User Manual of IoTDB-Quality

Author: Data Quality Group

Institute: School of Software, Tsinghua University

Date: July 2, 2021

Contents

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

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	Univeral functions of 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	

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:

```
+-----------------------------+---------------+
|                         Time|root.test.d2.s2|
+-----------------------------+---------------+
|2020-01-01T08:00:00.001+08:00|          Hello|
|2020-01-01T08:00:00.002+08:00|          hello|
|2020-01-01T08:00:00.003+08:00|          Hello|
|2020-01-01T08:00:00.004+08:00|          World|
|2020-01-01T08:00:00.005+08:00|          World|
+-----------------------------+---------------+
```

SQL for query:

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

Output series:

```
+-----------------------------+-------------------------+
|                         Time|distinct(root.test.d2.s2)|
+-----------------------------+-------------------------+
|1970-01-01T08:00:00.001+08:00|                    Hello|
|1970-01-01T08:00:00.002+08:00|                    hello|
|1970-01-01T08:00:00.003+08:00|                    World|
+-----------------------------+-------------------------+
```

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

Input series:

```
+-----------------------------+------------+
|                         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|
|2020-01-01T00:00:13.000+08:00|          14.0|
|2020-01-01T00:00:14.000+08:00|          15.0|
|2020-01-01T00:00:15.000+08:00|          16.0|
|2020-01-01T00:00:16.000+08:00|          17.0|
|2020-01-01T00:00:17.000+08:00|          18.0|
|2020-01-01T00:00:18.000+08:00|          19.0|
|2020-01-01T00:00:19.000+08:00|          20.0|
+-----------------------------+--------------+
```

SQL for query:

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

Output series:

```
+-----------------------------+---------------------------------------------------------------+
|                         Time|histogram(root.test.d1.s1, "start"="1", "end"="20", "count"="10")|
+-----------------------------+---------------------------------------------------------------+
|1970-01-01T08:00:00.000+08:00|                                                              2|
|1970-01-01T08:00:00.001+08:00|                                                              2|
|1970-01-01T08:00:00.002+08:00|                                                              2|
|1970-01-01T08:00:00.003+08:00|                                                              2|
|1970-01-01T08:00:00.004+08:00|                                                              2|
|1970-01-01T08:00:00.005+08:00|                                                              2|
|1970-01-01T08:00:00.006+08:00|                                                              2|
|1970-01-01T08:00:00.007+08:00|                                                              2|
|1970-01-01T08:00:00.008+08:00|                                                              2|
|1970-01-01T08:00:00.009+08:00|                                                              2|
+-----------------------------+---------------------------------------------------------------+
```

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|              1|
|2020-01-01T00:00:02.000+08:00|              2|
|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|              7|
|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:

```
+-----------------------------+-------------------------+
|                         Time|integral(root.test.d1.s1)|
+-----------------------------+-------------------------+
|1970-01-01T08:00:00.000+08:00|                     57.5|
+-----------------------------+-------------------------+
```

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:

```
select integral(s1, "unit"="1m") from root.test.d1 where time <= 2020-01-01 00:00:10
```

Output series:

```
+-----------------------------+-------------------------+
|                         Time|integral(root.test.d1.s1)|
+-----------------------------+-------------------------+
|1970-01-01T08:00:00.000+08:00|                    0.958|
+-----------------------------+-------------------------+
```

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|
|2021-03-17T10:32:27.054+08:00|   0.7307929|
|2021-03-17T10:32:28.054+08:00|   2.3167255|
|2021-03-17T10:32:29.054+08:00|    2.342443|
|2021-03-17T10:32:30.054+08:00|   1.5809103|
|2021-03-17T10:32:31.054+08:00|   1.4829416|
|2021-03-17T10:32:32.054+08:00|   1.5800357|
|2021-03-17T10:32:33.054+08:00|   0.7124368|
|2021-03-17T10:32:34.054+08:00| -0.78597564|
|2021-03-17T10:32:35.054+08:00|   1.2058644|
|2021-03-17T10:32:36.054+08:00|   1.4215064|
|2021-03-17T10:32:37.054+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
```

Output series:

```
+-----------------------------+------------------+
|                         Time| mad(root.test.s0)|
+-----------------------------+------------------+
|1970-01-01T08:00:00.000+08:00|0.6806197166442871|
+-----------------------------+-----------------+
```

2.4.2.2 Approximate Query

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

SQL for query:

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

Output series:

