**Data Transformation – Manipulation**

**Data manipulation tasks**

The modules in this category are intended to support core data management tasks that might need to be performed in Machine Learning Studio (classic). The following tasks are examples of core data management tasks:

* Combine two datasets, either by using joins, or by merging columns or rows.
* Create new categories to use in grouping data.
* Modify column headings, change column data types, or flag columns as features or labels.
* Check for missing values, and then replace them with appropriate values.

The Data Transformation - Manipulation category includes the following modules:

* Add Columns: Adds a set of columns from one dataset to another.
* Add Rows: Appends a set of rows from an input dataset to the end of another dataset.
* Apply SQL Transformation: Runs a SQLite query on input datasets to transform the data.
* Clean Missing Data: Specifies how to handle values that are missing from a dataset. This module replaces Missing Values Scrubber, which has been deprecated.
* Convert to Indicator Values: Converts categorical values in columns to indicator values.
* Edit Metadata: Edits metadata that's associated with columns in a dataset.
* Group Categorical Values: Groups data from multiple categories into a new category.
* Join Data: Joins two datasets.
* Remove Duplicate Rows: Removes duplicate rows from a dataset.
* Select Columns in Dataset: Selects columns to include in a dataset or exclude from a dataset in an operation.
* Select Columns Transform: Creates a transformation that selects the same subset of columns as in a specified dataset.
* SMOTE: Increases the number of low-incidence examples in a dataset by using synthetic minority oversampling.

**Add Columns**

This describes how to use the Add Columns module in Machine Learning Studio (classic) to concatenate two datasets.

You combine all columns from the two datasets that you specify as inputs to create a single dataset. If you need to concatenate more than two datasets, use several instances of Add Columns.

When combining two datasets that contain a different number of rows, we recommend using the Join Data module, which supports outer joins on a common key column.

**How to configure Add Columns**

* Add the Add Columns module to your experiment.
* Connect the two datasets that you want to concatenate. If you want to combine more than two datasets, you can chain together several combinations of Add Columns.

1. It is possible to combine two columns that have a different number of rows. The output dataset is padded with missing values for each row in the smaller source column.
2. You cannot choose individual columns to add. All the columns from each dataset are concatenated when you use Add Columns. Therefore, if you want to add only a subset of the columns, use Select Columns in Dataset to create a dataset with the columns you want.

* Run the experiment.

**Results**

After the experiment has run:

* To see the first rows of the new dataset, right-click the output of Add Columns and select Visualize.
* To save and name the concatenated dataset, right-click the output and select Save as Dataset .

The number of columns in the new dataset equals the sum of the columns of both input datasets.

If there are two columns with the same name in the input datasets, a numeric suffix is added to the name of the column from the dataset used in the right input column. For example, if there are two instances of a column named TargetOutcome, the right column would be renamed TargetOutcome (1).

**Add Rows**

This describes how to use the Add Rows module in Machine Learning Studio (classic) to concatenate two datasets. In concatenation, the rows of the second dataset are added to the end of the first dataset.

Concatenation of rows is useful in scenarios such as these:

* You have generated a series of evaluation statistics, and you want to combine them into one table for easier reporting.
* You have been working with different datasets, and you want to combine the datasets to create a final dataset.

**How to use Add Rows**

To concatenate rows from two datasets, the rows must have exactly the same schema. This means, the same number of columns, and the same type of data in the columns.

* Drag the Add Rows module into your experiment, You can find it under Data Transformation, in the Manipulate category.
* Connect the datasets to the two input ports. The dataset that you want to append should be connected to the second (right) port.
* Run the experiment. The number of rows in the output dataset should equal the sum of the rows of both input datasets.

If you add the same dataset to both inputs of the Add Rows module, the dataset is duplicated.

**Apply SQL Transformation**

This describes how to use the Apply SQL Transformation module in Machine Learning Studio (classic), to specify a SQL query on an input dataset or datasets.

SQL is handy when you need to modify your data in complex ways, or persist the data for use in other environments. For example, using the Apply SQL Transformation module, you can:

* Create tables for results and save the datasets in a portable database.
* Perform custom transformations on data types, or create aggregates.
* Execute SQL query statements to filter or alter data and return the query results as a data table.

**What is SQLite?**

SQLite is a public domain relational database management system that is contained in a C programming library. SQLite is a popular choice as an embedded database for local storage in web browsers.

SQLite was originally designed in 2000 for the U.S. Navy, to support serverless transactions. It is a self-contained database engine that has no management system and hence requires no configuration or administration.

