

User Manual of IoTDB-Quality

Author: Data Quality Group

Institute: School of Software, Tsinghua University

Date: August 18, 2021

Contents

1	Get	Started
	1.1	Introduction
	1.2	Comparison
	1.3	Q&A
2	Data	Profiling
	2.1	Distinct
	2.2	Histogram
	2.3	Integral
	2.4	Mad
	2.5	Median
	2.6	Mode
	2.7	Percentile
	2.8	Period
	2.9	QLB
	2.10	Resample
	2.11	Sample
	2.12	Segment
	2.13	Skew
	2.14	Spread
	2.15	Stddev
	2.16	TimeWeightedAvg
3	Data	Quality 27
-	3.1	Completeness
		Consistency
	3.3	Timeliness
	3.4	Validity
4	Data	Repairing 37
	4.1	Fill(TODO)
	4.2	TimestampRepair
	4.3	ValueRepair
5	Data	Matching 42
	5.1	Cov

	5.2	CrossCorrelation
	5.3	DTW(TODO)
	5.4	PatternSymmetric
	5.5	Pearson
	5.6	SelfCorrelation
	5.7	SeriesAlign(TODO)
	5.8	SeriesSimilarity(TODO)
	5.9	ValueAlign(TODO)
6	Ano	maly Detection 49
	6.1	ADWIN
	6.2	KSigma
	6.3	LOF
	6.4	Range
	6.5	TwoSidedFilter
7	Free	quency Domain 58
	7.1	Conv(TODO)
	7.2	Deconv(TODO)
	7.3	FFT
	7.4	Filter(TODO)
8	Seri	es Discovery 62
	8.1	ConsecutiveSequences
	8.2	ConsecutiveWindows
9	Con	nplex Event Processing 60
	9.1	AND(TODO)
	9.2	EventMatching(TODO)
	9.3	EventNameRepair(TODO)
	9.4	EventTag(TODO)
	9.5	EventTimeRepair(TODO)
	9.6	MissingEventRecovery(TODO)
	9.7	SEQ(TODO)

Chapter 1 Get Started

1.1 Introduction

1.1.1 What is IoTDB-Quality

Apache IoTDB (Internet of Things Database) is a data management system for time series data, which can provide users specific services, such as, data collection, storage and analysis.

For applications based on time series data, data quality is vital. **IoTDB-Quality** is IoTDB User Defined Functions (UDF) about data quality, including data profiling, data quality evalution and data repairing. It effectively meets the demand for data quality in the industrial field.

1.1.2 Quick Start

- 1. Download the JAR with all dependencies and the script of registering UDF.
- 2. Copy the JAR package to ext\udf under the directory of IoTDB system.
- 3. Run sbin\start-server.bat (for Windows) or sbin\start-server.sh (for Linux or MacOS) to start IoTDB server.
- 4. Copy the script to the directory of IoTDB system and run it to register UDF.

1.1.3 Contact

• Email: iotdb-quality@protonmail.com

1.2 Comparison

1.2.1 InfluxDB v2.0

InfluxDB is a popular time series database. InfluxQL is its query language, some of whose universal functions are related to data profiling. The comparison is shown below. *Native* means this function has been the native function of IoTDB and *Built-in UDF* means this function has been the built-in UDF of IoTDB.

Data profiling functions of IoTDB-Quality	Universal functions from InfluxQL
Native	COUNT()
Distinct	DISTINCT()
Integral	INTEGRAL()
Native	MEAN()
Median	MEDIAN()
Mode	MODE()
Spread	SPREAD()
Stddev	STDDEV()
Native	SUM()
Built-in UDF	BOTTOM()
Native	FIRST()
Native	LAST()
Native	MAX()
Native	MIN()
Percentile	PERCENTILE()
Sample	SAMPLE()
Built-in UDF	TOP()
Histogram	HISTOGRAM()
Mad	
Skew	SKEW()
TimeWeightedAVG	TIMEWEIGHTEDAVG()
SelfCorrelation	
CrossCorrelation	

Kapacitor offers UDF to realize user-defined anomaly detection. Python scripts can be applied to Kapacitor, and no native function for anomaly detection is offered in InfluxDB.

1.3 Q&A

1.3.1 Is the name of UDF case sensitive

The name of UDF is not case sensitive. Users can choose uppercase, lowercase or mixed case according to their own habits.

Chapter 2 Data Profiling

2.1 Distinct

2.1.1 Usage

This function returns all unique values in time series.

Name: DISTINCT

Input Series: Only support a single input series. The type is arbitrary. **Output Series:** Output a single series. The type is the same as the input.

Note:

- The timestamp of the output series is meaningless. The output order is arbitrary.
- Missing points and null points in the input series will be ignored, but NaN will not.

2.1.2 Examples

Input series:

SQL for query:

```
select distinct(s2) from root.test.d2
```

2.2 Histogram

2.2.1 Usage

This function is used to calculate the distribution histogram of a single column of numerical data.

Name: HISTOGRAM

Input Series: Only supports a single input sequence, the type is INT32 / INT64 / FLOAT / DOUBLE

Parameters:

- start: The lower limit of the requested data range, the default value is -Double.MAX_VALUE.
- end: The upper limit of the requested data range, the default value is Double.MAX_VALUE, and the value of start must be less than or equal to end.
- count: The number of buckets of the histogram, the default value is 1. It must be a positive integer.

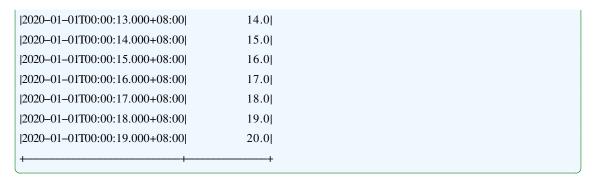
Output Series: The value of the bucket of the histogram, where the lower bound represented by the i-th bucket (index starts from 1) is $start + (i-1) \cdot \frac{end-start}{count}$ and the upper bound is $start + i \cdot \frac{end-start}{count}$.

Note:

- If the value is lower than start, it will be put into the 1st bucket. If the value is larger than end, it will be put into the last bucket.
- Missing points, null points and NaN in the input series will be ignored.

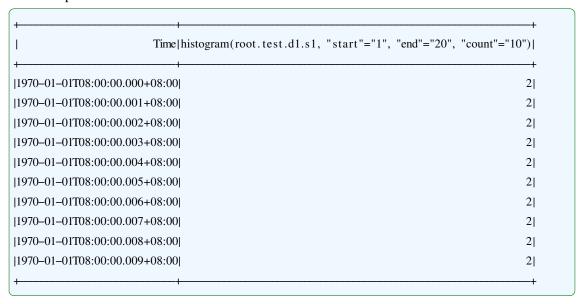
2.2.2 Examples

+	
Time	root.test.d1.s1
+	
2020-01-01T00:00:00.000+08:00	1.0
2020-01-01T00:00:01.000+08:00	2.0
2020-01-01T00:00:02.000+08:00	3.0
2020-01-01T00:00:03.000+08:00	4.0
2020-01-01T00:00:04.000+08:00	5.0
2020-01-01T00:00:05.000+08:00	6.0
2020-01-01T00:00:06.000+08:00	7.0
2020-01-01T00:00:07.000+08:00	8.0
2020-01-01T00:00:08.000+08:00	9.0
2020-01-01T00:00:09.000+08:00	10.0
2020-01-01T00:00:10.000+08:00	11.0
2020-01-01T00:00:11.000+08:00	12.0
2020-01-01T00:00:12.000+08:00	13.0



```
select histogram(s1,"start"="1","end"="20","count"="10") from root.test.d1
```

Output series:



2.2.2.1 examples on zeppelin

link: http://101.6.15.213:18181/#/notebook/2GC1HE97R

2.3 Integral

2.3.1 Usage

This function is used to calculate the integration of time series, which equals to the area under the curve with time as X-axis and values as Y-axis.

Name: INTEGRAL

Input Series: Only support a single input numeric series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

• unit: The unit of time used when computing the integral.

The value should be chosen from "1S", "1s", "1m", "1H", "1d"(case-sensitive), and each represents taking one millisecond / second / minute / hour / day as 1.0 while calculating the area and integral.

Output Series: Output a single series. The type is DOUBLE. There is only one data point in the series, whose timestamp is 0 and value is the integration.

Note:

• The integral value equals to the sum of the areas of right-angled trapezoids consisting of each two adjacent points and the time-axis.

Choosing different unit implies different scaling of time axis, thus making it apparent to convert the value among those results with constant coefficient.

• NaN values in the input series will be ignored. The curve or trapezoids will skip these points and use the next valid point.

2.3.2 Examples

2.3.2.1 Default Parameters

With default parameters, this function will take one second as 1.0.

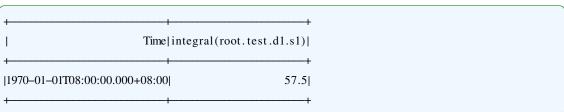
Input series:

```
Time | root.test.d1.s1|
|2020-01-01T00:00:01.000+08:00|
                                              11
|2020-01-01T00:00:02.000+08:00|
                                              21
[2020-01-01T00:00:03.000+08:00]
                                              5|
|2020-01-01T00:00:04.000+08:00|
                                              6|
|2020-01-01T00:00:05.000+08:00|
                                              71
|2020-01-01T00:00:08.000+08:00|
                                              8|
|2020-01-01T00:00:09.000+08:00|
                                            NaN|
|2020-01-01T00:00:10.000+08:00|
                                              10|
```

SQL for query:

```
select integral(s1) from root.test.d1 where time <= 2020-01-01 00:00:10
```

Output series:



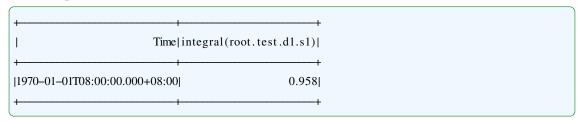
Calculation expression:

$$\frac{1}{2}[(1+2)\times 1 + (2+5)\times 1 + (5+6)\times 1 + (6+7)\times 1 + (7+8)\times 3 + (8+10)\times 2] = 57.5$$

2.3.2.2 Specific time unit

With time unit specified as "1m", this function will take one minute as 1.0. Input series is the same as above, the SQL for query is shown below:

Output series:



Calculation expression:

$$\frac{1}{2\times 60}[(1+2)\times 1 + (2+5)\times 1 + (5+6)\times 1 + (6+7)\times 1 + (7+8)\times 3 + (8+10)\times 2] = 0.958$$

2.4 Mad

2.4.1 Usage

The function is used to compute the exact or approximate median absolute deviation (MAD) of a numeric time series. MAD is the median of the deviation of each element from the elements' median.

Take a dataset $\{1, 3, 3, 5, 5, 6, 7, 8, 9\}$ as an instance. Its median is 5 and the deviation of each element from the median is $\{0, 0, 1, 2, 2, 2, 3, 4, 4\}$, whose median is 2. Therefore, the MAD of the original dataset is 2.

Name: MAD

Input Series: Only support a single input series. The data type is INT32 / INT64 / FLOAT / DOUBLE.

Parameter:

• error: The relative error of the approximate MAD. It should be within [0,1) and the default value is 0. Taking error =0.01 as an instance, suppose the exact MAD is a and the approximate MAD is b, we have $0.99a \le b \le 1.01a$. With error =0, the output is the exact MAD.

Output Series: Output a single series. The type is DOUBLE. There is only one data point in the series, whose timestamp is 0 and value is the MAD.

Note: Missing points, null points and NaN in the input series will be ignored.

2.4.2 Examples

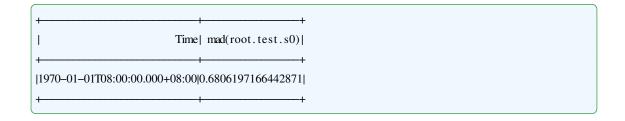
2.4.2.1 Exact Query

With the default error (error =0), the function queries the exact MAD. Input series:

```
Time | root.test.s0|
|2021-03-17T10:32:17.054+08:00|
                                 0.5319929
|2021-03-17T10:32:18.054+08:00|
                                  0.9304316
|2021-03-17T10:32:19.054+08:00|
                                 -1.4800133
|2021-03-17T10:32:20.054+08:00|
                                  0.6114087|
|2021-03-17T10:32:21.054+08:00|
                                  2.5163336
|2021-03-17T10:32:22.054+08:00|
                                 -1.0845392
|2021-03-17T10:32:23.054+08:00|
                                  1.0562582|
|2021-03-17T10:32:24.054+08:00|
                                  1.3867859
|2021-03-17T10:32:25.054+08:00| -0.45429882|
|2021-03-17T10:32:26.054+08:00|
                                  1.0353678
|2021-03-17Г10:32:27.054+08:00|
                                  0.7307929
|2021-03-17T10:32:28.054+08:00|
                                 2.3167255
|2021-03-17T10:32:29.054+08:00|
                                  2.342443|
|2021-03-17Г10:32:30.054+08:00|
                                  1.5809103
|2021-03-17T10:32:31.054+08:00|
                                  1.4829416|
|2021-03-17T10:32:32.054+08:00|
                                  1.5800357
|2021-03-17Г10:32:33.054+08:00|
                                  0.7124368|
|2021-03-17T10:32:34.054+08:00| -0.78597564|
|2021-03-17\Gamma 10:32:35.054+08:00|
                                  1.2058644|
|2021-03-17T10:32:36.054+08:00|
                                  1.4215064
|2021-03-17Г10:32:37.054+08:00|
                                  1.2808295
|2021-03-17T10:32:38.054+08:00|
                                 -0.6173715|
|2021-03-17T10:32:39.054+08:00|
                                 0.06644377|
|2021-03-17T10:32:40.054+08:00|
                                   2.349338|
|2021-03-17T10:32:41.054+08:00|
                                  1.7335888
|2021-03-17T10:32:42.054+08:00|
                                  1.5872132|
Total line number = 10000
```

SQL for query:

```
select mad(s0) from root.test
```



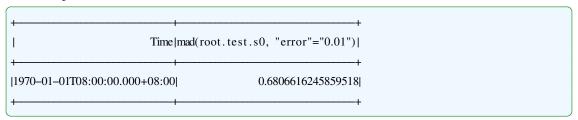
2.4.2.2 Approximate Query

By setting error within (0,1), the function queries the approximate MAD.