```
+-----------------------------+---------------------------------+
|                         Time|mad(root.test.s0, "error"="0.01")|
+-----------------------------+---------------------------------+
|1970-01-01T08:00:00.000+08:00|               0.6806616245859518|
+-----------------------------+---------------------------------+
```

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 0.49~0.51. 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

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|
|2021-03-17T10:32:27.054+08:00|   0.7307929|
|2021-03-17T10:32:28.054+08:00|   2.3167255|
|2021-03-17T10:32:29.054+08:00|    2.342443|
|2021-03-17T10:32:30.054+08:00|   1.5809103|
|2021-03-17T10:32:31.054+08:00|   1.4829416|
|2021-03-17T10:32:32.054+08:00|   1.5800357|
|2021-03-17T10:32:33.054+08:00|   0.7124368|
|2021-03-17T10:32:34.054+08:00| -0.78597564|
|2021-03-17T10:32:35.054+08:00|   1.2058644|
|2021-03-17T10:32:36.054+08:00|   1.4215064|
|2021-03-17T10:32:37.054+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 median(s0, "error"="0.01") from root.test
```

Output series:

```
+-----------------------------+------------------------------------+
|                         Time|median(root.test.s0, "error"="0.01")|
+-----------------------------+------------------------------------+
|1970-01-01T08:00:00.000+08:00|                   1.021884560585022|
+-----------------------------+------------------------------------+
```

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:

```
+-----------------------------+------------+
|                         Time|root.test.d2.s2|
+-----------------------------+------------+
|1970-01-01T08:00:00.001+08:00|       Hello|
|1970-01-01T08:00:00.002+08:00|       hello|
|1970-01-01T08:00:00.003+08:00|       Hello|
|1970-01-01T08:00:00.004+08:00|       World|
|1970-01-01T08:00:00.005+08:00|       World|
|1970-01-01T08:00:01.600+08:00|       World|
|1970-01-15T09:37:34.451+08:00|       Hello|
|1970-01-15T09:37:34.452+08:00|       hello|
|1970-01-15T09:37:34.453+08:00|       Hello|
|1970-01-15T09:37:34.454+08:00|       World|
|1970-01-15T09:37:34.455+08:00|       World|
+-----------------------------+------------+
```

SQL for query:

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

Output series:

```
+-----------------------------+---------------------+
|                         Time|mode(root.test.d2.s2)|
+-----------------------------+---------------------+
|1970-01-01T08:00:00.000+08:00|                World|
+-----------------------------+---------------------+
```

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

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|
|2021-03-17T10:32:27.054+08:00|   0.7307929|
|2021-03-17T10:32:28.054+08:00|   2.3167255|
|2021-03-17T10:32:29.054+08:00|    2.342443|
|2021-03-17T10:32:30.054+08:00|   1.5809103|
|2021-03-17T10:32:31.054+08:00|   1.4829416|
|2021-03-17T10:32:32.054+08:00|   1.5800357|
|2021-03-17T10:32:33.054+08:00|   0.7124368|
|2021-03-17T10:32:34.054+08:00| -0.78597564|
|2021-03-17T10:32:35.054+08:00|   1.2058644|
|2021-03-17T10:32:36.054+08:00|   1.4215064|
|2021-03-17T10:32:37.054+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 percentile(s0, "rank"="0.2", "error"="0.01") from root.test
```

Output series:

```
+-----------------------------+------------------------------------------------------+
|                         Time|percentile(root.test.s0, "rank"="0.2", "error"="0.01")|
+-----------------------------+------------------------------------------------------+
|1970-01-01T08:00:00.000+08:00|                                    0.1801469624042511|
+-----------------------------+------------------------------------------------------+
```

2.8 Resample

2.8.1 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.

2.8.2 Examples

2.8.2.1 Up-sampling

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

Input series:

```
+-----------------------------+---------------+
|                         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|
+-----------------------------+---------------+
```

SQL for query:

```
select resample(s1,'every'='5m','interp'='linear') from root.test.d1
```

Output series:

