PARSEDATE Function

Evaluates an input against the default input formats or (if specified) an array of Datetime format strings in their listed order. If the input matches one of the formats, the function outputs a Datetime value.

- Inputs can be of any type.
  - If the input is not a Datetime value and does match one of the specified formats, the output is in the following format: `yyyy-MM-dd HH:mm:ss`.
  - If the input is a Datetime value and does match, the output is in the input's Datetime format.

After you have converted your strings values to dates, if a sufficient percentage of input strings from a column are successfully converted to one of the matching formats, the column may be retyped as Datetime.

- Trifacta® Wrangler Enterprise supports a wide variety of formats for Datetime fields. For more information on supported date formats, see *Datetime Data Type*.
- You can explore the available Datetime formats through the Transformer page. From a column's type drop-down, select **Date/Time** . Then, select the formatting category. From the displayed drop-down, you can select a specific format. When this transform step is added to your recipe, you can edit it to see how the format is specified in Wrangle .
- You can then use the DATEFORMAT function to convert the output values to your preferred Datetime format. See *DATEFORMAT Function*.

Wrangle vs. SQL: This function is part of Wrangle , a proprietary data transformation language. Wrangle is not SQL. For more information, see *Wrangle Language*.

Basic Usage

```
parsedate(strDate, ['yyyy-MM-dd','yyyy-MM','yyyy/MM','yyyy-MM-dd'])
```

Output: Returns a value structured in `yyyy-MM-dd HH:mm:ss` format if the input value in `strDate` matches any of default formats, which are the following:

```
'yyyy-MM-dd HH:mm:ss'
'yyyy/MM/dd HH:mm:ss'
'yyyy-MM-dd'
'yyyy/MM/dd'
```

```
parsedate(strDate, ['yyyy-MM','yyyy/MM',])
```

Output: Returns a value structured in `yyyy-MM-dd HH:mm:ss` format if the input value in `strDate` matches any of the listed formats for dates.

Syntax and Arguments

```
parsedate(date_col, date_formats_array)
```

Argument	Required?	Data Type	Description
date_col	Y	any	Literal, name of a column, or a function returning values to match
date_formats_array	N	string	(optional) An array of date format strings that are used to match against input values.

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

date_col

Literal, column name, or function returning values that are to be evaluated for conversion to Datetime values.

- Inputs values can be of any type.
- Missing values for this function in the source data result in null values in the output.
- Multiple columns and wildcards are not supported.

Usage Notes:

Required?	Data Type	Example Value
Yes	any	`'February 24, 2019'`

date_formats_array

Array of String values of the date formats to evaluate the inputs.

- When a non-Datetime input value matches one of the date formats in the array, the output is the input value converted to the following format:

```
yyyy-MM-dd HH:mm:ss
```

- Datetime inputs are outputted in their source format.

Trifacta Wrangler Enterprise supports Java formatting strings, with some exceptions.

> **NOTE:** If the platform cannot recognize the date format string, a null value is written as the output.

For more information on supported date formats, see *Datetime Data Type*.

Usage Notes:

Required?	Data Type	Example Value
No	Array of Strings	`['yyyy-MM','yyyy/MM']`

Examples

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

Example - formatting date values

This example illustrates several ways of wrangling heterogeneous date values, including the use of the `DATEFORMAT` function.

Source:

Your dataset includes the following messy date values:

MyDateStrings
2/1/00 14:20
4/5/10 11:25
6/7/99 22:00
13/7/1999 22:00
12-20-1894 15:45:00
08-12-1956 22:01:04

Transformation:

To enable easier comparison in the data grid, you choose to create a new column with the parsed values. From the above, you identify two date formats:

```
'MM/dd/yy hh:mm'
'MM/dd/YYYY hh:mm:ss'
```

> **NOTE:** Since only one of the above formats matches the default formats, you must specify both in the transformation to perform the proper evalation.

You create the following transformation to parse them:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`parsedate(myDateStrings, ['MM/dd/yy hh:mm','MM/dd/YYYY hh:mm:ss'])`
Parameter: New column name	`'myParsedDates'`

When the above step is added to the recipe, the output is as follows:

MyDateStrings	myParsedDates
2/1/00 14:20	2000-02-01 14:20:00
4/5/10 11:25	2010-04-05 11:25:00
6/7/99 22:00	1999-06-07 22:00:00
13/7/1999 22:00	13/7/1999 22:00
12-20-1894 15:45:00	1894-12-20 15:45:00
08-12-1956 22:01:04	1956-08-12 22:01:04

The output `myParsedDates` column is retyped as a Datetime column with one mismatched value: `3/7/1999 22:00`.