SQL for query:

```
select mad(s0, "error"="0.01") from root.test
```

Output series:



2.5 Median

2.5.1 Usage

The function is used to compute the exact or approximate median of a numeric time series.

Name: MEDIAN

Input Series: Only support a single input series. The data type is INT32 / INT64 / FLOAT / DOUBLE.

Parameter:

• error: The rank error of the approximate median. It should be within [0,1) and the default value is 0. For instance, a median with error =0.01 is the value of the element with rank percentage $0.49\sim0.51$. With error =0, the output is the exact median.

Output Series: Output a single series. The type is DOUBLE. There is only one data point in the series, whose timestamp is 0 and value is the median.

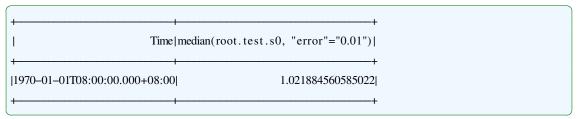
2.5.2 Examples



```
|2021-03-17T10:32:17.054+08:00|
                                 0.5319929|
|2021-03-17T10:32:18.054+08:00|
                                  0.9304316
|2021-03-17T10:32:19.054+08:00|
                                -1.4800133
|2021-03-17T10:32:20.054+08:00|
                                  0.6114087|
|2021-03-17T10:32:21.054+08:00|
                                  2.5163336|
|2021-03-17T10:32:22.054+08:00|
                                -1.0845392
|2021-03-17T10:32:23.054+08:00|
                                  1.0562582|
|2021-03-17T10:32:24.054+08:00|
                                  1.3867859
|2021 - 03 - 17\Gamma 10:32:25.054 + 08:00| \ -0.45429882|
|2021-03-17T10:32:26.054+08:00|
                                  1.0353678
|2021-03-17T10:32:27.054+08:00|
                                  0.7307929
|2021-03-17T10:32:28.054+08:00|
                                 2.3167255
|2021-03-17T10:32:29.054+08:00|
                                   2.342443|
|2021-03-17Г10:32:30.054+08:00|
                                  1.5809103|
|2021-03-17T10:32:31.054+08:00|
                                  1.4829416
|2021-03-17T10:32:32.054+08:00|
                                  1.5800357
|2021-03-17T10:32:33.054+08:00|
                                  0.7124368
|2021-03-17T10:32:34.054+08:00| -0.78597564|
|2021-03-17T10:32:35.054+08:00|
                                 1.2058644|
|2021-03-17Г10:32:36.054+08:00|
                                 1.4215064
|2021-03-17Г10:32:37.054+08:00|
                                  1.2808295|
|2021-03-17T10:32:38.054+08:00|
                                -0.6173715|
|2021-03-17T10:32:39.054+08:00| 0.06644377|
|2021-03-17T10:32:40.054+08:00|
                                   2.349338|
|2021-03-17T10:32:41.054+08:00|
                                 1.7335888
|2021-03-17T10:32:42.054+08:00|
                                 1.5872132
Total line number = 10000
```

```
select median(s0, "error"="0.01") from root.test
```

Output series:



2.6 Mode

2.6.1 Usage

This function is used to calculate the mode of time series, that is, the value that occurs most frequently.

Name: MODE

Input Series: Only support a single input series. The type is arbitrary.

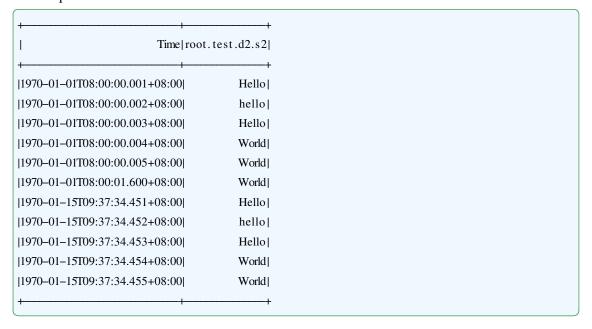
Output Series: Output a single series. The type is the same as the input. There is only one data point in the series, whose timestamp is 0 and value is the mode.

Note:

- If there are multiple values with the most occurrences, the arbitrary one will be output.
- Missing points and null points in the input series will be ignored, but NaN will not.

2.6.2 Examples

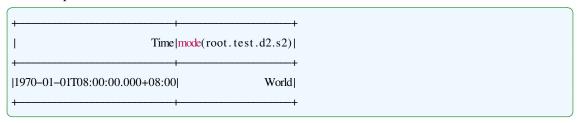
Input series:



SQL for query:

```
select mode(s2) from root.test.d2
```

Output series:



2.7 Percentile

2.7.1 Usage

The function is used to compute the exact or approximate percentile of a numeric time series. A percentile is value of element in the certain rank of the sorted series.

Name: PERCENTILE

Input Series: Only support a single input series. The data type is INT32 / INT64 / FLOAT / DOUBLE.

Parameter:

- rank: The rank percentage of the percentile. It should be (0,1] and the default value is 0.5. For instance, a percentile with rank =0.5 is the median.
- error: The rank error of the approximate percentile. It should be within [0,1) and the default value is 0. For instance, a 0.5-percentile with error =0.01 is the value of the element with rank percentage 0.49~0.51. With error =0, the output is the exact percentile.

Output Series: Output a single series. The type is DOUBLE. There is only one data point in the series, whose timestamp is 0 and value is the percentile.

Note: Missing points, null points and NaN in the input series will be ignored.

2.7.2 Examples

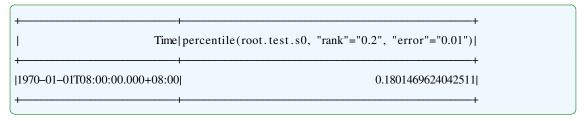
```
Time | root.test.s0|
|2021-03-17T10:32:17.054+08:00|
                                0.5319929
|2021-03-17T10:32:18.054+08:00|
                                 0.9304316
|2021-03-17T10:32:19.054+08:00| -1.4800133|
|2021-03-17Г10:32:20.054+08:00|
                                 0.6114087
|2021-03-17T10:32:21.054+08:00|
                                 2.5163336
|2021-03-17T10:32:22.054+08:00| -1.0845392|
|2021-03-17T10:32:23.054+08:00|
                                 1.0562582
|2021-03-17T10:32:24.054+08:00|
                                 1.3867859
|2021-03-17T10:32:25.054+08:00| -0.45429882|
|2021-03-17T10:32:26.054+08:00|
                                 1.0353678
|2021-03-17Г10:32:27.054+08:00|
                                 0.7307929
|2021-03-17T10:32:28.054+08:00|
                                 2.3167255|
|2021-03-17T10:32:29.054+08:00|
                                  2.342443|
|2021-03-17T10:32:30.054+08:00|
                                 1.5809103|
|2021-03-17T10:32:31.054+08:00|
                                 1.4829416
|2021-03-17T10:32:32.054+08:00|
                                 1.5800357
|2021-03-17Г10:32:33.054+08:00|
                                 0.7124368
|2021-03-17T10:32:34.054+08:00| -0.78597564|
|2021-03-17T10:32:35.054+08:00|
                                 1.2058644
|2021-03-17T10:32:36.054+08:00|
                                 1.4215064
|2021-03-17Г10:32:37.054+08:00|
                                 1.2808295|
|2021-03-17T10:32:38.054+08:00|
                                -0.6173715
|2021-03-17T10:32:39.054+08:00| 0.06644377|
|2021-03-17T10:32:40.054+08:00|
                                  2.349338
|2021-03-17Г10:32:41.054+08:00|
                                 1.7335888
```

```
|2021-03-17T10:32:42.054+08:00| 1.5872132| ......

Total line number = 10000
```

```
select percentile(s0, "rank"="0.2", "error"="0.01") from root.test
```

Output series:



2.8 Period

2.8.1 Usage

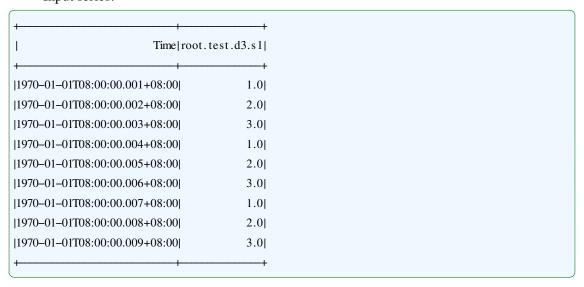
The function is used to compute the period of a numeric time series.

Name: PERIOD

Input Series: Only support a single input series. The data type is INT32 / INT64 / FLOAT / DOUBLE.

Output Series: Output a single series. The type is INT32. There is only one data point in the series, whose timestamp is 0 and value is the period.

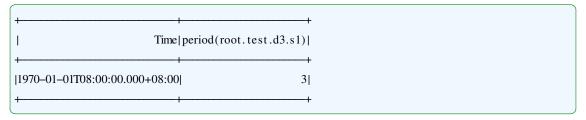
2.8.2 Examples



SQL for query:

```
select period(s1) from root.test.d3
```

Output series:



2.8.2.1 examples on zeppelin

link: http://101.6.15.213:18181/#/notebook/2GEJBUSZ9

2.9 QLB

2.9.1 Usage

This function is used to calculate Ljung-Box statistics Q_{LB} for time series, and convert it to p value.

Name: QLB

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

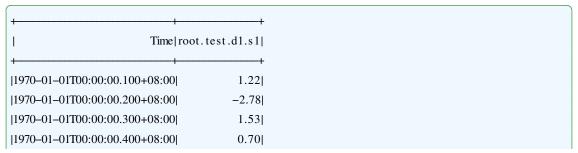
shift: max lag to calculate. Legal input shall be integer from 1 to n-2, where n is the sample number. Default value is n-2.

Output Series: Output a single series. The type is DOUBLE. The output series is p value, and timestamp means lag.

Note: If you want to calculate Ljung-Box statistics Q_{LB} instead of p value, you may use AutoCorrelation function.

2.9.2 Examples

2.9.2.1 Using Default Parameter



```
|1970-01-01T00:00:00.500+08:00|
                                          0.75|
|1970-01-01T00:00:00.600+08:00|
                                         -0.72
|1970-01-01T00:00:00.700+08:00|
                                         -0.22
|1970-01-01T00:00:00.800+08:00|
                                          0.28
|1970-01-01T00:00:00.900+08:00|
                                          0.57|
|1970-01-01T00:00:01.000+08:00|
                                         -0.22
|1970-01-01T00:00:01.100+08:00|
                                         -0.72|
|1970-01-01T00:00:01.200+08:00|
                                          1.34
|1970-01-01T00:00:01.300+08:00|
                                         -0.25
|1970-01-01T00:00:01.400+08:00|
                                          0.17|
|1970-01-01T00:00:01.500+08:00|
                                          2.51
|1970-01-01T00:00:01.600+08:00|
                                          1.42
|1970-01-01T00:00:01.700+08:00|
                                         -1.34|
|1970-01-01T00:00:01.800+08:00|
                                         -0.01
|1970-01-01T00:00:01.900+08:00|
                                         -0.49
|1970-01-01T00:00:02.000+08:00|
                                          1.63|
```

```
select QLB(s1) from root.test.dl
```

```
Time|QLB(root.test.d1.s1)|
[1970-01-01T00:00:00.001+08:00] 0.2168702295315677[
|1970-01-01T00:00:00.002+08:00| \quad 0.3068948509261751|
|1970-01-01T00:00:00.003+08:00| 0.4217859150918444|
[1970-01-01T00:00:00.004+08:00] 0.5114539874276656[
[1970-01-01T00:00:00.005+08:00] 0.6560619525616759[
[1970-01-01T00:00:00.006+08:00] 0.7722398654053280]
|1970-01-01T00:00:00.007+08:00|
                                0.8532491661465290
|1970-01-01T00:00:00.008+08:00| \quad 0.9028575017542528|
[1970-01-01T00:00:00.009+08:00] 0.9434989988192729[
|1970-01-01T00:00:00.010+08:00| 0.8950280161464689|
|1970-01-01T00:00:00.011+08:00| 0.7701048398839656|
[1970-01-01T00:00:00.012+08:00] 0.7845536060001281]
[1970-01-01T00:00:00.013+08:00] 0.5943030981705825[
[1970-01-01T00:00:00.014+08:00] 0.4618413512531093[
[1970-01-01T00:00:00.015+08:00] 0.2645948244673964[
|1970-01-01T00:00:00.016+08:00| 0.3167530476666645|
|1970-01-01T00:00:00.017+08:00|
                                0.2330010780351453
[1970-01-01T00:00:00.018+08:00] 0.0666611237622325[
```

2.10 Resample

2.10.1 Usage

This function is used to resample the input series according to a given frequency, including up-sampling and down-sampling. Currently, the supported up-sampling methods are NaN (filling with NaN), FFill (filling with previous value), BFill (filling with next value) and Linear (filling with linear interpolation). Down-sampling relies on group aggregation, which supports Max, Min, First, Last, Mean and Median.

