

## EXAMPLE - Rolling Functions 2

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*.

### 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

### Transform:

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

```
settype col: lap,quarter type: 'String'
```

```
derive type:single value: MERGE(['l',lap,'q',quarter]) as: 'splitId'
```

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

```
derive type:single value: ROUND(time_sc - PREV(time_sc, 1), 3) order: splitId as: '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:

```
derive type:single value: ROLLINGAVERAGE(split_time_sc, 3) order: splitId as: 'ravg'
```

```
derive type:single value: ROLLINGMAX(split_time_sc, 3) order: splitId as: 'rmax'
```

```
derive type:single value: ROLLINGMIN(split_time_sc, 3) order: splitId as: 'rmin'
```

```
derive type:single value: ROUND(ROLLINGSTDEV(split_time_sc, 3), 3) order: splitId as:  
'rstdev'
```

```
derive type:single value: ROUND(ROLLINGVAR(split_time_sc, 3), 3) order: splitId as: 'rvar'
```

## Results:

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

lap	quarter	splitId	time_sc	split_time_sc	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	0.029	0.169	20.096	20.434	20.265
1	3	l1q3	61.031	20.501	0.031	0.177	20.096	20.501	20.344
2	0	l2q0	81.087	20.056	0.039	0.198	20.056	20.501	20.272
2	1	l2q1	101.383	20.296	0.029	0.17	20.056	20.501	20.322
2	2	l2q2	122.092	20.709	0.059	0.242	20.056	20.709	20.39
2	3	l2q3	141.886	19.794	0.113	0.337	19.794	20.709	20.214
3	0	l3q0	162.581	20.695	0.139	0.373	19.794	20.709	20.373
3	1	l3q1	183.018	20.437	0.138	0.371	19.794	20.709	20.409
3	2	l3q2	203.493	20.475	0.113	0.336	19.794	20.695	20.35
3	3	l3q3	222.893	19.4	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:

```
window value: window1 = lap,rollingaverage(split_time_sc, 0, 3), rollingmax(split_time_sc,  
0, 3),rollingmin(split_time_sc, 0, 3),round(rollingstdev(split_time_sc, 0, 3), 3),round  
(rollingvar(split_time_sc, 0, 3), 3) group: lap order: lap
```

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