PARSEARRAY Function

Evaluates a String input against the Array datatype. If the input matches, the function outputs an Array value. Input can be a literal, a column of values, or a function returning String values.

After you have converted your strings to arrays, if a sufficient percentage of input strings from a column are successfully converted to the other data type, the column may be retyped.

Tip: If the column is not automatically retyped as a result of this function, you can manually set the type to Array in a subsequent recipe step.

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

```
parsearray(strInput)
```

Output: Returns the Array data type value for `strInput` String values.

Syntax and Arguments

```
parsearray(str_input)
```

Argument	Required?	Data Type	Description
str_input	Y	String	Literal, name of a column, or a function returning String values to match

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

str_input

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

- 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	String	`'[1,2,3]'`

Examples

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

Example - parsing strings as an array

Source:

The following table represents raw imported CSV data:

setId	itemsA	itemsB
s01	"1,2,3"	4
s02	"2,3,4"	4
s03	"3,4,5"	4
s04	"4,5,6"	4
s05	"5,6,7"	4
s06	"6,7,8"	4

In the above table, you can see that the two items columns are interpreted differently. In the following steps, you can see how you can parse the data as array values before producing a new column intersecting the two arrays.

Transformation:

Create a new column to store the array version of `itemsA`:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`itemsA`
Parameter: New column name	`arrA`

Remove the quotes from the column:

Transformation Name	`Replace text or pattern`
Parameter: Column	`arrA`
Parameter: Find	`` `"` ``
Parameter: Replace with	`''`
Parameter: Match all occurrences	`true`

Now create the array by merging the array text value with square brackets and then using the PARSEARRAY function to evaluate the merged value as an array:

Transformation Name	`Edit column with formula`
Parameter: Columns	`arrA`

Parameter: Formula	`parsearray(merge(['[',arrA,']']))`

You can create the second array column using a similar construction in a new column:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`parsearray(merge(['[',itemsB,']']))`
Parameter: New column name	`arrB`

Since both columns have been parsed as array values, you can use the ARRAYINTERSECT function to find the common values:

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`arrayintersect([arrA,arrB])`
Parameter: New column name	`arrIntersectAB`

Results:

setId	itemsA	itemsB	arrA	arrB	arrIntersectAB
s01	"1,2,3"	4	[1,2,3]	[4]	[]
s02	"2,3,4"	4	[2,3,4]	[4]	[4]
s03	"3,4,5"	4	[3,4,5]	[4]	[4]
s04	"4,5,6"	4	[4,5,6]	[4]	[4]
s05	"5,6,7"	4	[5,6,7]	[4]	[]
s06	"6,7,8"	4	[6,7,8]	[4]	[]