Name: RESAMPLE

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

- every: The frequency of resampling, which is a positive number with an unit. The unit is 'ms' for millisecond, 's' for second, 'm' for minute, 'h' for hour and 'd' for day. This parameter cannot be lacked.
- interp: The interpolation method of up-sampling, which is 'NaN', 'FFill', 'BFill' or 'Linear'. By default, NaN is used.
- aggr: The aggregation method of down-sampling, which is 'Max', 'Min', 'First', 'Last', 'Mean' or 'Median'. By default, Mean is used.
- start: The start time (inclusive) of resampling with the format 'yyyy-MM-dd HH:mm:ss'. By default, it is the timestamp of the first valid data point.
- end: The end time (exclusive) of resampling with the format 'yyyy-MM-dd HH:mm:ss'. By default, it is the timestamp of the last valid data point.

Output Series: Output a single series. The type is DOUBLE. It is strictly equispaced with the frequency every.

Note: NaN in the input series will be ignored.

2.10.2 Examples

2.10.2.1 Up-sampling

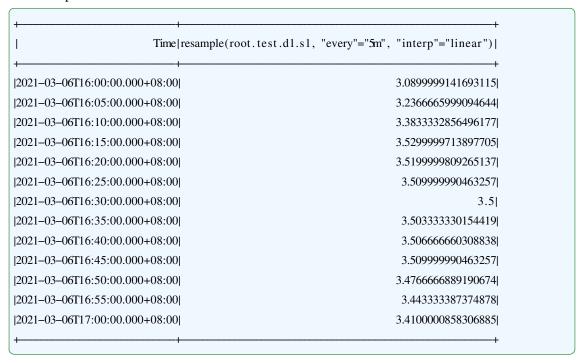
When the frequency of resampling is higher than the original frequency, up-sampling starts. Input series:

1	+
Time	root.test.d1.s1
+	++
2021-03-06T16:00:00.000+08:00	3.09
2021-03-06T16:15:00.000+08:00	3.53
2021-03-06T16:30:00.000+08:00	3.5
2021-03-06T16:45:00.000+08:00	3.51

```
|2021-03-06T17:00:00.000+08:00| 3.41|
+ + +
```

```
select resample(s1, 'every'='5m', 'interp'='linear') from root.test.dl
```

Output series:

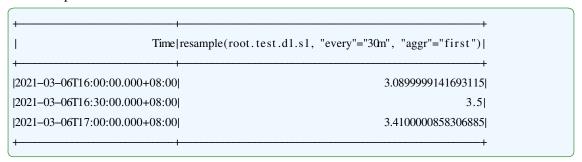


2.10.2.2 Down-sampling

When the frequency of resampling is lower than the original frequency, down-sampling starts.

Input series is the same as above, the SQL for query is shown below:

```
select resample(s1, 'every'='30m', 'aggr'='first') from root.test.dl
```



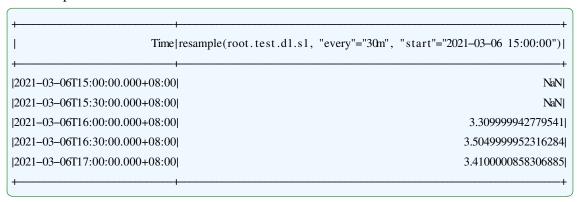
2.10.2.3 Specify the time period

The time period of resampling can be specified with start and end. The period outside the actual time range will be interpolated.

Input series is the same as above, the SQL for query is shown below:

```
select resample(s1, 'every'='30m', 'start'='2021-03-06_15:00:00') from root.test.dl
```

Output series:



2.11 Sample

2.11.1 Usage

This function is used to sample the input series, that is, select a specified number of data points from the input series and output them. Currently, two sampling methods are supported: **Reservoir sampling** randomly selects data points. All of the points have the same probability of being sampled. **Isometric sampling** selects data points at equal index intervals.

Name: SAMPLE

Input Series: Only support a single input series. The type is arbitrary.

Parameters:

- method: The method of sampling, which is 'reservoir' or 'isometric'. By default, reservoir sampling is used.
- k: The number of sampling, which is a positive integer. By default, it's 1.

Output Series: Output a single series. The type is the same as the input. The length of the output series is k. Each data point in the output series comes from the input series.

Note: If k is greater than the length of input series, all data points in the input series will be output.

2.11.2 Examples

2.11.2.1 Reservoir Sampling

When method is 'reservoir' or the default, reservoir sampling is used. Due to the randomness of this method, the output series shown below is only a possible result.

Input series:

```
Time | root.test.d1.s1|
|2020-01-01T00:00:01.000+08:00|
                                           1.0
|2020-01-01T00:00:02.000+08:00|
                                           2.0|
|2020-01-01T00:00:03.000+08:00|
                                           3.0
|2020-01-01T00:00:04.000+08:00|
                                           4.0
|2020-01-01T00:00:05.000+08:00|
                                           5.0
|2020-01-01T00:00:06.000+08:00|
                                           6.0|
|2020-01-01T00:00:07.000+08:00|
                                           7.0|
[2020-01-01T00:00:08.000+08:00]
                                           8.0|
|2020-01-01T00:00:09.000+08:00|
                                           9.0|
|2020-01-01T00:00:10.000+08:00|
                                           10.0
```

SQL for query:

```
select sample(s1, 'method'='reservoir', 'k'='5') from root.test.dl
```

Output series:

2.11.2.2 Isometric Sampling

When method is 'isometric', isometric sampling is used.

Input series is the same as above, the SQL for query is shown below:

```
select sample(s1, 'method'='isometric', 'k'='5') from root.test.dl
```

+ + + Time	sample(root.test.d1.s1, "method"="isometric", "k"="5")
1	
2020-01-01T00:00:01.000+08:00	1.0
2020-01-01T00:00:03.000+08:00	3.0
2020-01-01T00:00:05.000+08:00	5.0
2020-01-01T00:00:07.000+08:00	7.0
2020-01-01T00:00:09.000+08:00	9.0
 	+

2.12 Segment

2.12.1 Usage

This function is used to segment a time series into subsequences according to linear trend, and returns linear fitted values of first values in each subsequence or every data point.

Name: SEGMENT

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

- output :"all" to output all fitted points; "first" to output first fitted points in each subsequence.
- error: error allowed at linear regression. It is defined as mean absolute error of a subsequence.

Output Series: Output a single series. The type is DOUBLE.

Note: This function treat input series as equal-interval sampled. All data are loaded, so downsample input series first if there are too many data points.

2.12.2 Examples

```
Time | root.test.s1|
|1970-01-01T08:00:00.000+08:00|
                                         5.0
|1970-01-01T08:00:00.100+08:00|
                                         |0.0|
|1970-01-01T08:00:00.200+08:00|
                                         1.0|
|1970-01-01T08:00:00.300+08:00|
                                         2.0|
|1970-01-01T08:00:00.400+08:00|
                                         3.0|
|1970-01-01T08:00:00.500+08:00|
                                         4.0|
|1970-01-01T08:00:00.600+08:00|
                                         5.0
|1970-01-01T08:00:00.700+08:00|
                                         6.0|
```

```
|1970-01-01T08:00:00.800+08:00|
                                        7.0|
|1970-01-01T08:00:00.900+08:00|
                                        8.0
|1970-01-01T08:00:01.000+08:00|
                                        9.0
|1970-01-01T08:00:01.100+08:00|
                                        9.1|
|1970-01-01T08:00:01.200+08:00|
                                        9.2|
|1970-01-01T08:00:01.300+08:00|
                                        9.3
|1970-01-01T08:00:01.400+08:00|
                                        9.4
|1970-01-01T08:00:01.500+08:00|
                                        9.5
|1970-01-01T08:00:01.600+08:00|
                                        9.6
|1970-01-01T08:00:01.700+08:00|
                                        9.7
|1970-01-01T08:00:01.800+08:00|
                                        9.8
|1970-01-01T08:00:01.900+08:00|
                                        9.9
|1970-01-01T08:00:02.000+08:00|
                                        10.0
|1970-01-01T08:00:02.100+08:00|
                                        8.0|
|1970-01-01T08:00:02.200+08:00|
                                        6.0|
|1970-01-01T08:00:02.300+08:00|
                                        4.0|
|1970-01-01T08:00:02.400+08:00|
                                        2.0|
|1970-01-01T08:00:02.500+08:00|
                                        |0.0|
|1970-01-01T08:00:02.600+08:00|
                                       -2.0|
|1970-01-01T08:00:02.700+08:00|
                                       -4.0|
|1970-01-01T08:00:02.800+08:00|
                                       -6.0
|1970-01-01T08:00:02.900+08:00|
                                       -8.0
|1970-01-01T08:00:03.000+08:00|
                                      -10.0|
|1970-01-01T08:00:03.100+08:00|
                                       10.0|
|1970-01-01T08:00:03.200+08:00|
                                       10.0
|1970-01-01T08:00:03.300+08:00|
                                       10.0|
|1970-01-01T08:00:03.400+08:00|
                                       10.0
|1970-01-01T08:00:03.500+08:00|
                                       10.0|
|1970-01-01T08:00:03.600+08:00|
                                       10.0|
|1970-01-01T08:00:03.700+08:00|
                                       10.0
|1970-01-01T08:00:03.800+08:00|
                                       10.0|
|1970-01-01T08:00:03.900+08:00|
                                       10.0|
```

```
select segment(s1, "error"="0.1") from root.test
```

2.13 Skew

2.13.1 Usage

This function is used to calculate the population skewness.

Name: SKEW

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Output Series: Output a single series. The type is DOUBLE. There is only one data point in the series, whose timestamp is 0 and value is the population skewness.

Note: Missing points, null points and NaN in the input series will be ignored.

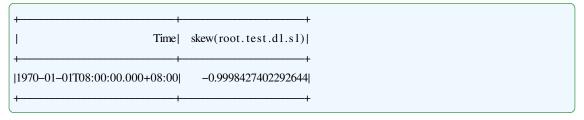
2.13.2 Examples

	+
Time	root.test.d1.s1
+	+
2020-01-01T00:00:00.000+08:00	1.0
2020-01-01T00:00:01.000+08:00	2.0
2020-01-01T00:00:02.000+08:00	3.0
2020-01-01T00:00:03.000+08:00	4.0
2020-01-01T00:00:04.000+08:00	5.0
2020-01-01T00:00:05.000+08:00	6.0
2020-01-01T00:00:06.000+08:00	7.0
2020-01-01T00:00:07.000+08:00	8.0
2020-01-01T00:00:08.000+08:00	9.0
2020-01-01T00:00:09.000+08:00	10.0
2020-01-01T00:00:10.000+08:00	10.0
2020-01-01T00:00:11.000+08:00	10.0
2020-01-01T00:00:12.000+08:00	10.0
2020-01-01T00:00:13.000+08:00	10.0
2020-01-01T00:00:14.000+08:00	10.0
2020-01-01T00:00:15.000+08:00	10.0
2020-01-01T00:00:16.000+08:00	10.0
2020-01-01T00:00:17.000+08:00	10.0
2020-01-01T00:00:18.000+08:00	10.0
2020-01-01T00:00:19.000+08:00	10.0
+	+

SQL for query:

```
select skew(s1) from root.test.d1
```

Output series:



2.14 Spread

2.14.1 Usage

This function is used to calculate the spread of time series, that is, the maximum value minus the minimum value.

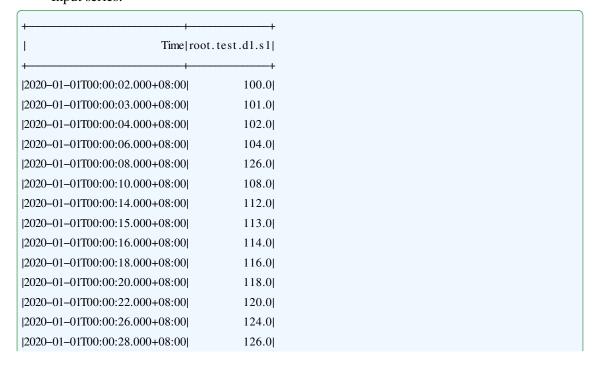
Name: SPREAD

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Output Series: Output a single series. The type is the same as the input. There is only one data point in the series, whose timestamp is 0 and value is the spread.

Note: Missing points, null points and NaN in the input series will be ignored.

2.14.2 Examples



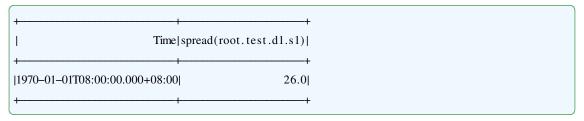
2.15 Stddev



SQL for query:

```
select spread(s1) from root.test.d1 where time <= 2020-01-01 00:00:30
```

Output series:



2.15 Stddev

2.15.1 Usage

This function is used to calculate the population standard deviation.

