inferencing has been applied to the loaded dataset.

Tip: As needed, you can disable initial type inferencing when data is imported into the product.

- Type-based matching applies a `settype` transform to any selected column. No pattern matching or standardization is applied. For more information, see *Overview of Pattern Matching*.

- Changes to the underlying objects of a target schema are not reflected in the schema. A target schema is a snapshot of the object at the time of its creation. To update, delete the target and create a new one.

Tip: If your target schema source is a recipe, then you can modify the recipe as needed and use it as your target again.

Creating Targets

Sources for creating a target

The schema used to define a target can be imported and assigned from any of the following objects, including:

- Output of a recipe in the same flow
- A reference dataset from another flow
- An imported dataset

Ideally, the source of the target schema should come from the publishing target. If you are publishing to a pre-existing target, you can create do one of the following:

- **Reference the target:** If the schema is represented in a dataset to which you have access in Trifacta, you can use it as your target schema.
- **Import the target:** Import the target table or schematized source into Trifacta as an imported dataset. Then, it can be selected as the target schema for any recipe to which you have access. See *Import Data Page*.
- **Extract target to a supported format:** If you cannot import the target directly into Trifacta, you could create an extract of a few rows, including the header, for the target into one of the formats supported for import. For more information, see *Supported File Formats*.

Creating a target for a recipe

You can create a target through one of the following mechanisms:

- **Flow View:** Select a recipe. From the context menu in the right panel, select **Assign Target to Recipe**. See *Flow View Page*.
- **Transformer Page:** Above the data grid, click the Target icon and select **Attach a new Target**.
 - See *Transformer Toolbar*.
 - You can do the same thing in the Column Browser panel.
- **Job Details Page:** After you have successfully run a job, you can create a new dataset from the Output Destinations tab. Through Flow View, this imported dataset can be used as the schema for wrangling. See *Job Details Page*.

For more information, see *Create Target*.

Using a target

After a target has been attached to a recipe, the target schema appears in a toolbar above the data grid along with a preview of the data. You can then make modifications to the data so that each column matches the definition for the corresponding column in the schema. See *Data Grid Panel*.

Through the data grid and the Column Browser, you can perform operations on selected columns in your dataset to align them with the target schema. For more information, see *Column Browser Panel*.

Running jobs on recipes with assigned targets

NOTE: You can run a job even if there are differences between the schema and your dataset. In Trifacta, no error checking is performed between schema and data prior to job execution. If you are publishing to a target that has a predefined schema, a publication error may be generated.

Configure

Configure fuzzy matching threshold

You can experiment with fuzzy matching thresholds to ensure that matches are occurring properly. This parameter applies a specific threshold value when two values are compared for matching. Lower values increase the probability of a match.

Steps:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
2. Adjust the value between 0.00 and 0.99 for the following parameter:

```
"feature.targetMatching.fuzzyMatchingThreshold": 0.30;
```

3. Save your changes and restart the platform.

Disable

If you prefer to disable this feature, please complete the following steps.

NOTE: If you are experiencing performance issues with target matching, you can first try to disable fuzzy matching, which can be resource-intensive.

Tip: If there is no schema associated with a recipe, then the target schema matching features are not displayed.

Steps:

1. You can apply this change through the *Admin Settings Page* (recommended) or `trifacta-conf.json`. For more information, see *Platform Configuration Methods*.
2. Set the following parameters to `false`:

```
"feature.targetMatching.enabled" : false,  
"feature.targetMatching.fuzzyMatchingEnabled" : false,
```

3. Save your changes and restart the platform.

Using Connections

This section contains some basic information on using common connections to various types of supported storage.

Using Databases

Contents:

- *Before You Begin*
 - *Access*
 - *Storing Data in Relational Databases*
 - *Reading from Database Tables and Views*
 - *Additional Notes on Database Views*
 - *Running Jobs from Database Sources*
 - *Writing to Databases*
-

This section describes how you interact with your databases through Trifacta®.

- Specific versions of each database are supported.
- Connections must be enabled and configured for each type of supported database.
- See *Connection Types*.

Before You Begin

- **Read Access:** Your database administrator must configure read permissions to the appropriate databases, tables and views for your use.

NOTE: To ensure that all user credentials used to access the database system are securely stored, you must first deploy the encryption key file to the Trifacta node. See *Relational Access*.

- **Write Access:** Some relational connection types support write access. For more information, see *Connection Types*.

Access

Database access is managed through connections.