**How to configure Apply SQL Transformation**

The module can take up to three datasets as inputs. When you reference the datasets connected to each input port, you must use the names t1, t2, and t3. The table number indicates the index of the input port.

The remaining parameter is a SQL query, which uses the SQLite syntax. This module supports all standard statements of the SQLite syntax. For a list of unsupported statements, see the Technical Notes section.

**General syntax and usage**

* When typing multiple lines in the SQL Script text box, use a semi-colon to terminate each statement. Otherwise, line breaks are converted to spaces.

For example, the following statements are equivalent

Code

*SELECT*

*\* from*

*t1;*

*SELECT \* from t1;*

* You can add comments by using either -- at the beginning of each line, or by enclosing text using /\* \*/.

For example, this statement is valid:

*SELECT \* from t1*

*/\*WHERE ItemID BETWEEN 1 AND 100\*/;*

* If a column name duplicates the name of a reserved keyword, syntax highlighting is applied to the text inside the SQL Script text box. To avoid confusion, you should enclose column names with square brackets (to follow the Transact-SQL convention) or backticks or double quotation marks (the ANSI SQL convention).

For example, in the following query on the Blood Donation dataset, Time is a valid column name but is also a reserved keyword.

*SELECT Recency, Frequency, Monetary, Time, Class*

*FROM t1*

*WHERE Time between 3 and 20;*

If you run the query as is, the query might return the correct results, but depending on the dataset, it might return an error. Here are some examples of how to avoid the issue:

*-- Transact-SQL*

*SELECT [Recency], [Frequency], [Monetary], [Time], [Class]*

*FROM t1*

*WHERE [Time] between 3 and 20;*

*-- ANSI SQL*

*SELECT "Recency", "Frequency", "Monetary", "Time", "Class"*

*FROM t1*

*WHERE `Time` between 3 and 20;*

* SQLite is case insensitive, except for a few commands that have case-sensitive variants with different meanings (GLOB vs. glob).

**SELECT statement**

In the SELECT statement, column names that include spaces or other characters prohibited in identifiers must be enclosed in double quotation marks, square brackets, or backtick characters (`).

For example, this query references the Two-Class Iris dataset on t1, but one column name contains a prohibited character, so the column name is enclosed in quotation marks.

*SELECT class, "sepal-length" FROM t1;*

You can add a WHERE clause to filter values in the dataset.

*SELECT class, "sepal-length" FROM t1 WHERE "sepal-length" >5.0;*

The SQLite syntax does not support the TOP keyword, which is used in Transact-SQL. Instead, you can use the LIMIT keyword, or a FETCH statement.

For example, compare these queries on the Bike Rental dataset.

*-- unsupported in SQLite*

*SELECT TOP 100 [dteday] FROM t1 ;*

*ORDER BY [dteday] DESC;*

*-- Returns top 100*

*SELECT [dteday] FROM t1 LIMIT 100 ;*

*ORDER BY [dteday] DESC;*

*-- Returns top 100. Note that FETCH is on a new line.*

*SELECT [dteday] FROM t1 - ;*

*FETCH FIRST 100 rows ONLY;*

*ORDER BY [dteday] DESC;*

**Joins**

The following examples use the Restaurant Ratings dataset on the input port corresponding to t1, and the Restaurant Features dataset on the input port corresponding to t2.

The following statement joins the two tables to create a dataset that combines the specified restaurant features with average ratings for each restaurant.

*SELECT DISTINCT(t2.placeid),*

*t2.name, t2.city, t2.state, t2.price, t2.alcohol,*

*AVG(rating) AS 'AvgRating'*

*FROM t1*

*JOIN t2*

*ON t1.placeID = t2.placeID*

*GROUP BY t2.placeid;*

**Aggregate functions**

This section provides basic examples of some common SQL aggregate functions, using SQLite.

Aggregate functions currently supported are: AVG, COUNT, MAX, MIN, SUM, TOTAL.

The following query returns a dataset containing the restaurant ID, along with the average rating for the restaurant.

*SELECT DISTINCT placeid,*

*AVG(rating) AS ‘AvgRating’,*

*FROM t1*

*GROUP BY placeid*

**Working with strings**

SQLite supports the double pipe operator for concatenating strings.

The following statement creates a new column by concatenating two text columns.

*SELECT placeID, name,*

*(city || '-' || state) AS 'Target Region',*

*FROM t1*

**COALESCE and CASE**

COALESCE evaluates multiple arguments, in order, and returns the value of the first expression that does not evaluate to NULL.