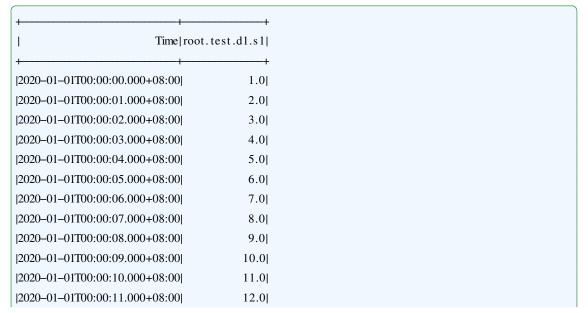
Name: STDDEV

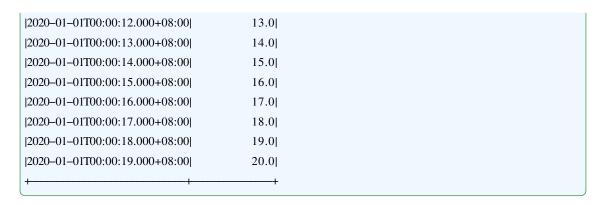
Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Output Series: Output a single series. The type is DOUBLE. There is only one data point in the series, whose timestamp is 0 and value is the population standard deviation.

Note: Missing points, null points and NaN in the input series will be ignored.

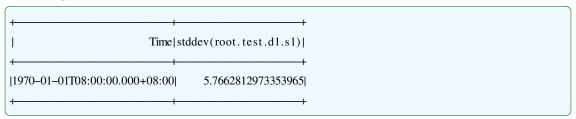
2.15.2 Examples





```
select stddev(s1) from root.test.dl
```

Output series:



2.16 TimeWeightedAvg

2.16.1 Usage

This function is used to calculate the time-weighted average of time series. Time is weighted using the linearly interpolated integral of values, and the output equals to the area divided by the time interval using the same time unit. For more information of the area under the curve, please refer to Integral function.

Name: TIMEWEIGHTEDAVG

Input Series: Only support a single input numeric series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Output Series: Output a single series. The type is DOUBLE. There is only one data point in the series, whose timestamp is 0 and value is the time-weighted average.

Note:

• The time-weighted value equals to the integral value with any unit divided by the time interval of input series.

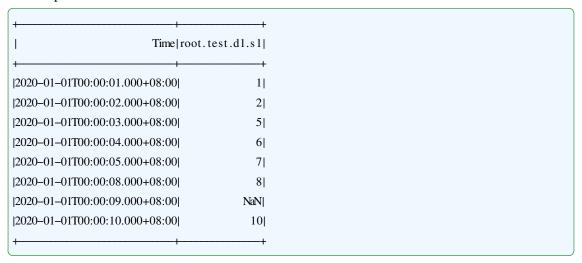
The result is irrelevant to the time unit used in integral, and it's consistent with the timestamp precision of IoTDB by default.

• NaN values in the input series will be ignored. The curve or trapezoids will skip these points and use the next valid point.

• If the input series is empty, the output value will be 0.0, but if there is only one data point, the value will equal to the input value.

2.16.2 Examples

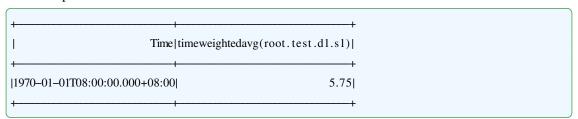
Input series:



SQL for query:

```
select timeweightedayg(s1) from root.test.d1 where time <= 2020-01-01 00:00:10
```

Output series:



Calculation expression:

$$\frac{1}{2}[(1+2)\times 1 + (2+5)\times 1 + (5+6)\times 1 + (6+7)\times 1 + (7+8)\times 3 + (8+10)\times 2]/10 = 5.75$$

Chapter 3 Data Quality

3.1 Completeness

3.1.1 Usage

This function is used to calculate the completeness of time series. The input series are divided into several continuous and non overlapping windows. The timestamp of the first data point and the completeness of each window will be output.

Name: COMPLETENESS

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

- window: The size of each window. It is a positive integer or a positive number with an unit. The former is the number of data points in each window. The number of data points in the last window may be less than it. The latter is the time of the window. The unit is 'ms' for millisecond, 's' for second, 'm' for minute, 'h' for hour and 'd' for day. By default, all input data belongs to the same window.
- downtime: Whether the downtime exception is considered in the calculation of completeness. It is 'true' or 'false' (default). When considering the downtime exception, long-term missing data will be considered as downtime exception without any influence on completeness.

Output Series: Output a single series. The type is DOUBLE. The range of each value is [0,1].

Note: Only when the number of data points in the window exceeds 10, the calculation will be performed. Otherwise, the window will be ignored and nothing will be output.

3.1.2 Examples

3.1.2.1 Default Parameters

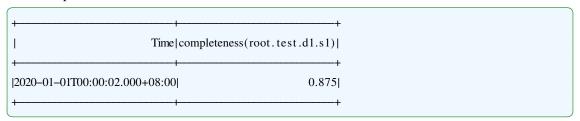
With default parameters, this function will regard all input data as the same window. Input series:

+	++
Time	root.test.d1.s1
+	++
2020-01-01T00:00:02.000+08:00	100.0
2020-01-01T00:00:03.000+08:00	101.0
2020-01-01T00:00:04.000+08:00	102.0
2020-01-01T00:00:06.000+08:00	104.0

+	+
2020-01-01T00:00:30.000+08:00	NaN
2020-01-01T00:00:28.000+08:00	126.0
2020-01-01T00:00:26.000+08:00	124.0
2020-01-01T00:00:22.000+08:00	120.0
2020-01-01T00:00:20.000+08:00	118.0
2020-01-01T00:00:18.000+08:00	116.0
2020-01-01T00:00:16.000+08:00	114.0
2020-01-01T00:00:15.000+08:00	113.0
2020-01-01T00:00:14.000+08:00	112.0
2020-01-01T00:00:10.000+08:00	108.0
2020-01-01T00:00:08.000+08:00	126.0

```
select completeness(s1) from root.test.d1 where time <= 2020-01-01 00:00:30
```

Output series:



3.1.2.2 Specific Window Size

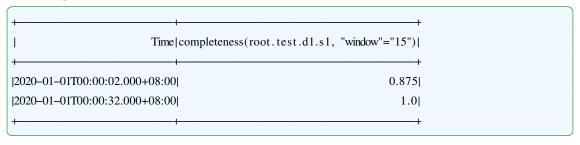
When the window size is given, this function will divide the input data as multiple windows. Input series:

```
Time | root.test.d1.s1|
                                         100.0
|2020-01-01T00:00:02.000+08:00|
|2020-01-01T00:00:03.000+08:00|
                                         101.0
|2020-01-01T00:00:04.000+08:00|
                                         102.0
|2020-01-01T00:00:06.000+08:00|
                                         104.0|
|2020-01-01T00:00:08.000+08:00|
                                         126.0
|2020-01-01T00:00:10.000+08:00|
                                         108.0
|2020-01-01T00:00:14.000+08:00|
                                         112.0
|2020-01-01T00:00:15.000+08:00|
                                         113.0|
|2020-01-01T00:00:16.000+08:00|
                                         114.0
|2020-01-01T00:00:18.000+08:00|
                                         116.0
|2020-01-01T00:00:20.000+08:00|
                                         118.0
|2020-01-01T00:00:22.000+08:00|
                                         120.0
|2020-01-01T00:00:26.000+08:00|
                                         124.0
|2020-01-01T00:00:28.000+08:00|
                                         126.0
|2020-01-01T00:00:30.000+08:00|
                                          NaN|
```

2020-01-01T00:00:32.000+08:00	130.0	
2020-01-01T00:00:34.000+08:00	132.0	
2020-01-01T00:00:36.000+08:00	134.0	
2020-01-01T00:00:38.000+08:00	136.0	
2020-01-01T00:00:40.000+08:00	138.0	
2020-01-01T00:00:42.000+08:00	140.0	
2020-01-01T00:00:44.000+08:00	142.0	
2020-01-01T00:00:46.000+08:00	144.0	
2020-01-01T00:00:48.000+08:00	146.0	
2020-01-01T00:00:50.000+08:00	148.0	
2020-01-01T00:00:52.000+08:00	150.0	
2020-01-01T00:00:54.000+08:00	152.0	
2020-01-01T00:00:56.000+08:00	154.0	
2020-01-01T00:00:58.000+08:00	156.0	
2020-01-01T00:01:00.000+08:00	158.0	
+	+	

```
select completeness(s1, "window"="15") from root.test.d1 where time <= 2020-01-01 00:01:00
```

Output series:



3.2 Consistency

3.2.1 Usage

This function is used to calculate the consistency of time series. The input series are divided into several continuous and non overlapping windows. The timestamp of the first data point and the consistency of each window will be output.

Name: CONSISTENCY

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

• window: The size of each window. It is a positive integer or a positive number with an unit. The former is the number of data points in each window. The number of data points in the last window may be less than it. The latter is the time of the window. The unit is

'ms' for millisecond, 's' for second, 'm' for minute, 'h' for hour and 'd' for day. By default, all input data belongs to the same window.

Output Series: Output a single series. The type is DOUBLE. The range of each value is [0,1].

Note: Only when the number of data points in the window exceeds 10, the calculation will be performed. Otherwise, the window will be ignored and nothing will be output.

3.2.2 Examples

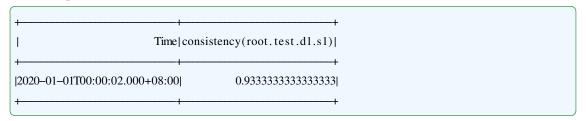
3.2.2.1 Default Parameters

With default parameters, this function will regard all input data as the same window. Input series:

+		+
I	Time root	.test.d1.s1
+	t	+
2020-01-01T00:00:02.00	0+08:00	100.0
2020-01-01T00:00:03.00	0+08:00	101.0
2020-01-01T00:00:04.00	0+08:00	102.0
2020-01-01T00:00:06.00	0+08:00	104.0
2020-01-01T00:00:08.00	0+08:00	126.0
2020-01-01T00:00:10.00	0+08:00	108.0
2020-01-01T00:00:14.00	0+08:00	112.0
2020-01-01T00:00:15.00	0+08:00	113.0
2020-01-01T00:00:16.00	0+08:00	114.0
2020-01-01T00:00:18.00	0+08:00	116.0
2020-01-01T00:00:20.00	0+08:00	118.0
2020-01-01T00:00:22.00	0+08:00	120.0
2020-01-01T00:00:26.00	0+08:00	124.0
2020-01-01T00:00:28.00	0+08:00	126.0
2020-01-01T00:00:30.00	0+08:00	NaN
+		+

SQL for query:

```
select consistency(s1) from root.test.d1 where time <= 2020-01-01 00:00:30
```



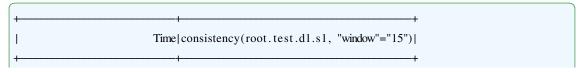
3.2.2.2 Specific Window Size

When the window size is given, this function will divide the input data as multiple windows. Input series:

	L
Time root.	test.d1.s1
+	·
2020-01-01T00:00:02.000+08:00	100.0
2020-01-01T00:00:03.000+08:00	101.0
2020-01-01T00:00:04.000+08:00	102.0
2020-01-01T00:00:06.000+08:00	104.0
2020-01-01T00:00:08.000+08:00	126.0
2020-01-01T00:00:10.000+08:00	108.0
2020-01-01T00:00:14.000+08:00	112.0
2020-01-01T00:00:15.000+08:00	113.0
2020-01-01T00:00:16.000+08:00	114.0
2020-01-01T00:00:18.000+08:00	116.0
2020-01-01T00:00:20.000+08:00	118.0
2020-01-01T00:00:22.000+08:00	120.0
2020-01-01T00:00:26.000+08:00	124.0
2020-01-01T00:00:28.000+08:00	126.0
2020-01-01T00:00:30.000+08:00	NaN
2020-01-01T00:00:32.000+08:00	130.0
2020-01-01T00:00:34.000+08:00	132.0
2020-01-01T00:00:36.000+08:00	134.0
2020-01-01T00:00:38.000+08:00	136.0
2020-01-01T00:00:40.000+08:00	138.0
2020-01-01T00:00:42.000+08:00	140.0
2020-01-01T00:00:44.000+08:00	142.0
2020-01-01T00:00:46.000+08:00	144.0
2020-01-01T00:00:48.000+08:00	146.0
2020-01-01T00:00:50.000+08:00	148.0
2020-01-01T00:00:52.000+08:00	150.0
2020-01-01T00:00:54.000+08:00	152.0
2020-01-01T00:00:56.000+08:00	154.0
2020-01-01T00:00:58.000+08:00	156.0
2020-01-01T00:01:00.000+08:00	158.0
+	+

SQL for query:

```
select consistency(s1, "window"="15") from root.test.d1 where time <= 2020-01-01 00:01:00
```



2020-01-01T00:00:02.000+08:00	0.933333333333333333	
2020-01-01T00:00:32.000+08:00	1.0	
+	+	

3.3 Timeliness

3.3.1 Usage

This function is used to calculate the timeliness of time series. The input series are divided into several continuous and non overlapping windows. The timestamp of the first data point and the timeliness of each window will be output.

Name: TIMELINESS

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

• window: The size of each window. It is a positive integer or a positive number with an unit. The former is the number of data points in each window. The number of data points in the last window may be less than it. The latter is the time of the window. The unit is 'ms' for millisecond, 's' for second, 'm' for minute, 'h' for hour and 'd' for day. By default, all input data belongs to the same window.