- Individual users can create private connections through the application. See *Create Connection Window*.
- An administrator can make your connection public or create public connections through the application.

Storing Data in Relational Databases

NOTE: Trifacta does not modify source data nor store transformed data in the relational systems. Datasets sourced from database tables or views are read without modification from their source locations.

Reading from Database Tables and Views

You can create a Trifacta dataset from a table or view stored in a connected database.

Tip: In some scenarios, you can improve performance of loading from database tables by creating a view on the table to restrict the amount of data loaded to only the required fields. Additional, you can pre-filter the dataset using custom SQL statements. See *Create Dataset with SQL*.

Additional Notes on Database Views

- Some metadata, such as row counts, is not available for database views.
- For complex view definitions that require significant processing on the database, there may be a significant delay when previewing the contents of those views. In some cases, the preview may time out waiting for the database to respond with the view contents.

For more information, see *Database Browser*.

Running Jobs from Database Sources

NOTE: When executing a job using a relational source, the job may fail if one or more columns has been dropped from the underlying source table. As a workaround, the recipe panel may show steps referencing the missing columns, which be used to fix to either fix the recipe or the source data.

Writing to Databases

Relational connections can be configured to support writing results back to the database.

NOTE: You can only write to databases from the Run Job page. You cannot ad-hoc publish to a relational database.

NOTE: When writing to a new table in a relational target, the first entry in any mapping is used for writing out the value. Subsequent entries in the mapping are used for validation only on writing to new tables.

Natively supported connection types are automatically enabled for writeback.

Using HDFS

Contents:

- *Uses of HDFS*
 - *Before You Begin Using HDFS*
 - *Secure Access*
 - *Storing Data in HDFS*
 - *Ingest Caching*
 - *Reading from Sources in HDFS*
 - *Creating Datasets*
 - *Writing Job Results*
 - *Creating a new dataset from results*
 - *Purging Files*
-

This section describes how you interact through the Trifacta® platform with your HDFS environment.

- HDFS is a scalable file storage system for use across all of the nodes (servers) of a Hadoop cluster. Many interactions with HDFS are similar with desktop interactions with files and folders. However, what looks like a "file" or "folder" in HDFS may be spread across multiple nodes in the cluster. For more information, see https://en.wikipedia.org/wiki/Apache_Hadoop#HDFS.

Uses of HDFS

The Trifacta platform can use HDFS for the following reading and writing tasks:

1. **Creating Datasets from HDFS Files:** You can read in from a data source stored in HDFS. A source may be a single HDFS file or a folder of identically structured files. See *Reading from Sources in HDFS* below.
2. **Reading Datasets:** When creating a dataset, you can pull your data from another dataset defined in HDFS. See *Creating Datasets* below.
3. **Writing Job Results:** After a job has been executed, you can write the results back to HDFS. See *Writing Job Results* below.

In the Trifacta application, HDFS is accessed through the HDFS browser. See *HDFS Browser*.

NOTE: When the Trifacta platform executes a job on a dataset, the source data is untouched. Results are written to a new location, so that no data is disturbed by the process.

Before You Begin Using HDFS

- **Read/Write Access:** Your Hadoop administrator must configure read/write permissions to locations in HDFS. Please see the HDFS documentation provided with your Hadoop distribution.

Avoid using `/trifacta/uploads` for reading and writing data. This directory is used by the Trifacta application.

NOTE: Use of HDFS in safe mode is not supported.

- Your Hadoop administrator should provide a place or mechanism for raw data to be uploaded to your Hadoop datastore.

- Your Hadoop administrator should provide a writeable home output directory for you, which you can review. See *Storage Config Page*.

Secure Access

Depending on the security features you've enabled, the technical methods by which Trifacta users access HDFS may vary. For more information, see *Configure Hadoop Authentication*.

Storing Data in HDFS

Your Hadoop administrator should provide raw data or locations and access for storing raw data within HDFS. All Trifacta users should have a clear understanding of the folder structure within HDFS where each individual can read from and write their job results.

- Users should know where shared data is located and where personal data can be saved without interfering with or confusing other users.

NOTE: The Trifacta platform does not modify source data in HDFS. Sources stored in HDFS are read without modification from their source locations, and sources that are uploaded to the platform are stored in `/trifacta/uploads`.

Ingest Caching

If JDBC ingest caching has been enabled, users may see a `dataSourceCache` folder in their browser. This folder is used to store per-user caches of JDBC-based data that has been ingested into the platform from its source.