For example, this query on the Steel Annealing Multi-Class dataset returns the first non-null flag from a list of columns assumed to have mutually exclusive values. If no flag is found, the string “none” is returned.

*SELECT classes, family, [product-type],*

*COALESCE(bt,bc,bf,[bw/me],bl, "none") AS TemperType*

*FROM t1;*

The CASE statement is useful for testing values and returning a new value based on the evaluated results. SQLite supports the following syntax for CASE statements:

* CASE WHEN [condition] THEN [expression] ELSE [expression] END
* CASE [expression] WHEN [value] THEN [expression] ELSE [expression] END

For example, suppose you had previously used the Convert to Indicator Values module to create a set feature columns containing true-false values. The following query collapses the values in multiple feature columns into a single multivalued column.

*SELECT userID, [smoker-0], [smoker-1],*

*CASE*

*WHEN [smoker-0]= '1' THEN 'smoker'*

*WHEN [smoker-1]= '1' THEN 'nonsmoker'*

*ELSE 'unknown'*

*END AS newLabel*

*FROM t1;*

**Unsupported statements**

Although SQLite supports much of the ANSI SQL standard, it does not include many features supported by commercial relational database systems. For more information, see SQL as Understood by SQLite. Also, be aware of the following restrictions when creating SQL statements:

* SQLite uses dynamic typing for values, rather than assigning a type to a column as in most relational database systems. It is weakly typed, and allows implicit type conversion.
* LEFT OUTER JOIN is implemented, but not RIGHT OUTER JOIN or FULL OUTER JOIN.
* You can use RENAME TABLE and ADD COLUMN statements with the ALTER TABLE command, but other clauses are not supported, including DROP COLUMN, ALTER COLUMN, and ADD CONSTRAINT.
* You can create a VIEW within SQLite, but thereafter views are read-only. You cannot execute a DELETE, INSERT, or UPDATE statement on a view. However, you can create a trigger that fires on an attempt to DELETE, INSERT, or UPDATE on a view and perform other operations in the body of the trigger.

**Join Data**

To perform a join on two datasets, they must be related by a single key column. Composite keys are not supported.

**How to configure Join Data**

* In Machine Learning Studio (classic), add the datasets you want to combine, and then drag the Join Data module into your experiment.

You can find the module in the Data Transformation category, under Manipulation.

* Connect the datasets to the Join Data module.

The Join Data module does not support a right outer join, so if you want to ensure that rows from a particular dataset are included in the output, that dataset must be on the lefthand input.

* Click Launch column selector to choose a single key column for the dataset on the left input.
* Click Launch column selector to choose a single key column for the dataset on the right input.
* Select the Match case option if you are joining on a text column and want to ensure that the join preserves case sensitivity.

For example, if you select this option, A1000 would be considered a different key value than a1000.

If you deselect this option, case sensitivity is not enforced, and A1000 would be considered the same as a1000.

* Use the Join type dropdown list to specify how the datasets should be combined. types:

1. Inner Join: An inner join is the typical join operation. It returns the combined rows only when the values of the key columns match.
2. Left Outer Join: A left outer join returns joined rows for all rows from the left table. When a row in the left table has no matching rows in the right table, the returned row contains missing values for all columns that come from the right table unless you specify a replacement value for missing values.
3. Full Outer Join: A full outer join returns all rows from the left table (table1) and from the right table (table2).

For each of the rows in the left table that have no matching rows in the right table, the join results include a row containing missing values from the right table.

For each of the rows in the right table that have no matching rows in the left table, the join results include a row containing missing values for all columns from the left table.

1. Left Semi-Join: A left semi-join returns only the values from the left table when the values of the key columns match.

For the option, Keep right key colums in joined table:

1. Deselect the option to get a single key column in the results.
2. Leave the option selected to view the keys from both input tables.

* Run the experiment, or select the Join Data module and selected Run Selected, to perform the join.
* To view the results, right-click the Join Data module, select Results dataset, and click Visualize.

**Select Columns in Dataset**

This describes how to use the Select Columns in Dataset module in Machine Learning Studio (classic), to choose a subset of columns to use in downstream operations. The module does not physically remove the columns from the source dataset; instead, it creates a subset of columns, much like a database view or projection.

This module is particularly useful when you need to limit the columns available for a downstream operation, or if you want to reduce the size of the dataset by removing unneeded columns.

The columns in the dataset are output in the same order as in the original data, even if you specify them in a different order.