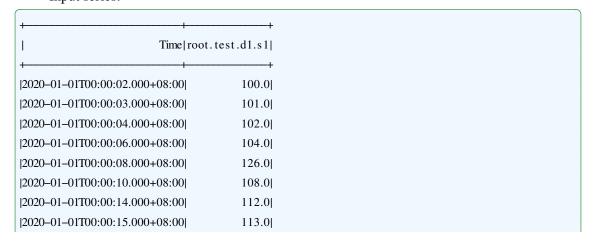
Output Series: Output a single series. The type is DOUBLE. The range of each value is [0,1].

Note: Only when the number of data points in the window exceeds 10, the calculation will be performed. Otherwise, the window will be ignored and nothing will be output.

3.3.2 Examples

3.3.2.1 Default Parameters

With default parameters, this function will regard all input data as the same window. Input series:



```
|2020-01-01T00:00:16.000+08:00| 114.0|

|2020-01-01T00:00:18.000+08:00| 116.0|

|2020-01-01T00:00:20.000+08:00| 118.0|

|2020-01-01T00:00:22.000+08:00| 120.0|

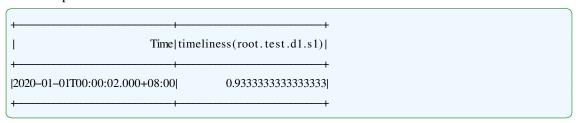
|2020-01-01T00:00:26.000+08:00| 124.0|

|2020-01-01T00:00:28.000+08:00| 126.0|

|2020-01-01T00:00:30.000+08:00| NaN|
```

```
select timeliness(s1) from root.test.d1 where time <= 2020-01-01 00:00:30
```

Output series:



3.3.2.2 Specific Window Size

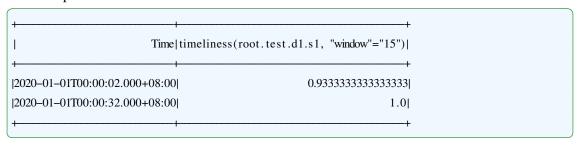
When the window size is given, this function will divide the input data as multiple windows. Input series:

+	+
Time	root.test.d1.s1
+	+
2020-01-01T00:00:02.000+08:00	100.0
2020-01-01T00:00:03.000+08:00	101.0
2020-01-01T00:00:04.000+08:00	102.0
2020-01-01T00:00:06.000+08:00	104.0
2020-01-01T00:00:08.000+08:00	126.0
2020-01-01T00:00:10.000+08:00	108.0
2020-01-01T00:00:14.000+08:00	112.0
2020-01-01T00:00:15.000+08:00	113.0
2020-01-01T00:00:16.000+08:00	114.0
2020-01-01T00:00:18.000+08:00	116.0
2020-01-01T00:00:20.000+08:00	118.0
2020-01-01T00:00:22.000+08:00	120.0
2020-01-01T00:00:26.000+08:00	124.0
2020-01-01T00:00:28.000+08:00	126.0
2020-01-01T00:00:30.000+08:00	NaN
2020-01-01T00:00:32.000+08:00	130.0
2020-01-01T00:00:34.000+08:00	132.0
2020-01-01T00:00:36.000+08:00	134.0
2020-01-01T00:00:38.000+08:00	136.0

2020–01–01T00:01:00.000+08:00 	158.0
2020-01-01T00:00:58.000+08:00	156.0
2020-01-01T00:00:56.000+08:00	154.0
2020-01-01T00:00:54.000+08:00	152.0
2020-01-01T00:00:52.000+08:00	150.0
2020-01-01T00:00:50.000+08:00	148.0
2020-01-01T00:00:48.000+08:00	146.0
2020-01-01T00:00:46.000+08:00	144.0
2020-01-01T00:00:44.000+08:00	142.0
2020-01-01T00:00:42.000+08:00	140.0
2020-01-01T00:00:40.000+08:00	138.0

```
select timeliness(s1, "window"="15") from root.test.d1 where time <= 2020-01-01 00:01:00
```

Output series:



3.4 Validity

3.4.1 Usage

This function is used to calculate the Validity of time series. The input series are divided into several continuous and non overlapping windows. The timestamp of the first data point and the Validity of each window will be output.

Name: VALIDITY

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

• window: The size of each window. It is a positive integer or a positive number with an unit. The former is the number of data points in each window. The number of data points in the last window may be less than it. The latter is the time of the window. The unit is 'ms' for millisecond, 's' for second, 'm' for minute, 'h' for hour and 'd' for day. By default, all input data belongs to the same window.

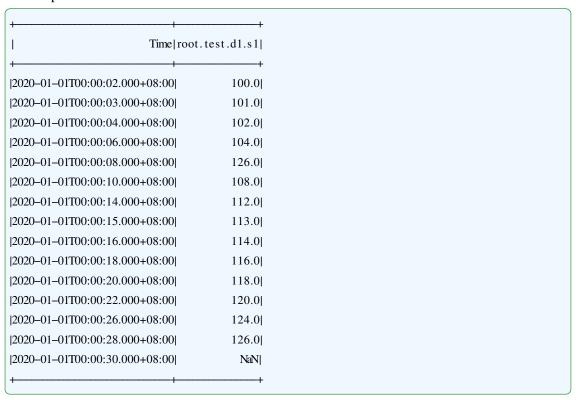
Output Series: Output a single series. The type is DOUBLE. The range of each value is [0,1].

Note: Only when the number of data points in the window exceeds 10, the calculation will be performed. Otherwise, the window will be ignored and nothing will be output.

3.4.2 Examples

3.4.2.1 Default Parameters

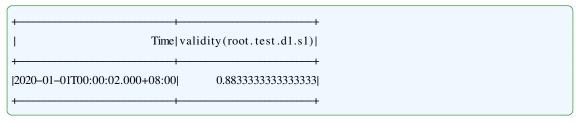
With default parameters, this function will regard all input data as the same window. Input series:



SQL for query:

```
select Validity(s1) from root.test.d1 where time <= 2020-01-01 00:00:30
```

Output series:



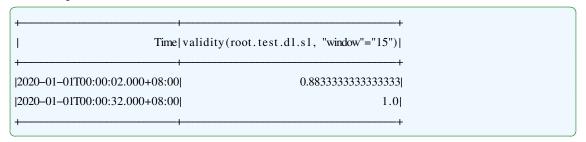
3.4.2.2 Specific Window Size

When the window size is given, this function will divide the input data as multiple windows. Input series:

```
<del>'</del>
```

Time root.	test.d1.s1
+ + + + + + + + + + + + + + + + + + +	100.0
2020-01-01T00:00:03.000+08:00	101.0
2020-01-01T00:00:04.000+08:00	102.0
2020-01-01T00:00:06.000+08:00	104.0
2020-01-01T00:00:08.000+08:00	126.0
2020-01-01T00:00:10.000+08:00	108.0
2020-01-01T00:00:14.000+08:00	112.0
2020-01-01T00:00:15.000+08:00	113.0
2020-01-01T00:00:16.000+08:00	114.0
2020-01-01T00:00:18.000+08:00	116.0
2020-01-01T00:00:20.000+08:00	118.0
2020-01-01T00:00:22.000+08:00	120.0
2020-01-01T00:00:26.000+08:00	124.0
2020-01-01T00:00:28.000+08:00	126.0
2020-01-01T00:00:30.000+08:00	NaN
2020-01-01T00:00:32.000+08:00	130.0
2020-01-01T00:00:34.000+08:00	132.0
2020-01-01T00:00:36.000+08:00	134.0
2020-01-01T00:00:38.000+08:00	136.0
2020-01-01T00:00:40.000+08:00	138.0
2020-01-01T00:00:42.000+08:00	140.0
2020-01-01T00:00:44.000+08:00	142.0
2020-01-01T00:00:46.000+08:00	144.0
2020-01-01T00:00:48.000+08:00	146.0
2020-01-01T00:00:50.000+08:00	148.0
2020-01-01T00:00:52.000+08:00	150.0
2020-01-01T00:00:54.000+08:00	152.0
2020-01-01T00:00:56.000+08:00	154.0
2020-01-01T00:00:58.000+08:00	156.0
2020-01-01T00:01:00.000+08:00	158.0
+	+

```
select Validity(s1, "window"="15") from root.test.d1 where time <= 2020-01-01 00:01:00
```



Chapter 4 Data Repairing

4.1 Fill(TODO)

4.2 TimestampRepair

This function is used for timestamp repair. According to the given standard time interval, the method of minimizing the repair cost is adopted. By fine-tuning the timestamps, the original data with unstable timestamp interval is repaired to strictly equispaced data. If no standard time interval is given, this function will use the **median**, **mode** or **cluster** of the time interval to estimate the standard time interval.

Name: TIMESTAMPREPAIR

Input Series: Only support a single input series. The data type is INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

- interval: The standard time interval whose unit is millisecond. It is a positive integer. By default, it will be estimated according to the given method.
- method: The method to estimate the standard time interval, which is 'median', 'mode' or 'cluster'. This parameter is only valid when interval is not given. By default, median will be used.

Output Series: Output a single series. The type is the same as the input. This series is the input after repairing.

4.2.1 Examples

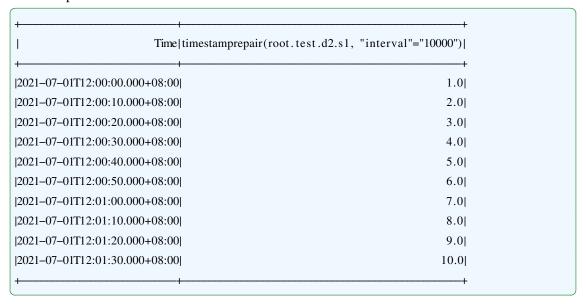
4.2.1.1 Manually Specify the Standard Time Interval

When interval is given, this function repairs according to the given standard time interval. Input series:

1	+
Time	root.test.d2.s1
+	+
2021-07-01T12:00:00.000+08:00	1.0
2021-07-01T12:00:10.000+08:00	2.0
2021-07-01T12:00:19.000+08:00	3.0
2021-07-01T12:00:30.000+08:00	4.0
2021-07-01T12:00:40.000+08:00	5.0
2021-07-01T12:00:50.000+08:00	6.0
2021-07-01T12:01:01.000+08:00	7.0
2021-07-01T12:01:11.000+08:00	8.0

```
select timestamprepair(s1, 'interval'='10000') from root.test.d2
```

Output series:

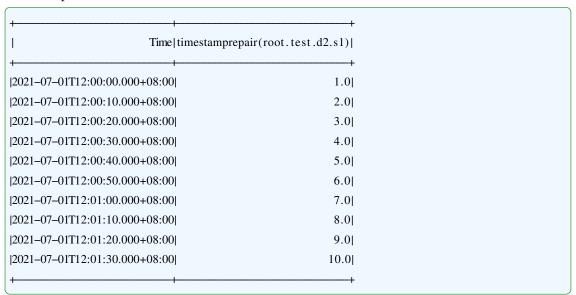


4.2.1.2 Automatically Estimate the Standard Time Interval

When interval is default, this function estimates the standard time interval.

Input series is the same as above, the SQL for query is shown below:

```
select timestamprepair(s1) from root.test.d2
```



4.3 ValueRepair

4.3.1 Usage

This function is used to repair the value of the time series. Currently, two methods are supported: **Screen** is a method based on speed threshold, which makes all speeds meet the threshold requirements under the premise of minimum changes; **LsGreedy** is a method based on speed change likelihood, which models speed changes as Gaussian distribution, and uses a greedy algorithm to maximize the likelihood.

Name: VALUEREPAIR

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

- method: The method used to repair, which is 'Screen' or 'LsGreedy'. By default, Screen is used.
- minSpeed: This parameter is only valid with Screen. It is the speed threshold. Speeds below it will be regarded as outliers. By default, it is the median minus 3 times of median absolute deviation.
- maxSpeed: This parameter is only valid with Screen. It is the speed threshold. Speeds above it will be regarded as outliers. By default, it is the median plus 3 times of median absolute deviation.
- center: This parameter is only valid with LsGreedy. It is the center of the Gaussian distribution of speed changes. By default, it is 0.
- sigma: This parameter is only valid with LsGreedy. It is the standard deviation of the Gaussian distribution of speed changes. By default, it is the median absolute deviation.

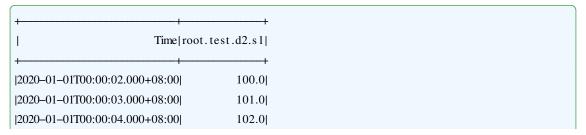
Output Series: Output a single series. The type is the same as the input. This series is the input after repairing.

Note: NaN will be filled with linear interpolation before repairing.