NOTE: The `datasourceCache` folder should not be used for reading and writing of datasets, metadata, or results.

For more information, see *Configure JDBC Ingestion*.

Reading from Sources in HDFS

You can create a dataset from one or more files stored in HDFS.

NOTE: To be able to import datasets from the base storage layer, your user account must include the `daAdmin` role.

Wildcards:

You can parameterize your input paths to import source files as part of the same imported dataset. For more information, see *Overview of Parameterization*.

Folder selection:

When you select a folder in HDFS to create your dataset, you select all files in the folder to be included. Notes:

- This option selects all files in all sub-folders. If your sub-folders contain separate datasets, you should be more specific in your folder selection.
- All files used in a single dataset must be of the same format and have the same structure. For example, you cannot mix and match CSV and JSON files if you are reading from a single directory.
- When a folder is selected from HDFS, the following file types are ignored:
 - `*_SUCCESS` and `*_FAILED` files, which may be present if the folder has been populated by Hadoop.

- If you have stored files in HDFS that begin with an underscore (`_`), these files cannot be read during batch transformation and are ignored. Please rename these files through HDFS so that they do not begin with an underscore.

Creating Datasets

When creating a dataset, you can choose to read data in from a source stored from HDFS or from a local file.

- HDFS sources are not moved or changed.
- Local file sources are uploaded to `/trifacta/uploads` where they remain and are not changed.

Data may be individual files or all of the files in a folder. For more information, see *Reading from Sources in HDFS*.

- In the Import Data page, click the HDFS tab. See *Import Data Page*.

Writing Job Results

When your job results are generated, they can be stored back in HDFS for you at the location defined for your user account.

- The HDFS location is available through the Output Destinations tab of the Job Details page. See *Job Details Page*.
- Each set of job results must be stored in a separate folder within your HDFS output home directory.
- For more information on your output home directory, see *Storage Config Page*.

If your deployment is using HDFS, do not use the `trifacta/uploads` directory. This directory is used for storing uploads and metadata, which may be used by multiple users. Manipulating files outside of the Trifacta application can destroy other users' data. Please use the tools provided through the interface for managing uploads from HDFS.

NOTE: Users can specify a default output home directory and, during job execution, an output directory for the current job. In an encrypted HDFS environment, these two locations must be in the same encryption zone. Otherwise, writing the job results fails with a `Publish Job Failed` error.

Access to results:

Depending on how the platform is integrated with HDFS, other users may or may not be able to access your job results.

- If user impersonation is enabled, results are written to HDFS through the HDFS account configured for your use. Depending on the permissions of your HDFS account, you may be the only person who can access these results.
- If user impersonation is not enabled, then each Trifacta user writes results to HDFS using a shared account. Depending on the permissions of that account, your results may be visible to all platform users.

Creating a new dataset from results

As part of writing job results, you can choose to create a new dataset, so that you can chain together data wrangling tasks.

NOTE: When you create a new dataset as part of your job results, the file or files are written to the designated output location for your user account. Depending on how your Hadoop permissions are configured, this location may not be accessible to other users.

Purging Files

Other than temporary files, the Trifacta platform does not remove any files that were generated or used by the platform, including:

- Uploaded datasets
- Generated samples
- Generated results

If you are concerned about data accumulation, please contact your HDFS administrator.

Using S3

Contents:

- *Uses of S3*
 - *Before You Begin Using S3*
 - *Secure Access*
 - *Storing Data in S3*
 - *Reading from Sources in S3*
 - *Creating Datasets*
 - *Writing Results*
 - *Creating a new dataset from results*
 - *Purging Files*
-

This section describes how you interact through the Trifacta® platform with your S3 environment.

- Simple Storage Service (S3) is an online data storage service provided by Amazon, which provides low-latency access through web services. For more information, see <https://aws.amazon.com/s3/>.

Uses of S3

The Trifacta platform can use S3 for the following tasks:

1. **Enabled S3 Integration:** The Trifacta platform has been configured to integrate with your S3 instance. For more information, see *S3 Access*.
2. **Creating Datasets from S3 Files:** You can read in source data stored in S3. An imported dataset may be a single S3 file or a folder of identically structured files. See *Reading from Sources in S3* below.
3. **Reading Datasets:** When creating a dataset, you can pull your data from a source in S3. See *Creating Datasets* below.
4. **Writing Results:** After a job has been executed, you can write the results back to S3.

