

# IFVALID Function

## Contents:

- *Basic Usage*
- *Syntax and Arguments*
  - *source\_value*
  - *datatype\_literal*
  - *output\_value*
- *Examples*
  - *Example - IF\* functions for data type validation*

---

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.

The `VALID` function simply tests if a value is valid. See *VALID Function*.

## Basic Usage

```
derive type:single value:IFVALID(myZip,'ZipCode', 'ok') as:'myZipCheck'
```

**Output:** Generates a new column called, `myZipCheck`, which contains the value `ok` if the value in `myZip` matches the `ZipCode` data type.

## Data type with formatting options:

For data types with formatting options, such as `Datetime`, you can specify the format using an array, as in the following:

```
set col: validDate value: IFVALID(myDate, ['Datetime','mm-dd-yy','mm*dd*yy'], 'true')
```

**Output:** Sets the values in the `validDate` column to `true`, if the corresponding value in the `myDate` column is a valid `Datetime` value in `yy-mm-dd` or `yy*mm*dd` format.

## Syntax and Arguments

```
derive type:single value:IFVALID(column_string, data_type_literal, computed_value)
```

Argument	Required?	Data Type	Description
<code>source_value</code>	Y	string	Name of column, string literal or function to be tested
<code>datatype_literal</code>	Y	string	String literal that identifies the data type against which to validate the source values
<code>output_value</code>	y	string	String literal value to write

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

### **source\_value**

Name of the column, string literal, or function to be tested for data type matches.

- Missing literals or column values generate missing string results.
- Multiple columns and wildcards are not supported.

**Usage Notes:**

Required?	Data Type	Example Value
Yes	String literal, column reference, or function	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:

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

**i 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*.

### output\_value

The output value to write if the tested value is valid for the specified data type.

### Usage Notes:

Required?	Data Type	Example Value
Yes	String or numeric literal	'Data type mismatch'

### Examples

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

### Example - IF\* functions for data type validation

This section provides simple examples for how to use the IF\* functions for data type validation. These functions include the following:

- **IFNULL** - For an input expression or value, this function returns the specified value if the input is a null value. See *IFNULL Function*.
- **IFMISSING** - Returns the specified value if the input value or expression is a missing value. See *IFMISSING Function*.
- **IFMISMATCHED** - Returns the specified value if the input value or expression is mismatched against the column's data type. See *IFMISMATCHED Function*.
- **IFVALID** - Returns the specified value if the input value or expression is valid against the column's data type. See *IFVALID Function*.

### Source:

The following simple table lists zip codes by customer identifier:

custId	custZip
C001	98123
C002	94105
C003	12415
C004	12451-2234
C005	12441-298
C006	
C007	
C008	1242

C009	1104
------	------

## Transform:

When the above is imported into the Transformer page, you notice the following:

- The `custZip` column is typed as Integer.
- There are two missing and two mismatched values in the `custZip` column.

First, you test for valid values in the `custZip` column. Using the `IFVALID` function, you can validate against any data type:

```
derive type:single value:IFVALID(custZip, 'Zipcode', 'ok') as:'status'
```

**Fix four-digit zips:** In the `status` column are instances of `ok` for the top four rows. You notice that the bottom two rows contain four-digit codes.

Since the `custZip` values were originally imported as Integer, any leading 0 values are deleted. In this case, you can add back the leading zero. Before the previous step, change the data type of `zip` to String and insert the following:

```
derive type:single value:IF(LEN(custZip)==4,'0','') as:'FourDigitZip'
```

```
derive type:single value: merge([FourDigitZip,custZip]) as:'custZip2'
```

```
set col:zip value:custZip2
```

```
drop col:FourDigitZip,custZip2
```

Now, when you click the last recipe step, you should see that two more rows in `status` are listed as `Ok`.

For the zip code with the three-digit extension, you can simply remove that extension to make it valid. Click the step above the last one. In the data grid, highlight the value. Click the Replace suggestion card. Select the option that uses the following for the matching pattern:

```
'-{\digit}{3}{end}'
```

The above means that all three-digit extensions are deleted from the zip. You can do the same for any two- and one-digit extensions, although there are none in this sample.

**Missing and null values:** Now, you need to address how to handle missing and null values. The `IFMISSING` tests for both missing and null values, while the `IFNULL` tests just for null values. In this example, you want to delete null values, which could mean that the data for that row is malformed and to write a status of `missing` for missing values.

Click above the last line in the recipe to insert the following:

```
set col:custZip value:IFNULL(custZip, 'xxxxx')
```

```
set col:custZip value:IFMISSING(custZip, '00000')
```

Now, when you click the last line of the recipe, only the null value is listed as having a status other than `ok`. You can use the following to remove this row and all like it:

```
delete row:(status == 'xxxxx')
```

**Results:**

custId	custZip	status
C001	98123	ok
C002	94105	ok
C003	12415	ok
C004	12451-2234	ok
C005	12441-298	ok
C006	00000	ok
C008	1242	ok
C009	1104	ok

As an exercise, you might repeat the above steps starting with the IFMISMATCHED function determining the value in the status column:

```
derive type:single value:IFMISMATCHED(custZip, 'Zipcode', 'mismatched') as:'status'
```