4.3.2 Examples

4.3.2.1 Repair with Screen

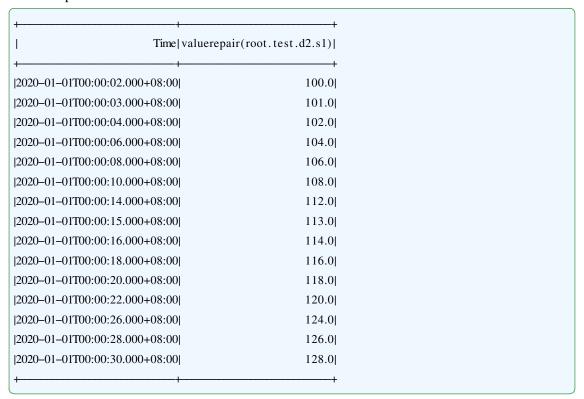
When method is 'Screen' or the default, Screen method is used. Input series:



```
|2020-01-01T00:00:06.000+08:00|
                                         104.0|
|2020-01-01T00:00:08.000+08:00|
                                         126.0
|2020-01-01T00:00:10.000+08:00|
                                         108.0
|2020-01-01T00:00:14.000+08:00|
                                         112.0
|2020-01-01T00:00:15.000+08:00|
                                         113.0
|2020-01-01T00:00:16.000+08:00|
                                         114.0
|2020-01-01T00:00:18.000+08:00|
                                         116.0
|2020-01-01T00:00:20.000+08:00|
                                         118.0
[2020-01-01T00:00:22.000+08:00]
                                         100.0
|2020-01-01T00:00:26.000+08:00|
                                         124.0
|2020-01-01T00:00:28.000+08:00|
                                         126.0
|2020-01-01T00:00:30.000+08:00|
                                           NaN|
```

```
select valuerepair(s1) from root.test.d2
```

Output series:



4.3.2.2 Repair with LsGreedy

When method is 'LsGreedy', LsGreedy method is used.

Input series is the same as above, the SQL for query is shown below:

```
select valuerepair(s1, 'method'='LsGreedy') from root.test.d2
```

Time valuerepair(root.test.c	d2.s1, "method"="LsGreedy")
020-01-01T00:00:02.000+08:00	100.0
020-01-01T00:00:03.000+08:00	101.0
020-01-01T00:00:04.000+08:00	102.0
020-01-01T00:00:06.000+08:00	104.0
020-01-01T00:00:08.000+08:00	106.0
020-01-01T00:00:10.000+08:00	108.0
020-01-01T00:00:14.000+08:00	112.0
020-01-01T00:00:15.000+08:00	113.0
020-01-01T00:00:16.000+08:00	114.0
020-01-01T00:00:18.000+08:00	116.0
020-01-01T00:00:20.000+08:00	118.0
020-01-01T00:00:22.000+08:00	120.0
020-01-01T00:00:26.000+08:00	124.0
020-01-01T00:00:28.000+08:00	126.0
020-01-01T00:00:30.000+08:00	128.0
	+

Chapter 5 Data Matching

5.1 Cov

5.1.1 Usage

This function is used to calculate the population covariance.

Name: COV

Input Series: Only support two input series. The types are both INT32 / INT64 / FLOAT / DOUBLE.

Output Series: Output a single series. The type is DOUBLE. There is only one data point in the series, whose timestamp is 0 and value is the population covariance.

Note:

- If a row contains missing points, null points or NaN, it will be ignored;
- If all rows are ignored, NaN will be output.

5.1.2 Examples

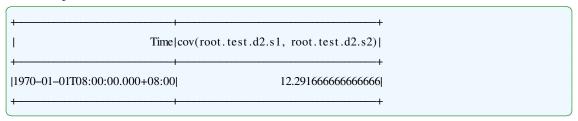
Input series:

```
Time | root.test.d2.s1 | root.test.d2.s2 |
|2020-01-01T00:00:02.000+08:00|
                                         100.0
                                                         101.0
|2020-01-01T00:00:03.000+08:00|
                                         101.0
                                                          null|
|2020-01-01T00:00:04.000+08:00|
                                         102.0
                                                         101.0
|2020-01-01T00:00:06.000+08:00|
                                         104.0
                                                         102.0
|2020-01-01T00:00:08.000+08:00|
                                         126.0
                                                         102.0
|2020-01-01T00:00:10.000+08:00|
                                         108.0
                                                         103.0
|2020-01-01T00:00:12.000+08:00|
                                          null|
                                                         103.0
|2020-01-01T00:00:14.000+08:00|
                                         112.0
                                                         104.0
|2020-01-01T00:00:15.000+08:00|
                                         113.0
                                                          null|
|2020-01-01T00:00:16.000+08:00|
                                         114.0
                                                         104.0
|2020-01-01T00:00:18.000+08:00|
                                         116.0
                                                         105.0
|2020-01-01T00:00:20.000+08:00|
                                         118.0
                                                         105.0
|2020-01-01T00:00:22.000+08:00|
                                         100.0
                                                         106.0
|2020-01-01T00:00:26.000+08:00|
                                         124.0
                                                         108.0
|2020-01-01T00:00:28.000+08:00|
                                         126.0
                                                         108.0
|2020-01-01T00:00:30.000+08:00|
                                           NaN|
                                                         108.0
```

SQL for query:

```
select cov(s1,s2) from root.test.d2
```

Output series:



5.2 CrossCorrelation

5.2.1 Usage

This function is used to calculate the cross correlation function of given two time series. For discrete time series, cross correlation is given by

$$CR(n) = \frac{1}{N} \sum_{m=1}^{N} S_1[m] S_2[m+n]$$

which represent the similarities between two series with different index shifts.

Name: CROSSCORRELATION

Input Series: Only support two input numeric series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Output Series: Output a single series with DOUBLE as datatype. There are 2N-1 data points in the series, the center of which represents the cross correlation calculated with prealigned series(that is CR(0) in the formula above), and the previous(or post) values represent those with shifting the latter series forward(or backward otherwise) until the two series are no longer overlapped(not included). In short, the values of output series are given by(index starts from 1)

$$OS[i] = CR(-N+i) = \frac{1}{N} \sum_{m=1}^{i} S_1[m]S_2[N-i+m], if i <= N$$

$$OS[i] = CR(i-N) = \frac{1}{N} \sum_{m=1}^{2N-i} S_1[i-N+m]S_2[m], if i > N$$

Note:

• null and NaN values in the input series will be ignored and treated as 0.

5.2.2 Examples

Input series:



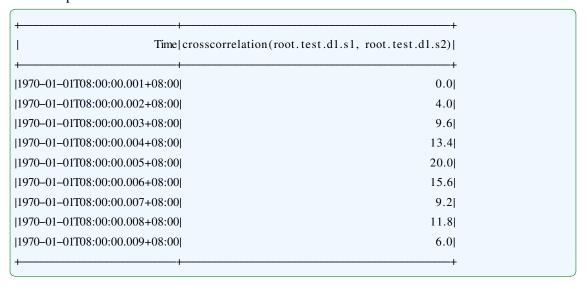
5.3 DTW(TODO)

2020-01-01T00	0:00:01.000+08:00	null	6
2020-01-01T00	0:00:02.000+08:00	2	7
2020-01-01T00	0:00:03.000+08:00	3	NaN
2020-01-01T00	0:00:04.000+08:00	4	9
2020-01-01T00	0:00:05.000+08:00	5	10
+	+	t	+

SQL for query:

```
select crosscorrelation(s1, s2) from root.test.d1 where time <= 2020-01-01 00:00:05
```

Output series:



5.2.2.1 examples on zeppelin

link: http://101.6.15.213:18181/#/notebook/2GETVW6AT

5.3 DTW(TODO)

5.4 PatternSymmetric

5.4.1 Usage

This function is used to find all symmetric subseries in the input whose degree of symmetry is less than the threshold. The degree of symmetry is calculated by DTW. The smaller the degree, the more symmetrical the series is.

Name: PATTERNSYMMETRIC

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE

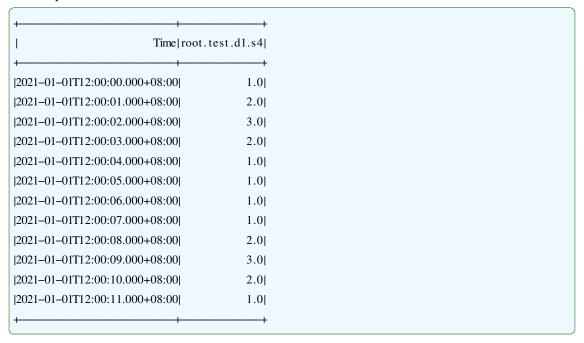
Parameter:

- window: The length of the symmetric subseries. It's a positive integer and the default value is 10.
- threshold: The threshold of the degree of symmetry. It's non-negative. Only the subseries whose degree of symmetry is below it will be output. By default, all subseries will be output.

Output Series: Output a single series. The type is DOUBLE. Each data point in the output series corresponds to a symmetric subseries. The output timestamp is the starting timestamp of the subseries and the output value is the degree of symmetry.

5.4.2 Example

Input series:



SQL for query:

```
select patternsymmetric(s4, 'window'='5', 'threshold'='0') from root.test.dl
```

5.5 Pearson

5.5.1 Usage

This function is used to calculate the Pearson Correlation Coefficient.

Name: PEARSON

Input Series: Only support two input series. The types are both INT32 / INT64 / FLOAT / DOUBLE.

Output Series: Output a single series. The type is DOUBLE. There is only one data point in the series, whose timestamp is 0 and value is the Pearson Correlation Coefficient.

Note:

- If a row contains missing points, null points or NaN, it will be ignored;
- If all rows are ignored, NaN will be output.

5.5.2 Examples

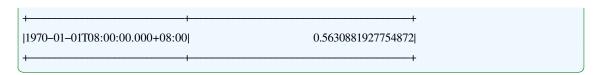
Input series:

```
Time | root.test.d2.s1 | root.test.d2.s2 |
|2020-01-01T00:00:02.000+08:00|
                                         100.0
                                                         101.0
|2020-01-01T00:00:03.000+08:00|
                                         101.0
                                                          null|
|2020-01-01T00:00:04.000+08:00|
                                         102.0
                                                         101.0
|2020-01-01T00:00:06.000+08:00|
                                         104.0
                                                         102.0
|2020-01-01T00:00:08.000+08:00|
                                         126.0
                                                         102.0
|2020-01-01T00:00:10.000+08:00|
                                         108.0
                                                         103.0
|2020-01-01T00:00:12.000+08:00|
                                          null|
                                                         103.0
|2020-01-01T00:00:14.000+08:00|
                                         112.0|
                                                         104.0
|2020-01-01T00:00:15.000+08:00|
                                         113.0
                                                          null|
|2020-01-01T00:00:16.000+08:00|
                                         114.0
                                                         104.0
|2020-01-01T00:00:18.000+08:00|
                                         116.0
                                                         105.0
|2020-01-01T00:00:20.000+08:00|
                                         118.0
                                                         105.0
|2020-01-01T00:00:22.000+08:00|
                                         100.0
                                                         106.0
|2020-01-01T00:00:26.000+08:00|
                                         124.0
                                                         108.0
|2020-01-01T00:00:28.000+08:00|
                                         126.0
                                                         108.0
|2020-01-01T00:00:30.000+08:00|
                                                         108.0
                                           NaN|
```

SQL for query:

```
select pearson(s1,s2) from root.test.d2
```

```
Time|pearson(root.test.d2.s1, root.test.d2.s2)|
```



5.6 SelfCorrelation

5.6.1 Usage

This function is used to calculate the self-correlation function of the input time series, which equals to cross correlation between the same series. For more information, please refer to CrossCorrelation function.

Name: SELFCORRELATION

Input Series: Only support a single input numeric series. The type is INT32 / INT64 / FLOAT / DOUBLE.

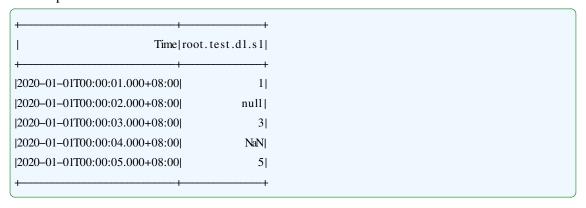
Output Series: Output a single series. The type is DOUBLE. There are 2N-1 data points in the series, and the values are interpreted in details in CrossCorrelation function.

Note:

• null and NaN values in the input series will be ignored and treated as 0.

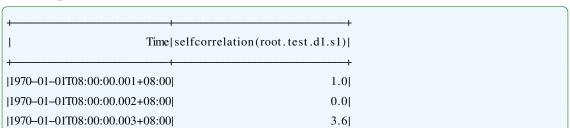
5.6.2 Examples

Input series:



SQL for query:

```
select selfcorrelation(s1) from root.test.d1 where time <= 2020-01-01 00:00:05
```



1970-01-01T08:00:00.004+08:00	0.0	
1970-01-01T08:00:00.005+08:00	7.0	
1970-01-01T08:00:00.006+08:00	0.0	
1970-01-01T08:00:00.007+08:00	3.6	
1970-01-01T08:00:00.008+08:00	0.0	
1970-01-01T08:00:00.009+08:00	1.0	
+	+	

5.6.2.1 examples on zeppelin

link: http://101.6.15.213:18181/#/notebook/2GC91M5DY

- 5.7 SeriesAlign(TODO)
- **5.8** SeriesSimilarity(TODO)
- 5.9 ValueAlign(TODO)

Chapter 6 Anomaly Detection

6.1 ADWIN

6.1.1 Usage

This function is used to detect possible concept drift of time series. According to delta parameter, the function judges if concept drift occurs at an input value using ADWIN algorithm, and a new time series of detected concept drifts will be output. For detail, see

```
Learning from Time-Changing Data with Adaptive Windowing, A Bifetet al., 2005
```

Name: ADWIN

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

- delta :a threshold to distinguish concept drifts. See the δ defined in the paper. Default value is 0.01.
- windowsize :the size of window to detect. Should be larger than 2
 Output Series: Output a single series. The type is INT32. 1 for anomaly, and 0 for others.