In the Trifacta application, S3 is accessed through the S3 browser. See *S3 Browser*.

NOTE: When the Trifacta platform executes a job on a dataset, the source data is untouched. Results are written to a new location, so that no data is disturbed by the process.

Before You Begin Using S3

- **Access:** If you are using system-wide permissions, your administrator must configure access parameters for S3 locations. If you are using per-user permissions, this requirement does not apply. See *S3 Access*.

Avoid using `/trifacta/uploads` for reading and writing data. This directory is used by the Trifacta application.

- Your administrator should provide a writeable home output directory for you. This directory location is available through your user profile. See *Storage Config Page*.

Secure Access

Your administrator can grant access on a per-user basis or for the entire Trifacta platform.

The Trifacta platform utilizes an S3 key and secret to access your S3 instance. These keys must enable read /write access to the appropriate directories in the S3 instance.

NOTE: If you disable or revoke your S3 access key, you must update the S3 keys for each user or for the entire system.

Storing Data in S3

Your administrator should provide raw data or locations and access for storing raw data within S3. All Trifacta users should have a clear understanding of the folder structure within S3 where each individual can read from and write results.

- Users should know where shared data is located and where personal data can be saved without interfering with or confusing other users.
- The Trifacta application stores the results of each job in a separate folder in S3.

NOTE: The Trifacta platform does not modify source data in S3. Source data stored in S3 is read without modification from source locations, and source data uploaded to the Trifacta platform is stored in `/trifacta/uploads`.

Reading from Sources in S3

You can create an imported dataset from one or more files stored in S3.

NOTE: To be able to import datasets from the base storage layer, your user account must include the `dataAdmin` role.

NOTE: Import of glaciered objects is not supported.

Wildcards:

You can parameterize your input paths to import source files as part of the same imported dataset. For more information, see *Overview of Parameterization*.

Folder selection:

When you select a folder in S3 to create your dataset, you select all files in the folder to be included.

Notes:

- This option selects all files in all sub-folders and bundles them into a single dataset. If your sub-folders contain separate datasets, you should be more specific in your folder selection.
- All files used in a single imported dataset must be of the same format and have the same structure. For example, you cannot mix and match CSV and JSON files if you are reading from a single directory.

When a folder is selected from S3, the following file types are ignored:

- `*_SUCCESS` and `*_FAILED` files, which may be present if the folder has been populated by the running environment.

NOTE: If you have a folder and file with the same name in S3, search only retrieves the file. You can still navigate to locate the folder.

Creating Datasets

When creating a dataset, you can choose to read data in from a source stored from S3 or local file.

- S3 sources are not moved or changed.
- Local file sources are uploaded to `/trifacta/uploads` where they remain and are not changed.

Data may be individual files or all of the files in a folder. In the Import Data page, click the S3 tab. See *Import Data Page*.

Tip: Users can create secondary connections to specific S3 buckets. For more information, see *External S3 Connections*.

Writing Results

When you run a job, you can specify the S3 bucket and file path where the generated results are written. By default, the output is generated in your default bucket and default output home directory.

- Each set of results must be stored in a separate folder within your S3 output home directory.
- For more information on your output home directory, see *Storage Config Page*.

NOTE: The `append` action is not supported when publishing to S3.

If Trifacta installation is using S3, do not use the `trifacta/uploads` directory. This directory is used for storing uploads and metadata, which may be used by multiple users. Manipulating files outside of the Trifacta application can destroy other users' data. Please use the tools provided through the Trifacta application interface for managing uploads from S3.

NOTE: When writing files to S3, you may encounter an issue where the UI indicates that the job failed, but the output file or files have been written to S3. This issue may be caused when S3 does not report the files back to the application before the S3 consistency timeout has expired. For more information on raising this timeout setting, see *S3 Access*.

Creating a new dataset from results

As part of writing results, you can choose to create a new dataset, so that you can chain together data wrangling tasks.

NOTE: When you create a new dataset as part of your results, the file or files are written to the designated output location for your user account. Depending on how your permissions are configured, this location may not be accessible to other users.

Purging Files

Other than temporary files, the Trifacta platform does not remove any files that were generated or used by the platform, including:

- Uploaded datasets
- Generated samples
- Generated results

If you are concerned about data accumulation, you should create a bucket policy to periodically backup or purge directories in use. For more information, please see the S3 documentation.