**How to use Select Columns in Dataset**

This module has no parameters. You use the column selector to choose the columns to include or exclude.

**Choose columns by name**

There are multiple options in the module for choosing columns by name:

* **Filter and search**

Click the BY NAME option.

If you have connected a dataset that is already populated, a list of available columns should appear. If no columns appear, you might need to run upstream modules to view the column list.

To filter the list, type in the search box. For example, if you type the letter w in the search box, the list is filtered to show the column names that contain the letter w.

Select columns and click the right arrow button to move the selected columns to the list in the right-hand pane.

* To select a continuous range of column names, press Shift + Click.
* To add individual columns to the selection, press Ctrl + Click.

Click the checkmark button to save and close.

* Use names in combination with other rules

Click the WITH RULES option.

Choose a rule, such as showing columns of a specific data type.

Then, click individual columns of that type by name, to add them to the selection list.

* Type or paste a comma-separated list of column names

If your dataset is very wide, it might be easier to use indexes or generated lists of names, rather than selecting columns individually. Assuming you have prepared the list in advance:

* Click the WITH RULES option.
* Select No columns, select Include, and then click inside the text box with the red exclamation mark.
* Paste in or type a comma-separated list of previously validated column names. You cannot save the module if any column has an invalid name, so be sure to check the names beforehand.

You can also use this method to specify a list of columns using their index values.

**Choose by type** If you use the WITH RULES option, you can apply multiple conditions on the column selections. For example, you might need to get only feature columns of a numeric data type.

The BEGIN WITH option determines your starting point and is very important for understanding the results.

* If you select the ALL COLUMNS option, all columns are added to the list. Then, you must use the Exclude option to remove columns that meet certain conditions.

For example, you might start with all columns and then remove columns by name, or by type.

* If you select the NO COLUMNS option, the list of columns starts out empty. You then specify conditions to add columns to the list.

If you apply multiple rules, each condition is additive. For example, say you start with no columns, and then add a rule to get all numeric columns. In the Automobile price dataset, that results in 16 columns. Then, you click the + sign to add a new condition, and select Include all features. The resulting dataset includes all the numeric columns, plus all the feature columns, including some string feature columns.

**Choose by column index**

The column index refers to the order of the column within the original dataset.

* Columns are numbered sequentially starting at 1.
* To get a range of columns, use a hyphen.
* Open-ended specifications such as 1- or -3 are not allowed.
* Duplicate index values (or column names) are not allowed, and might result in an error.

For example, assuming your dataset has at least eight columns, you could paste in any of the following examples to return multiple non-contiguous columns:

* 8,1-4,6
* 1,3-8
* 1,3-6,4

the final example does not result in an error; however, it returns a single instance of column 4.

**Change order of columns**

The option Allow duplicates and preserve column order in selection starts with an empty list, and adds columns that you specify by name or by index. Unlike other options, which always return columns in their "natural order", this option outputs the columns in the order that you name or list them.

For example, in a dataset with the columns Col1, Col2, Col3, and Col4, you could reverse the order of the columns and leave out column 2, by specifying either of the following lists:

* Col4, Col3, Col1
* 4,3,1

**Select Columns Transform**

The purpose of the Select Columns Transform module is to ensure that a predictable, consistent set of columns is always used in downstream machine learning operations.

This module is particularly helpful for tasks such as scoring, which require specific columns. Changes in the available columns might break the experiment or change the results.

You use the Select Columns Transform to create and save a set of columns. Then, use the Apply Transformation module to apply those selections to new data.

**How to use Select Columns Transform**

This scenario assumes that you intend to use feature selection to generate a dynamic set of columns that will be used for training a model. To ensure that column selections are the same for the scoring process, you use the Select Columns Transform module to capture the column selections and apply them elsewhere in the experiment.

* Add an input dataset to your experiment in Studio (classic).
* Add an instance of Filter Based Feature Selection.
* Connect the modules and configure the feature selection module to automatically find some number of best features in the input dataset.
* Add an instance of Train Model and use the output of Filter Based Feature Selection as the input for training.
* Now, attach an instance of the Select Columns Transform module.

This generates a column selection as a transformation that can be saved or applied to other datasets. This step ensures that the columns identified by feature selection are saved for reuse by other modules.

* Add the Score Model module.

Do not connect the input dataset.Instead, add the Apply Transformation module, and connect the output of the feature selection transformation.

* Run the experiment.

This process of saving and then applying a column selection ensures that the same data schema is available for training and scoring.