6.1.2 Examples

6.1.2.1 Assigning parameters

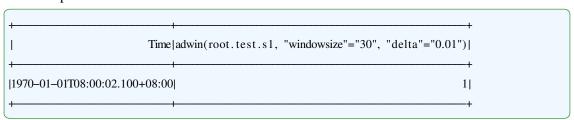
Input series:

```
Time | root.test.s1|
|1970-01-01T08:00:00.000+08:00|
                                        5.0
|1970-01-01T08:00:00.100+08:00|
                                        5.0
|1970-01-01T08:00:00.200+08:00|
                                        5.0|
|1970-01-01T08:00:00.300+08:00|
                                        5.0|
|1970-01-01T08:00:00.400+08:00|
                                        5.0
|1970-01-01T08:00:00.500+08:00|
                                        5.0
|1970-01-01T08:00:00.600+08:00|
                                        5.0
|1970-01-01T08:00:00.700+08:00|
                                        5.0
|1970-01-01T08:00:00.800+08:00|
                                        5.0
|1970-01-01T08:00:00.900+08:00|
                                        5.0
|1970-01-01T08:00:01.000+08:00|
                                        5.0
|1970-01-01T08:00:01.100+08:00|
                                        5.0
|1970-01-01T08:00:01.200+08:00|
                                        5.0
|1970-01-01T08:00:01.300+08:00|
                                        5.0
|1970-01-01T08:00:01.400+08:00|
                                        5.0
|1970-01-01T08:00:01.500+08:00|
                                        5.0
```

```
|1970-01-01T08:00:01.600+08:00|
                                        5.0|
|1970-01-01T08:00:01.700+08:00|
                                         5.0
|1970-01-01T08:00:01.800+08:00|
                                        5.0
|1970-01-01T08:00:01.900+08:00|
                                        5.0
|1970-01-01T08:00:02.000+08:00|
                                        10.0|
[1970-01-01T08:00:02.100+08:00]
                                        10.0|
|1970-01-01T08:00:02.200+08:00|
                                        10.0|
|1970-01-01T08:00:02.300+08:00|
                                        10.0|
|1970-01-01T08:00:02.400+08:00|
                                        10.0|
|1970-01-01T08:00:02.500+08:00|
                                        10.0
|1970-01-01T08:00:02.600+08:00|
                                        10.0|
|1970-01-01T08:00:02.700+08:00|
                                        10.0|
|1970-01-01T08:00:02.800+08:00|
                                        10.0|
|1970-01-01T08:00:02.900+08:00|
                                        10.0|
|1970-01-01T08:00:03.000+08:00|
                                        10.0|
|1970-01-01T08:00:03.100+08:00|
                                        10.0
[1970-01-01T08:00:03.200+08:00]
                                        10.0|
|1970-01-01T08:00:03.300+08:00|
                                        10.0|
|1970-01-01T08:00:03.400+08:00|
                                        10.0|
|1970-01-01T08:00:03.500+08:00|
                                        10.0|
|1970-01-01T08:00:03.600+08:00|
                                        10.0
|1970-01-01T08:00:03.700+08:00|
                                        10.0|
|1970-01-01T08:00:03.800+08:00|
                                        10.0
|1970-01-01T08:00:03.900+08:00|
                                        10.0|
```

```
select adwin(s1, "windowsize"="30", "delta"="0.01") from root.test
```

Output series:



6.2 KSigma

6.2.1 Usage

This function is used to detect anomalies based on the Dynamic K-Sigma Algorithm. Within a sliding window, the input value with a deviation of more than k times the standard deviation from the average will be output as anomaly.

Name: KSIGMA

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

- k: How many times to multiply on standard deviation to define anomaly, the default value is 3.
- window: The window size of Dynamic K-Sigma Algorithm, the default value is 10000.

 Output Series: Output a single series. The type is same as input series.

Note: Only when is larger than 0, the anomaly detection will be performed. Otherwise, nothing will be output.

6.2.2 Examples

6.2.2.1 Assigning k

Input series:

```
Time | root.test.d1.s1|
|2020-01-01T00:00:02.000+08:00|
                                           |0.0|
|2020-01-01T00:00:03.000+08:00|
                                          50.0|
|2020-01-01T00:00:04.000+08:00|
                                         100.0
|2020-01-01T00:00:06.000+08:00|
                                         150.0
|2020-01-01T00:00:08.000+08:00|
                                         200.0
[2020-01-01T00:00:10.000+08:00]
                                         200.0
|2020-01-01T00:00:14.000+08:00|
                                         200.0
|2020-01-01T00:00:15.000+08:00|
                                         200.0
|2020-01-01T00:00:16.000+08:00|
                                         200.0
|2020-01-01T00:00:18.000+08:00|
                                         200.0
|2020-01-01T00:00:20.000+08:00|
                                         150.0|
|2020-01-01T00:00:22.000+08:00|
                                         100.0
[2020-01-01T00:00:26.000+08:00]
                                          50.0
|2020-01-01T00:00:28.000+08:00|
                                           |0.0|
|2020-01-01T00:00:30.000+08:00|
                                           NaN
```

SQL for query:

```
select ksigma(s1,"k"="1.0") from root.test.d1 where time <= 2020-01-01 00:00:30
```

2020-01-01T00:00:28.000+08:00	0.0	
+	+	

6.3 LOF

6.3.1 Usage

This function is used to detect density anomaly of time series. According to k-th distance calculation parameter and local outlier factor (lof) threshold, the function judges if a set of input values is an density anomaly, and a bool mark of anomaly values will be output.

Name: LOF

Input Series: Multiple input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

- method :assign a detection method. The default value is "default", when input data has
 multiple dimensions. The alternative is "series", when a input series will be transformed
 to high dimension.
- k :use the k-th distance to calculate lof. Default value is 3.
- window: size of window to split origin data points. Default value is 10000.
- windowsize :dimension that will be transformed into when method is "series". The default value is 5.

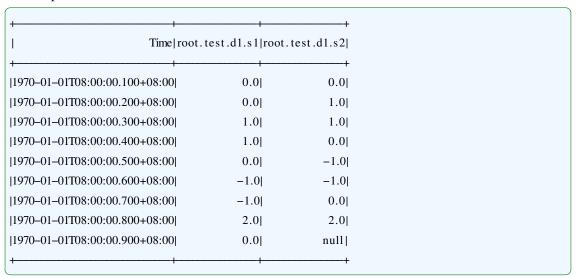
Output Series: Output a single series. The type is DOUBLE.

Note: Incomplete rows will be ignored. They are neither calculated nor marked as anomaly.

6.3.2 Examples

6.3.2.1 Using default parameters

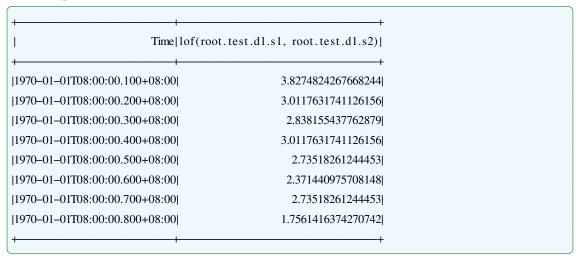
Input series:



SQL for query:

```
select lof(s1,s2) from root.test.dl where time<1000
```

Output series:



6.3.2.2 Diagnosing 1d timeseries

Input series:

```
Time | root.test.d1.s1|
|1970-01-01T08:00:00.100+08:00|
                                            1.0|
|1970-01-01T08:00:00.200+08:00|
                                           2.0
|1970-01-01T08:00:00.300+08:00|
                                           3.0
|1970-01-01T08:00:00.400+08:00|
                                           4.0|
|1970-01-01T08:00:00.500+08:00|
                                           5.0
|1970-01-01T08:00:00.600+08:00|
                                           6.0|
|1970-01-01T08:00:00.700+08:00|
                                           7.0|
|1970-01-01T08:00:00.800+08:00|
                                           8.0
|1970-01-01T08:00:00.900+08:00|
                                           9.0|
|1970-01-01T08:00:01.000+08:00|
                                           10.0
|1970-01-01T08:00:01.100+08:00|
                                          11.0|
|1970-01-01T08:00:01.200+08:00|
                                          12.0
|1970-01-01T08:00:01.300+08:00|
                                          13.0|
|1970-01-01T08:00:01.400+08:00|
                                          14.0
|1970-01-01T08:00:01.500+08:00|
                                          15.0
|1970-01-01T08:00:01.600+08:00|
                                           16.0|
|1970-01-01T08:00:01.700+08:00|
                                          17.0|
|1970-01-01T08:00:01.800+08:00|
                                          18.0|
|1970-01-01T08:00:01.900+08:00|
                                          19.0
|1970-01-01T08:00:02.000+08:00|
                                          20.0
```

SQL for query:

```
select lof(s1, "method"="series") from root.test.d1 where time<1000
```

Output series:

```
Time | lof(root.test.d1.s1) |
|1970-01-01T08:00:00.100+08:00|
                                  3.777777777778
|1970-01-01T08:00:00.200+08:00|
                                  4.327272727273
|1970-01-01T08:00:00.300+08:00|
                                  4.85714285714286
|1970-01-01T08:00:00.400+08:00|
                                  5.409090909090911
|1970-01-01T08:00:00.500+08:00|
                                  5.94999999999999
|1970-01-01T08:00:00.600+08:00|
                                  6.43243243243243
|1970-01-01T08:00:00.700+08:00|
                                  6.79999999999999
|1970-01-01T08:00:00.800+08:00|
                                               7.0|
|1970-01-01T08:00:00.900+08:00|
                                               7.0
|1970-01-01T08:00:01.000+08:00|
                                  6.79999999999999
|1970-01-01T08:00:01.100+08:00|
                                  6.43243243243243
|1970-01-01T08:00:01.200+08:00|
                                  5.94999999999999
|1970-01-01T08:00:01.300+08:00|
                                  5.40909090909091
|1970-01-01T08:00:01.400+08:00|
                                  4.85714285714286|
|1970-01-01T08:00:01.500+08:00|
                                  4.327272727273
|1970-01-01T08:00:01.600+08:00|
                                  3.77777777778
```

6.3.2.3

6.4 Range

6.4.1 Usage

This function is used to detect range anomaly of time series. According to upper bound and lower bound parameters, the function judges if a input value is beyond range, aka range anomaly, and a new time series of anomaly will be output.

Name: RANGE

Input Series: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

- lower_bound :lower bound of range anomaly detection.
- upper_bound :upper bound of range anomaly detection.

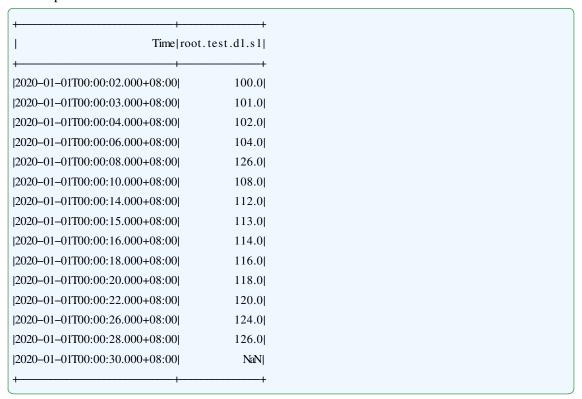
Output Series: Output a single series. The type is the same as the input.

Note: Only when upper_bound is larger than lower_bound, the anomaly detection will be performed. Otherwise, nothing will be output.

6.4.2 Examples

6.4.2.1 Assigning Lower and Upper Bound

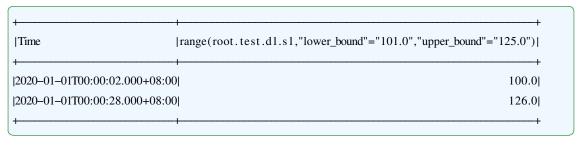
Input series:



SQL for query:

```
select\ range (s1,"lower_bound"="101.0","upper_bound"="125.0")\ from\ root.test.d1\ where\ time <= 2020-01-01\\00:00:30
```

Output series:



6.5 TwoSidedFilter

6.5.1 Usage

The function is used to filter anomalies of a numeric time series based on two-sided window detection.

Name: TWOSIDEDFILTER

Input Series: Only support a single input series. The data type is INT32 / INT64 / FLOAT / DOUBLE

Output Series: Output a single series. The type is the same as the input. It is the input without anomalies.

Parameter:

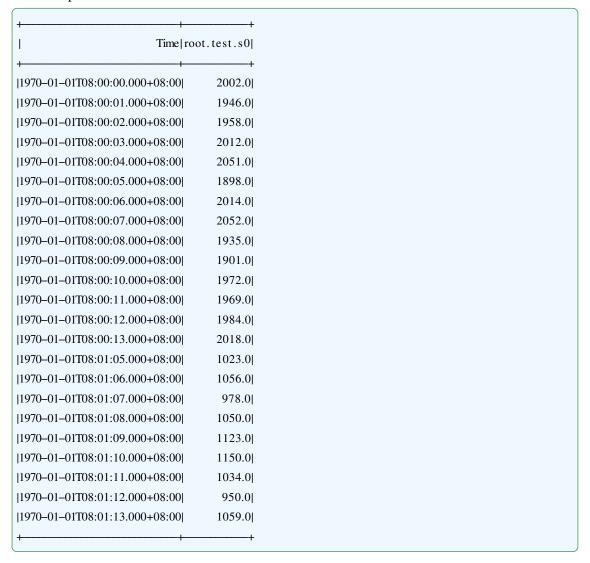
- len: The size of the windowwhich is a positible integer. By default, it's 5. When len = 3, the algorithm detects forward window and backward window with length 3 and calculates the outlierness of the current point.
- threshold: The threshold of outlierness, which is a floating number in (0,1). By default, it's 0.3. The strict standard of detecting anomalies is in proportion to the threshold.

