Settype Transform

Contents:

Sets the data type of the specified column or columns. The column data is validated against the new data type, which can change the results of column profiling.Type is specified as a string literal or comma-separated set of literals. For more information on valid string literals, see *Valid Data Type Strings*.

Tips:

> **Tip:** You can use the `settype` transform 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 `setttype` transforms as late as possible in your recipes.

- When a column is set to a data type, all values in the column are validated against the new type, which might change the number of mismatched values. Some cleanup might be required. Some operations might cause the data type to be re-validated automatically.
- It might be easier to set type using the column's drop-down. Selections of data type from the column drop-down are turned into recipe steps using the `settype` transform.
- If you encounter a significant number of mismatches after you change the data type, you might find it helpful to change or revert the type to String. All data can be interpreted as a String or a list of string values. The transforms and functions for manipulating String data might be easier to use to clean up mismatched data before changing the data type to the preferred one.
- Row values that do not match the new data type might be turned to null values during job execution.

You can also specify any custom data types that have been defined.

Basic Usage

Single-column example:

```
settype col: Score type: 'Integer'
```

Output: Changes the data type for the `Score` column to Integer.

Multi-column example:

```
settype col: Score,studentId type: 'Integer'
```

Output: Changes the data type for the `Score` and `studentId` columns to Integer.

Syntax and Parameters

```
settype col:col1,col2 type:'string_literal'
```

Token	Required?	Data Type	Description
settype	Y	transform	Name of the transform
col	Y	string	Comma-separated list of columns to which to apply the specified type.
type	Y	string	String literal identifying the data type to apply to the column(s). See *Valid Data Type Strings*.

For more information on syntax standards, see *Language Documentation Syntax Notes*.

col

Identifies the column(s) to which to apply the transform. You can specify one or more columns.

Usage Notes:

Required?	Data Type
Yes	Comma-separated strings (column name or names)

type

Defines the data type that is to be applied to the transform. Type is defined as a String literal. For a list of valid strings, see *Valid Data Type Strings*.

```
settype col: zips type:'Zipcode'
```

Output: Changes the data type of the `zips` column to Zip Code data type. All values are validated as U.S. Zip code.

Usage Notes:

Required?	Data Type
Yes	String value

Examples

> **Tip:** For additional examples, see *Common Tasks*.

Example - Simple settype with date values

Source:

Here is a list of activities listed by date. Note the variation in date values, including what is clearly an invalid date. Here is the source data:

```
myDate, myAction
4/4/2016,Woke up at 6:30
4-4-2016,Got ready
9-9-9999,Drove kids to school
4-4-2016, Commuted to work
```

Transform:

When this data is imported into the Transformer page, there are couple of immediate issues: no column headings and blank rows at the bottom. These two transforms fix that:

```
header
```

```
delete row: ISMISSING([myDate])
```

For the invalid date, you can infer from the rows around it that it should be from the same date. You can make the following change to fix it:

```
replace col: myDate on: `9-9-9999` with: '4-4-2016' global: true
```

Now that the dates look fairly consistent, you can set the data type of the column to a matching Datetime format:

```
settype col: myDate type: 'Datetime','mm-dd-yy','mm*dd*yyyy'
```

Note the syntax above for specifying Datetime types. In addition to the `Datetime` keyword, you must specify the format type, followed by the variation of that format.

> **Tip:** A set of supported formats and variations for Datetime are available through the column data type selector. When you select your desired Datetime format, the `setttype` transform is added to your recipe.

Results:

myDate	myAction
4/4/2016	Woke up at 6:30
4-4-2016	Got ready
4-4-2016	Drove kids to school
4-4-2016	Commuted to work

Example - Use merge and settype to clean up numeric data that should be treated as other data types

This example illustrates how to clean up data that has been interpreted as numeric in nature, when it is actually String or a structured string type, such as Gender. This example uses:

- `settype` - defines the data type for a column or columns. See *Settype Transform*.
- `merge` - merges two String type columns together. See *Merge Transform*.

Source:

The following example contains customer ID and Zip code information in two columns. When this data is loaded into the Transformer page, it is initially interpreted as numeric, since it contains all numerals.

The four-digit `ZipCode` values should have five digits, with a `0` in front.

CustId	ZipCode
4020123	1234
2012121	94105
3212012	94101
1301212	2020

Transform:

CustId column: This column needs to be retyped as String values. You can set the column data type to String through the column drop-down, which is rendered as the following transform:

```
settype col:CustId type:'String'
```

While the column is now of String type, future transforms might cause it to be re-inferred as Integer values. To protect against this possibility, you might want to add a marker at the front of the string. This marker should be removed prior to execution.

The basic method is to create a new column containing the customer ID marker (`C`) and then merge this column and the existing `CustId` column together. It's useful to add such an indicator to the front in case the customer identifier is a numeric value that could be confused with other numeric values. Also, this merge step forces the value to be interpreted as a String value, which is more appropriate for an identifier.

```
merge col:'C', CustId
```

You can now drop the `CustId` columns and rename the new column as `CustId`.

ZipCode column: This column needs to be converted to valid Zip Code values. For ease of use, this column should be of type String:

```
settype col:ZipCode type:'Zipcode'
```

The transform below changes the value in the `ZipCode` column if the length of the value is four in any row. The new value is the original value prepended with the numeral `0`:

```
set col: ZipCode value: if(len($col) == 4, merge(['0',$col]), $col)
```

This column might now be re-typed as Zipcode type.

Results:

CustId	ZipCode
C4020123	01234
C2012121	94105
C3212012	94101
C1301212	02020

Remember to remove the `C` marker from the `CustId` column. Select the `C` value in the `CustId` column and choose the `replace` transform. You might need to re-type the cleaned data as String data.