Using SQL DW

Contents:

- *Limitations*
 - *Uses of Azure Synapse Analytics (Formerly Microsoft SQL DW)*
 - *Before You Begin Using Azure Synapse Analytics (Formerly Microsoft SQL DW)*
 - *Secure Access*
 - *Storing Data*
 - *Reading Data*
 - *Writing to Azure Synapse Analytics (Formerly Microsoft SQL DW)*
-

This section describes how you interact through the Trifacta® platform with your Azure® Synapse Analytics (Formerly Microsoft® SQL DW)® data warehouse.

- Azure Synapse Analytics (Formerly Microsoft SQL DW) is a scalable data warehouse solution available through Microsoft Azure. For more information, see <https://docs.microsoft.com/en-us/azure/sql-data-warehouse/sql-data-warehouse-overview-what-is>.
- Azure Synapse Analytics (Formerly Microsoft SQL DW) connections can interact with data stored as managed tables or external tables.
- Azure Synapse Analytics (Formerly Microsoft SQL DW) connections can use dedicated or serverless SQL pools.
- For more information, see *Microsoft SQL Data Warehouse Connections*.

Limitations

- Azure Synapse Analytics (Formerly Microsoft SQL DW) connections are available only if you have deployed the Trifacta platform onto Azure.
- The defined length of a table row cannot exceed 1 MB.

NOTE: In this release, this connection cannot be created through the APIs. Please create connections of this type through the application.

Uses of Azure Synapse Analytics (Formerly Microsoft SQL DW)

The Trifacta platform can use Azure Synapse Analytics (Formerly Microsoft SQL DW) for the following tasks:

1. Create datasets by reading from Azure Synapse Analytics (Formerly Microsoft SQL DW) tables.
2. Write to Azure Synapse Analytics (Formerly Microsoft SQL DW) tables with your job results.
3. Ad-hoc publication of data to Azure Synapse Analytics (Formerly Microsoft SQL DW) .

Before You Begin Using Azure Synapse Analytics (Formerly Microsoft SQL DW)

- **Enable Access:** Integration requires the following:
 - Installation of the Trifacta platform on Microsoft Azure.
 - Either ADL or WASB is supported as the base storage layer. For more information, see *Set Base Storage Layer*.
- **Read Access:** Your administrator must configure read permissions. Your administrator should provide a database for upload.

- **Write Access:** You can write and publish jobs results to Azure Synapse Analytics (Formerly Microsoft SQL DW) .

Secure Access

These connections require SSL access.

Storing Data

Your Azure Synapse Analytics (Formerly Microsoft SQL DW) administrator should provide database access for storing datasets. Users should know where shared data is located and where personal data can be saved without interfering with or confusing other users.

NOTE: The Trifacta platform does not modify source data. Datasets sourced from Azure Synapse Analytics (Formerly Microsoft SQL DW) connections are read without modification from their source locations.

Reading Data

You can create a Trifacta dataset from a managed or external table through Azure Synapse Analytics (Formerly Microsoft SQL DW) .

For more information, see *Database Browser*.

Writing to Azure Synapse Analytics (Formerly Microsoft SQL DW)

You can write back data to Azure Synapse Analytics (Formerly Microsoft SQL DW) using one of the following methods:

NOTE: Writing and publishing to Azure Synapse Analytics (Formerly Microsoft SQL DW) is not supported if Azure AD SSO has been enabled.

- Job results can be written directly to Azure Synapse Analytics (Formerly Microsoft SQL DW) as part of the normal job execution. Create a new publishing action to write to Azure Synapse Analytics (Formerly Microsoft SQL DW) . See *Microsoft SQL Data Warehouse Table Settings*.
- As needed, you can publish results to Azure Synapse Analytics (Formerly Microsoft SQL DW) for previously executed jobs.
- For more information on how data is converted to Azure Synapse Analytics (Formerly Microsoft SQL DW) , see *SQL DW Data Type Conversions*.

Data Validation issues:

- No validation is performed for the connection and any required permissions during job execution. So, you can be permitted to launch your job even if you do not have sufficient connectivity or permissions to access the data. The corresponding publish job fails at runtime.
- No data validation is performed during writing and publication to Azure Synapse Analytics (Formerly Microsoft SQL DW) . Your job fails if the schema for the Trifacta dataset varies from the target schema.
- Prior to publication, no validation is performed on whether a target is a table or a view, so the job that was launched fails at runtime.



Copyright © 2022 - Trifacta, Inc.
All rights reserved.