6.5.2 Examples

Input series:

```
Time | root.test.s0|
|1970-01-01T08:00:00.000+08:00|
                                     2002.0|
|1970-01-01T08:00:01.000+08:00|
                                     1946.0
|1970-01-01T08:00:02.000+08:00|
                                     1958.0
|1970-01-01T08:00:03.000+08:00|
                                     2012.0
|1970-01-01T08:00:04.000+08:00|
                                     2051.0
|1970-01-01T08:00:05.000+08:00|
                                     1898.0
|1970-01-01T08:00:06.000+08:00|
                                     2014.0
|1970-01-01T08:00:07.000+08:00|
                                     2052.0|
|1970-01-01T08:00:08.000+08:00|
                                     1935.0
|1970-01-01T08:00:09.000+08:00|
                                     1901.0
|1970-01-01T08:00:10.000+08:00|
                                     1972.0|
|1970-01-01T08:00:11.000+08:00|
                                     1969.0
|1970-01-01T08:00:12.000+08:00|
                                     1984.0|
|1970-01-01T08:00:13.000+08:00|
                                     2018.0
|1970-01-01T08:00:37.000+08:00|
                                     1484.0
|1970-01-01T08:00:38.000+08:00|
                                     1055.0
|1970-01-01T08:00:39.000+08:00|
                                     1050.0
|1970-01-01T08:01:05.000+08:00|
                                     1023.0
|1970-01-01T08:01:06.000+08:00|
                                     1056.0|
|1970-01-01T08:01:07.000+08:00|
                                      978.0
|1970-01-01T08:01:08.000+08:00|
                                     1050.0
|1970-01-01T08:01:09.000+08:00|
                                     1123.0
|1970-01-01T08:01:10.000+08:00|
                                     1150.0
|1970-01-01T08:01:11.000+08:00|
                                     1034.0|
|1970-01-01T08:01:12.000+08:00|
                                      950.0|
|1970-01-01T08:01:13.000+08:00|
                                     1059.0
```

```
select TwoSidedFilter(s0, 'len'='5', 'threshold'='0.3') from root.test
```



Chapter 7 Frequency Domain

7.1 Conv(**TODO**)

7.1.1 Usage

This function is used to calculate the convolution.

Name: CONV

Input: Only support two input series. The types are both INT32 / INT64 / FLOAT / DOUBLE.

Output: Output a single series. The type is DOUBLE. It is the result of convolution whose timestamps starting from 0 only indicate the order.

Note: NaN in the input series will be ignored.

7.1.2 Examples

7.2 Deconv(TODO)

7.2.1 Usage

This function is used to calculate the deconvolution, i.e. polynomial division.

Name: DECONV

Input: Only support two input series. The types are both INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

• result The result of deconvolution, which is 'quotient' or 'remainder'. By default, the quotient will be output.

Output: Output a single series. The type is DOUBLE. It is the result of deconvolving the second series from the first series (dividing the first series by the second series) whose timestamps starting from 0 only indicate the order.

Note: NaN in the input series will be ignored.

7.2.2 Examples

7.3 FFT

7.3.1 Usage

This function is used to calculate the fast Fourier transform (FFT) of a numerical series.

Name: FFT

Input: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE. **Parameters:**

- type: The type of FFT, which is 'uniform' (by default) or 'nonuniform'. If the value is 'uniform', the timestamps will be ignored and all data points will be regarded as equidistant. Thus, the equidistant fast Fourier transform algorithm will be applied. If the value is 'nonuniform' (TODO), the non-equidistant fast Fourier transform algorithm will be applied based on timestamps.
- result: The result of FFT, which is 'real', 'imag', 'abs' or 'angle', corresponding to the real part, imaginary part, magnitude and phase angle. By default, the magnitude will be output.

Output: Output a single series. The type is DOUBLE. The length is the same as the input. The timestamps starting from 0 only indicate the order.

Note: NaN in the input series will be ignored.

7.3.2 Examples

7.3.2.1 Uniform FFT

With the default type, uniform FFT is applied. Input series:

+	+
Time r	oot.test.d1.s1
+	+
1970-01-01T08:00:00.000+08:00	2.902113
1970-01-01T08:00:01.000+08:00	1.1755705
1970-01-01T08:00:02.000+08:00	-2.1755705
1970-01-01T08:00:03.000+08:00	-1.9021131
1970-01-01T08:00:04.000+08:00	1.0
1970-01-01T08:00:05.000+08:00	1.9021131
1970-01-01T08:00:06.000+08:00	0.1755705
1970-01-01T08:00:07.000+08:00	-1.1755705
1970-01-01T08:00:08.000+08:00	-0.902113
1970-01-01T08:00:09.000+08:00	0.0
1970-01-01T08:00:10.000+08:00	0.902113
1970-01-01T08:00:11.000+08:00	1.1755705
1970-01-01T08:00:12.000+08:00	-0.1755705
1970-01-01T08:00:13.000+08:00	-1.9021131
1970-01-01T08:00:14.000+08:00	-1.0
1970-01-01T08:00:15.000+08:00	1.9021131
1970-01-01T08:00:16.000+08:00	2.1755705
1970-01-01T08:00:17.000+08:00	-1.1755705
1970-01-01T08:00:18.000+08:00	-2.902113
1970-01-01T08:00:19.000+08:00	0.0
+	+

```
select fft(s1) from root.test.d1
```

Output series:

```
Time | fft(root.test.d1.s1)|
|1970-01-01T08:00:00.000+08:00|
                                                 |0.0|
|1970-01-01T08:00:00.001+08:00| 1.2727111142703152E-8|
[1970-01-01T08:00:00.002+08:00] 2.385520799101839E-7[
[1970-01-01T08:00:00.003+08:00] 8.723291723972645E-8[
|1970-01-01T08:00:00.004+08:00| 19.999999960195904|
|1970-01-01T08:00:00.005+08:00|
                                   9.999999850988388|
[1970-01-01T08:00:00.006+08:00] 3.2260694930700566E-7[
[1970-01-01T08:00:00.007+08:00] 8.723291605373329E-8[
|1970-01-01T08:00:00.008+08:00| 1.108657103979944E-7|
|1970-01-01T08:00:00.009+08:00| 1.2727110997246171E-8|
|1970-01-01T08:00:00.010+08:00|1.9852334701272664E-23|
|1970-01-01T08:00:00.011+08:00| 1.2727111194499847E-8|
[1970-01-01T08:00:00.012+08:00] 1.108657103979944E-7[
|1970-01-01T08:00:00.013+08:00| 8.723291785769131E-8|
|1970-01-01T08:00:00.014+08:00| 3.226069493070057E-7|
|1970-01-01T08:00:00.015+08:00|
                                 9.999999850988388|
|1970-01-01T08:00:00.016+08:00|
                                19.999999960195904|
[1970-01-01T08:00:00.017+08:00] 8.723291747109068E-8[
|1970-01-01T08:00:00.018+08:00| 2.3855207991018386E-7|
[1970-01-01T08:00:00.019+08:00] 1.2727112069910878E-8[
```

Note: The input is $y = sin(2\pi t/4) + 2sin(2\pi t/5)$ with a length of 20. Thus, there are peaks in k=4 and k=5 of the output.

7.4 Filter(TODO)

7.4.1 Usage

This function is used to filter the input series by a rational transfer function, which is

$$Y(z) = \frac{b(0) + b(1)z^{-1} + \dots + b(n_b)z^{-n_b}}{a(0) + a(1)z^{-1} + \dots + a(n_a)z^{-n_a}}X(z)$$

Name: FILTER

Input: Only support a single input series. The type is INT32 / INT64 / FLOAT / DOUBLE.

Parameters:

- numerator: The numerator coefficients of the transfer function $a(0)...a(n_a)$. Coefficients are separated by commas.
- denominator : The denominator coefficients of the transfer function $b(0)..b(n_b)$. Coefficients are separated by commas.

Output: Output a single series. The type is DOUBLE. It is the filtered series whose length and timestamps are the same as the input.

Note: NaN in the input series will be ignored.

7.4.2 Examples

Chapter 8 Series Discovery

8.1 ConsecutiveSequences

8.1.1 Usage

This function is used to find locally longest consecutive subsequences in strictly equispaced multidimensional data.

Strictly equispaced data is the data whose time intervals are strictly equal. Missing data, including missing rows and missing values, is allowed in it, while data redundancy and timestamp drift is not allowed.

Consecutive subsequence is the subsequence that is strictly equispaced with the standard time interval without any missing data. If a consecutive subsequence is not a proper subsequence of any consecutive subsequence, it is locally longest.

Name: CONSECUTIVESEQUENCES

Input Series: Support multiple input series. The type is arbitrary but the data is strictly equispaced.

Parameters:

• gap: The standard time interval which is a positive number with an unit. The unit is 'ms' for millisecond, 's' for second, 'm' for minute, 'h' for hour and 'd' for day. By default, it will be estimated by the mode of time intervals.

Output Series: Output a single series. The type is INT32. Each data point in the output series corresponds to a locally longest consecutive subsequence. The output timestamp is the starting timestamp of the subsequence and the output value is the number of data points in the subsequence.

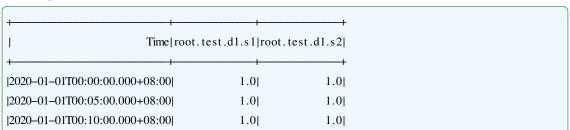
Note: For input series that is not strictly equispaced, there is no guarantee on the output.

8.1.2 Examples

8.1.2.1 Manually Specify the Standard Time Interval

It's able to manually specify the standard time interval by the parameter gap. It's notable that false parameter leads to false output.

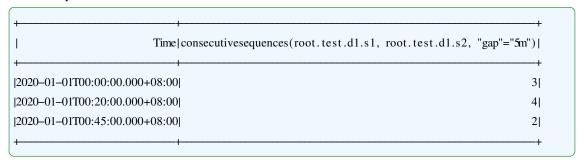
Input series:



2020-01-01T00:20:00.000+08:00	1.0	1.0
2020-01-01T00:25:00.000+08:00	1.0	1.0
2020-01-01T00:30:00.000+08:00	1.0	1.0
2020-01-01T00:35:00.000+08:00	1.0	1.0
2020-01-01T00:40:00.000+08:00	1.0	null
2020-01-01T00:45:00.000+08:00	1.0	1.0
2020-01-01T00:50:00.000+08:00	1.0	1.0
		+

```
select consecutivesequences(s1,s2,'gap'='5m') from root.test.dl
```

Output series:



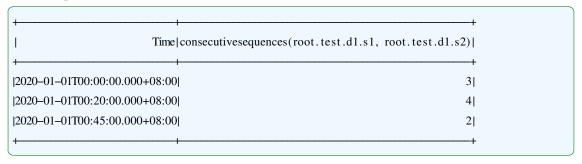
8.1.2.2 Automatically Estimate the Standard Time Interval

When gap is default, this function estimates the standard time interval by the mode of time intervals and gets the same results. Therefore, this usage is more recommended.

Input series is the same as above, the SQL for query is shown below:

```
select consecutivesequences(s1,s2) from root.test.d1
```

Output series:



8.2 ConsecutiveWindows

8.2.1 Usage

This function is used to find consecutive windows of specified length in strictly equispaced multidimensional data.

Strictly equispaced data is the data whose time intervals are strictly equal. Missing data, including missing rows and missing values, is allowed in it, while data redundancy and timestamp drift is not allowed.

Consecutive window is the subsequence that is strictly equispaced with the standard time interval without any missing data.

Name: CONSECUTIVEWINDOWS

Input Series: Support multiple input series. The type is arbitrary but the data is strictly equispaced.

Parameters:

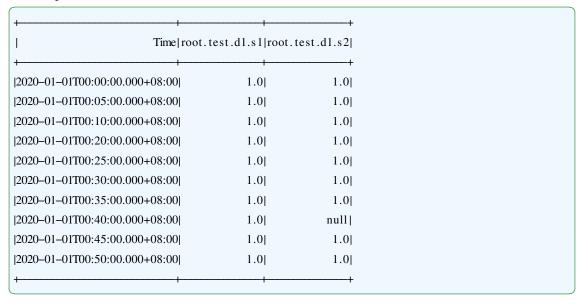
- gap: The standard time interval which is a positive number with an unit. The unit is 'ms' for millisecond, 's' for second, 'm' for minute, 'h' for hour and 'd' for day. By default, it will be estimated by the mode of time intervals.
- length: The length of the window which is a positive number with an unit. The unit is 'ms' for millisecond, 's' for second, 'm' for minute, 'h' for hour and 'd' for day. This parameter cannot be lacked.

Output Series: Output a single series. The type is INT32. Each data point in the output series corresponds to a consecutive window. The output timestamp is the starting timestamp of the window and the output value is the number of data points in the window.

Note: For input series that is not strictly equispaced, there is no guarantee on the output.

8.2.2 Examples

Input series:



SQL for query:

```
select consecutivewindows(s1,s2,'length'='10m') from root.test.d1
```

Chapter 9 Complex Event Processing

- **9.1 AND**(**TODO**)
- **9.2** EventMatching(TODO)
- **9.3** EventNameRepair(TODO)
- 9.4 EventTag(TODO)
- **9.5** EventTimeRepair(TODO)
- **9.6** MissingEventRecovery(TODO)
- **9.7 SEQ**(**TODO**)