ISMISMATCHED Function

Contents:

Tests whether a set of values is not valid for a specified data type.

- For a tested value, this function returns `true` or `false` .
- Inputs can be literal values or column references.

You can define a conditional test in a single step for valid values. See *IFMISMATCHED Function*.

> **NOTE:** This function is similar to the `ISVALID` function, which tests for validity against a specified data type. However, unlike the `ISVALID` function, the `ISMISMATCHED` function also matches against missing values. See *VALID Function*.

Basic Usage

Column reference example:

```
ismismatched(Qty, 'Integer') || (Qty < 0)
```

Output: Returns `true` when the value in the `Qty` column does not contain a valid Integer and the value is less than zero.

Numeric literal example:

```
ismismatched('ZZ', 'State')
```

Output: Returns `true`, since the value `ZZ` is not a valid U.S. State code.

Syntax and Arguments

```
ismismatched(column_string,datatype_literal)
```

Argument	Required?	Data Type	Description
column_string	Y	string	Name of column or string literal to be applied to the function
datatype_literal	Y	string	String literal that identifies the data type against which to validate the source values

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

column_string

Name of the column or string literal to be evaluated for mismatches against the specified type.

- Missing literals or column values generate missing string results.
  - Constants must be quoted (`'Hello, World'`).
- Multiple columns and wildcards are not supported.

Usage Notes:

Required?	Data Type	Example Value
Yes	String literal or column reference	`myColumn`

datatype_literal

Literal value for data type to which to validate the source column or string.

- Column references are not supported.

Usage Notes:

Required?	Data Type	Example Value
Yes	String literal	`'Integer'`

Valid data type strings:

When referencing a data type within a transform, you can use the following strings to identify each type:

> **NOTE:** In Wrangle transforms, these values are case-sensitive.

> **NOTE:** When specifying a data type by name, you must use the String value listed below. The Data Type value is the display name for the type.

Data Type	String
String	`'String'`
Integer	`'Integer'`
Decimal	`'Float'`
Boolean	`'Bool'`
Social Security Number	`'SSN'`
Phone Number	`'Phone'`
Email Address	`'Emailaddress'`
Credit Card	`'Creditcard'`
Gender	`'Gender'`
Object	`'Map'`

Array	`'Array'`
IP Address	`'Ipaddress'`
URL	`'Url'`
HTTP Code	`'Httpcodes'`
Zip Code	`'Zipcode'`
State	`'State'`
Date / Time	`'Datetime'`

For custom types, you should reference the name of the type in the string value. For more information, see *Create Custom Data Types*.

Examples

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

Example - Type check functions

This example illustrates how various type checking functions can be applied to your data.

- `ISVALID` - Returns `true` if the input matches the specified data type. See *VALID Function*.
- `ISMISMATCHED` - Returns `true` if the input does not match the specified data type. See *ISMISMATCHED Function*.
- `ISMISSING` - Returns `true` if the input value is missing. See *ISMISSING Function*.
- `ISNULL` - Returns `true` if the input value is null. See *ISNULL Function*.
- `NULL` - Generates a null value. See *NULL Function*.

Source:

Some source values that should match the State and Integer data types:

State	Qty
CA	10
OR	-10
WA	2.5
ZZ	15
ID	
	4

Transformation:

Invalid State values: You can test for invalid values for State using the following:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`ISMISMATCHED (State, 'State')`

The above transform flags rows 4 and 6 as mismatched.

> **NOTE:** A missing value is not valid for a type, including String type.

Invalid Integer values: You can test for valid matches for Qty using the following:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`(ISVALID (Qty, 'Integer') && (Qty > 0))`
Parameter: New column name	`'valid_Qty'`

The above transform flags as valid all rows where the `Qty` column is a valid integer that is greater than zero.

Missing values: The following transform tests for the presence of missing values in either column:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`(ISMISSING(State) || ISMISSING(Qty))`
Parameter: New column name	`'missing_State_Qty'`

After re-organizing the columns using the `move` transform, the dataset should now look like the following:

State	Qty	mismatched_State	valid_Qty	missing_State_Qty
CA	10	false	true	false
OR	-10	false	false	false
WA	2.5	false	false	false
ZZ	15	true	true	false
ID		false	false	true
	4	false	true	true

Since the data does not contain null values, the following transform generates null values based on the preceding criteria:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`((mismatched_State == 'true') || (valid_Qty == 'false') || (missing_State_Qty == 'true')) ? NULL() : 'ok'`
Parameter: New column name	`'status'`

You can then use the `ISNULL` check to remove the rows that fail the above test:

Transformation Name	`Filter rows`
Parameter: Condition	`Custom formula`
Parameter: Type of formula	`Custom single`
Parameter: Condition	`ISNULL('status')`
Parameter: Action	`Delete matching rows`

Results:

Based on the above tests, the output dataset contains one row:

State	Qty	mismatched_State	valid_Qty	missing_State_Qty	status
CA	10	false	true	false	ok