Ternary Operators

Ternary operators allow you to build if/then/else conditional logic within your transforms. Please use the `IF` function instead.

> **NOTE:** Ternary operators have been superseded by the `IF` function. See *IF Function*.

In the following, if the `test expression` evaluates to `true`, the `true_expression` is executed. Otherwise, the `false_expression` is executed.

```
(test_expression) ? (true_expression) : (false_expression)
```

All of these expressions can be constants (strings, integers, or any other supported literal value) or sophisticated elements of logical, although the test expression must evaluate to a Boolean value.

> **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.

Usage

Example data:

X	Y
true	true
true	false
false	true
false	false

Transforms:

```
derive type:single value:(X == Y) ? 'yes' : 'no' as: 'equals'
```

Results:

Your output looks like the following:

X	Y	equals
true	true	yes
true	false	no
false	true	no
false	false	yes

Examples

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

Copyright © 2018 Trifacta Inc.

Example - Stock Quotes

You have a set of stock prices that you want to analyze. Based on a set of rules, you want to determine any buy, sell, or hold action to take.

Source:

Ticket	Qty	BuyPrice	CurrentPrice
GOOG	10	705.25	674.5
FB	100	84.00	101.125
AAPL	50	125.25	97.375
MSFT	100	38.875	45.25

Transform:

You can perform evaluations of this data using ternary operators to determine if you want to take action.

NOTE: In a larger dataset, you might maintain your buy, sell, and hold evaluations for each stock in a separate dataset that you join to the source dataset before performing comparisons between column values. See *Join Panel*.

To assist in evaluation, you might first want to create columns that contain the cost (`Basis`) and the current value (`CurrentValue`) for each stock:

```
derive type:single value:(Qty * BuyPrice) as:'Basis'
```

```
derive type:single value:(Qty * CurrentPrice) as:'CurrentValue'
```

Now, you can build some rules based on the spread between `Basis` and `CurrentValue`.

The most important action is determining if it is time to sell. The following rule writes a `sell` notification if the current value is $1000 or more than the cost. Otherwise, no value is written to the action column.

```
derive type:single value:(CurrentValue - 1000 > Basis) ? 'sell' : '' as:'action'
```

But what about buying more? The following transform is an edit to the previous one. In this new version, the sell test is performed, and if writes a `buy` action if the `CurrentPrice` is within 10% of the `BuyPrice`.

This second evaluation is performed after the first one, as it replaces the else clause, which did nothing in the previous version. In the Recipe panel, click the previous transform and edit it, replacing it with the new version:

```
derive type:single value: ((CurrentValue - 1000) > Basis) ? 'sell' : ((abs(CurrentValue -
Basis) <= (Basis * 0.1)) ? 'buy' : 'hold') as: 'action'
```

If neither test evaluates to `true`, the written action is `hold`.

You might want to format some of your columns using dollar formatting, as in the following:

NOTE: The following formatting inserts a dollar sign ($) in front of the value, which changes the data type to String.

```
set col:BuyPrice value:NUMFORMAT(BuyPrice, '$ ##,###.00')
```

Copyright © 2018 Trifacta Inc.

Results:

After moving your columns, your dataset should look like the following, if you completed the number formatting steps:

Ticket	Qty	BuyPrice	CurrentPrice	Basis	CurrentValue	action
GOOG	10	705.25	$ 674.50	$ 7,052.50	$ 6,745.00	buy
FB	100	84.00	$ 101.13	$ 8,400.00	$ 10,112.50	sell
AAPL	50	125.25	$ 97.38	$ 6,262.50	$ 4,868.75	hold
MSFT	100	38.88	$ 45.25	$ 3,887.50	$ 4,525.00	hold

Copyright © 2018 Trifacta Inc.