ROLLINGMIN Function

Contents:

Computes the rolling minimum of values forward or backward of the current row within the specified column. Input s can be Integer, Decimal, or Datetime.

- If an input value is missing or null, it is not factored in the computation. For example, for the first row in the dataset, the rolling minimum of previous values is undefined.
- The row from which to extract a value is determined by the order in which the rows are organized based on the `order` parameter.
- If you are working on a randomly generated sample of your dataset, the values that you see for this function might not correspond to the values that are generated on the full dataset during job execution.
- The function takes a column name and two optional integer parameters that determine the window backward and forward of the current row.
  - The default integer parameter values are `-1` and `0`, which computes the rolling function from the current row back to the first row of the dataset.
- This function works with the Window transform. See *Window Transform*.

For more information on a non-rolling version of this function, see *MIN 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

Column example:

```
rollingmin(myCol)
```

Output: Returns the rolling minimum of all values in the `myCol` column.

Rows before example:

```
rollingmin(myNumber, 3)
```

Output: Returns the rolling minimum of the current row and the three previous row values in the `myNumber` column.

Rows before and after example:

```
rollingmin(myNumber, 3, 2)
```

Output: Returns the rolling minimum of the three previous row values, the current row value, and the two rows after the current one in the `myNumber` column.

Syntax and Arguments

```
rollingmin(col_ref, rowsBefore_integer, rowsAfter_integer) order: order_col [group:
group_col]
```

Argument	Required?	Data Type	Description
col_ref	Y	string	Name of column whose values are applied to the function
rowsBefore_integer	N	integer	Number of rows before the current one to include in the computation
rowsAfter_integer	N	integer	Number of rows after the current one to include in the computation

For more information on the `order` and `group` parameters, see *Window Transform*.

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

col_ref

Name of the column whose values are used to compute the function. Inputs must be Integer, Decimal, or Datetime values.

> **NOTE:** If the input is in Datetime type, the output is in unixtime format. You can wrap these outputs in the DATEFORMAT function to generate the results in the appropriate Datetime format. See *DATEFORMAT Function*.

- Multiple columns and wildcards are not supported.

Usage Notes:

Required?	Data Type	Example Value
Yes	String (column reference to Integer or Decimal values)	`myColumn`

rowsBefore_integer, rowsAfter_integer

Integers representing the number of rows before or after the current one from which to compute the rolling function, including the current row. For example, if the first value is `5`, the current row and the five rows before it are used in the computation. Negative values for `k` compute the rolling average from rows preceding the current one.

- `rowBefore=0` generates the current row value only.
- `rowBefore=-1` uses all rows preceding the current one.
- If `rowsAfter` is not specified, then the value `0` is applied.
- If a `group` parameter is applied, then these parameter values should be no more than the maximum number of rows in the groups.

Usage Notes:

Required?	Data Type	Example Value
No	Integer	4

Examples

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

Example - Rolling computations for racing splits

This example describes how to use the rolling computational functions:

- `ROLLINGAVERAGE` - computes a rolling average from a window of rows before and after the current row. See *ROLLINGAVERAGE Function*.
- `ROLLINGMIN` - computes a rolling minimum from a window of rows. See *ROLLINGMIN Function*.
- `ROLLINGMAX` - computes a rolling maximum from a window of rows. See *ROLLINGMAX Function*.
- `ROLLINGSTDEV` - computes a rolling standard deviation from a window of rows. See *ROLLINGSTDEV Function*.
- `ROLLINGVAR` - computes a rolling variance from a window of rows. See *ROLLINGVAR Function*.
- `ROLLINGSTDEVSAMP` - computes a rolling standard deviation from a window of rows using the sample method of statistical calculation. See *ROLLINGSTDEVSAMP Function*.
- `ROLLINGVARSAMP` - computes a rolling variance from a window of rows using the sample method of statistical calculation. See *ROLLINGVARSAMP Function*.

Source:

In this example, the following data comes from times recorded at regular intervals during a three-lap race around a track. The source data is in cumulative time in seconds (`time_sc`). You can use ROLLING and other windowing functions to break down the data into more meaningful metrics.

lap	quarter	time_sc
1	0	0.000
1	1	19.554
1	2	39.785
1	3	60.021
2	0	80.950
2	1	101.785
2	2	121.005
2	3	141.185
3	0	162.008
3	1	181.887
3	2	200.945
3	3	220.856

Transformation:

Primary key: Since the quarter information repeats every lap, there is no unique identifier for each row. The following steps create this identifier:

Transformation Name	`Change column data type`

Parameter: Columns	`lap,quarter`
Parameter: New type	`String`

Transformation Name	`New formula`
Parameter: Formula type	`Single row formula`
Parameter: Formula	`MERGE(['l',lap,'q',quarter])`
Parameter: New column name	`'splitId'`