This value does not match any of our date formats specified in the array. The solution is to modify the recipe step to include the appropriate format as part of the array:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`parsedate(myDateStrings, ['MM/dd/yy hh:mm','MM/dd/YYYY hh:mm:ss','dd/MM/yyyy hh:mm'])`
Parameter: New column name	`'myParsedDates'`

Results:

MyDateStrings	myParsedDates
2/1/00 14:20	2000-02-01 14:20:00
4/5/10 11:25	2010-04-05 11:25:00
6/7/99 22:00	1999-06-07 22:00:00
13/7/1999 22:00	1999-07-13 22:00:00
12-20-1894 15:45:00	1894-12-20 15:45:00
08-12-1956 22:01:04	1956-08-12 22:01:04

For more information on supported date formats, see *Datetime Data Type*.

Example - type parsing functions

This example shows how to use the following parsing functions for evaluating input against the function-specific data type:

- `PARSEBOOL` - If the input String value is a valid Boolean value, the value is returned as a Boolean data type value. See *PARSEBOOL Function*.
- `PARSEDATE` - If the input String value is valid against the specified or default Datetime formats, the value is returned as a Datetime value. See *PARSEDATE Function*.
- `PARSEFLOAT` - If the input String value is a valid Float (Decimal) value, the value is returned as a Decimal data type value. See *PARSEFLOAT Function*.
- `PARSEINT` - If the input String value is a valid Integer value, the value is returned as an Integer data type value. See *PARSEINT Function*.

Source:

The following table contains data on a series of races.

raceId	disqualified	date	racerId	time_sc
1	FALSE	2/1/20	1	24.22
2	f	2/8/20	1	25
3	no	2/8/20	1	24.11
4	n	1-Feb-20	2	26.1
5	TRUE	8-Feb-20	2.2	-25.22
6	t	2/8/2020 10:16:00 AM	2	25.44
7	yes	2/1/20	3	24
8	y	2/8/20	33	29.22

9	0	2/8/20	3	24.78
10	1	1-Feb-20	4	26.2.1
11	FALSE	8-Feb-20		28.22 sec
12	FALSE	2/8/2020 10:16:00 AM	4	27.11

As you can see, this dataset has variation in values (FALSE, f, no, n) and problems with the data.

Transformation:

When the data is first imported, it may be properly typed for each column. To use the parsing functions, these columns should be converted to String data type:

Transformation Name	`Change column data type`
Parameter: Columns	`disqualified,date,racerId,time_sc`
Parameter: New type	`String`

Now, you can parse individual columns.

disqualified column:

Transformation Name	`Edit column with formula`
Parameter: Columns	`disqualified`
Parameter: Formula	`PARSEBOOL($col)`

racerId column:

Transformation Name	`Edit column with formula`
Parameter: Columns	`racerId`
Parameter: Formula	`PARSEINT($col)`

time_sc column:

Transformation Name	`Edit column with formula`
Parameter: Columns	`time_sc`
Parameter: Formula	`PARSEFLOAT($col)`

date column:

For the date column, the PARSEDATE function supports a default set of Datetime formats. Since some of the listed formats are different from these defaults, you must specify all of the formats. These formats are specified as an array of string values as the second argument of the function:

> **Tip:** For the PARSEDATE function, it's useful to use the Preview to verify that all of the dates in the column are represented in the array of output formats. You can see the available output formats through the data type menu at the top of a column. See *Choose Datetime Format Dialog*.

Transformation Name	`Edit column with formula`
Parameter: Columns	`date`
Parameter: Formula	`PARSEDATE($col, ['yyyy-MM-dd','yyyy\/MM\/dd','M\/d\/yyy hh:mm','MMMM d, yyyy','MMM d, yyyy'])`

After all of the date values have been standardized to the output format of the PARSEDATE function, you may choose to remove the time element of the values:

Transformation Name	`Replace text or pattern`
Parameter: Column	`date`
Parameter: Find	`` ` {digit}{2}:{digit}{2}:{digit}{2}{end}` ``
Parameter: Replace with	`''`

Results:

After executing the above steps, the data appears as follows. Notes on each column's output are below the table.

raceId	disqualified	date	racerId	time_sc
1	false	2020-02-01	1	24.22
2	false	2020-02-08	1	25
3	false	2020-02-08	1	24.11
4	false	2020-02-01	2	26.1
5	true	2020-02-08	*null*	-25.22
6	true	2020-02-08	2	25.44
7	true	2020-02-01	3	24
8	true	2020-02-08	33	29.22
9	false	2020-02-08	3	24.78
10	true	2020-02-01	4	*null*
11	false	2020-02-08	*null*	*null*
12	false	2020-02-08	4	27.11

disqualified column:

- The PARSEBOOL function normalizes all valid Boolean values to either `false` or `true`.

racerId column:

- The PARSEINT function writes invalid values as null values.
- The function writes empty values as null values.
- The value 33 remains, since it is a valid Integer. This value should be fixed manually.

time_sc:

- The PARSEFLOAT function writes the source value `25.00` as `25` in output.
- The source value `-25.22` remains. However, since this is time-based data, it needs to be fixed.
- Invalid values are written as nulls.

date column:

- All values are written in the standardized format: `yyyy-MM-dd HH:mm:ss`. Time data has been stripped.