Get split times: Use the following transform to break down the splits for each quarter of the race:

Transformation Name	`New formula`
Parameter: Formula type	`Multiple row formula`
Parameter: Formula	`ROUND(time_sc - PREV(time_sc, 1), 3)`
Parameter: Order rows by	`splitId`
Parameter: New column name	`'split_time_sc'`

Compute rolling computations: You can use the following types of computations to provide rolling metrics on the current and three previous splits:

Transformation Name	`New formula`
Parameter: Formula type	`Multiple row formula`
Parameter: Formula	`ROLLINGAVERAGE(split_time_sc, 3)`
Parameter: Order rows by	`splitId`
Parameter: New column name	`'ravg'`

Transformation Name	`New formula`
Parameter: Formula type	`Multiple row formula`
Parameter: Formula	`ROLLINGMAX(split_time_sc, 3)`
Parameter: Order rows by	`splitId`
Parameter: New column name	`'rmax'`

Transformation Name	`New formula`
Parameter: Formula type	`Multiple row formula`
Parameter: Formula	`ROLLINGMIN(split_time_sc, 3)`
Parameter: Order rows by	`splitId`

Parameter: New column name	`'rmin'`

Transformation Name	`New formula`
Parameter: Formula type	`Multiple row formula`
Parameter: Formula	`ROUND(ROLLINGSTDEV(split_time_sc, 3), 3)`
Parameter: Order rows by	`splitId`
Parameter: New column name	`'rstdev'`

Transformation Name	`New formula`
Parameter: Formula type	`Multiple row formula`
Parameter: Formula	`ROUND(ROLLINGVAR(split_time_sc, 3), 3)`
Parameter: Order rows by	`splitId`
Parameter: New column name	`'rvar'`

Compute rolling computations using sample method: These metrics compute the rolling STDEV and VAR on the current and three previous splits using the sample method:

Transformation Name	`New formula`
Parameter: Formula type	`Multiple row formula`
Parameter: Formula	`ROUND(ROLLINGSTDEVSAMP(split_time_sc, 3), 3)`
Parameter: Order rows by	`splitId`
Parameter: New column name	`'rstdev_samp'`

Transformation Name	`New formula`
Parameter: Formula type	`Multiple row formula`
Parameter: Formula	`ROUND(ROLLINGVARSAMP(split_time_sc, 3), 3)`
Parameter: Order rows by	`splitId`
Parameter: New column name	`'rvar_samp'`

Results:

When the above transforms have been completed, the results look like the following:

lap	quarter	splitId	time_sc	split_time_sc	rvar_samp	rstdev_samp	rvar	rstdev	rmin	rmax	ravg
1	0	l1q0	0								
1	1	l1q1	20.096	20.096			0	0	20.096	20.096	20.096
1	2	l1q2	40.53	20.434	0.229	0.479	0.029	0.169	20.096	20.434	20.265

1	3	l1q3	61.031	20.501	0.154	0.392	0.031	0.177	20.096	20.501	20.344
2	0	l2q0	81.087	20.056	0.315	0.561	0.039	0.198	20.056	20.501	20.272
2	1	l2q1	101.383	20.296	0.142	0.376	0.029	0.17	20.056	20.501	20.322
2	2	l2q2	122.092	20.709	0.617	0.786	0.059	0.242	20.056	20.709	20.39
2	3	l2q3	141.886	19.794	0.621	0.788	0.113	0.337	19.794	20.709	20.214
3	0	l3q0	162.581	20.695	0.579	0.761	0.139	0.373	19.794	20.709	20.373
3	1	l3q1	183.018	20.437	0.443	0.666	0.138	0.371	19.794	20.709	20.409
3	2	l3q2	203.493	20.475	0.537	0.733	0.113	0.336	19.794	20.695	20.35
3	3	l3q3	222.893	19.4	0.520	0.721	0.252	0.502	19.4	20.695	20.252

You can reduce the number of steps by applying a `window` transform such as the following:

Transformation Name	`Window`
Parameter: Formula1	`lap`
Parameter: Formula2	`rollingaverage(split_time_sc, 0, 3)`
Parameter: Formula3	`rollingmax(split_time_sc, 0, 3)`
Parameter: Formula4	`rollingmin(split_time_sc, 0, 3)`
Parameter: Formula5	`round(rollingstdev(split_time_sc, 0, 3), 3)`
Parameter: Formula6	`round(rollingvar(split_time_sc, 0, 3), 3)`
Parameter: Formula7	`round(rollingstdevsamp(split_time_sc, 0, 3), 3)`
Parameter: Formula8	`round(rollingvarsamp(split_time_sc, 0, 3), 3)`
Parameter: Group by	`lap`
Parameter: Order by	`lap`

However, you must rename all of the generated `windowX` columns.