```
+-----------------------------+----------------------------------------------------------+
|                         Time|resample(root.test.d1.s1, "every"="5m", "interp"="linear")|
+-----------------------------+----------------------------------------------------------+
|2021-03-06T16:00:00.000+08:00|                                        3.0899999141693115|
|2021-03-06T16:05:00.000+08:00|                                        3.2366665999094644|
|2021-03-06T16:10:00.000+08:00|                                        3.3833332856496177|
|2021-03-06T16:15:00.000+08:00|                                        3.5299999713897705|
|2021-03-06T16:20:00.000+08:00|                                        3.5199999809265137|
|2021-03-06T16:25:00.000+08:00|                                         3.509999990463257|
|2021-03-06T16:30:00.000+08:00|                                                       3.5|
|2021-03-06T16:35:00.000+08:00|                                         3.503333330154419|
|2021-03-06T16:40:00.000+08:00|                                         3.506666660308838|
|2021-03-06T16:45:00.000+08:00|                                         3.509999990463257|
|2021-03-06T16:50:00.000+08:00|                                        3.4766666889190674|
|2021-03-06T16:55:00.000+08:00|                                         3.443333387374878|
|2021-03-06T17:00:00.000+08:00|                                        3.4100000858306885|
+-----------------------------+----------------------------------------------------------+
```

2.8.2.2 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.d1
```

Output series:

```
+-----------------------------+-----------------------------------------------------------+
|                         Time|resample(root.test.d1.s1, "every"="30m", "aggr"="first")|
+-----------------------------+-----------------------------------------------------------+
|2021-03-06T16:00:00.000+08:00|                                         3.0899999141693115|
|2021-03-06T16:30:00.000+08:00|                                                        3.5|
|2021-03-06T17:00:00.000+08:00|                                         3.4100000858306885|
+-----------------------------+-----------------------------------------------------------+
```

2.8.2.3 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:

```sql
select resample(s1,'every'='30m','start'='2021-03-06 15:00:00') from root.test.d1
```

Output series:

```
+-----------------------------+-----------------------------------------------------------------------+
|                         Time|resample(root.test.d1.s1, "every"="30m", "start"="2021-03-06 15:00:00")|
+-----------------------------+-----------------------------------------------------------------------+
|2021-03-06T15:00:00.000+08:00|                                                                    NaN|
|2021-03-06T15:30:00.000+08:00|                                                                    NaN|
|2021-03-06T16:00:00.000+08:00|                                                      3.309999942779541|
|2021-03-06T16:30:00.000+08:00|                                                     3.5049999952316284|
|2021-03-06T17:00:00.000+08:00|                                                     3.4100000858306885|
+-----------------------------+-----------------------------------------------------------------------+
```

2.9 Sample

2.9.1 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.

2.9.2 Examples

2.9.2.1 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:

```sql
select sample(s1,'method'='reservoir','k'='5') from root.test.d1
```

Output series:

```
+-----------------------------+------------------------------------------------------+
|                         Time|sample(root.test.d1.s1, "method"="reservoir", "k"="5")|
+-----------------------------+------------------------------------------------------+
|2020-01-01T00:00:02.000+08:00|                                                   2.0|
|2020-01-01T00:00:03.000+08:00|                                                   3.0|
|2020-01-01T00:00:05.000+08:00|                                                   5.0|
|2020-01-01T00:00:08.000+08:00|                                                   8.0|
|2020-01-01T00:00:10.000+08:00|                                                  10.0|
+-----------------------------+------------------------------------------------------+
```

2.9.2.2 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.d1
```

Output series:

```
+-----------------------------+------------------------------------------------------+
|                         Time|sample(root.test.d1.s1, "method"="isometric", "k"="5")|
+-----------------------------+------------------------------------------------------+
|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|
+-----------------------------+------------------------------------------------------+
```

2.10 Skew

2.10.1 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.

2.10.2 Examples

Input series:

```
+-----------------------------+---------------+
|                         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:

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

Output series:

```
+-----------------------------+-----------------------+
|                         Time|  skew(root.test.d1.s1)|
+-----------------------------+-----------------------+
|1970-01-01T08:00:00.000+08:00|    -0.9998427402292644|
+-----------------------------+-----------------------+
```

2.11 Spread

2.11.1 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.

2.11.2 Examples

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|
+-----------------------------+-------------+
```

SQL for query:

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

Output series:

```
+-----------------------------+-----------------------+
|                         Time|spread(root.test.d1.s1)|
+-----------------------------+-----------------------+
|1970-01-01T08:00:00.000+08:00|                   26.0|
+-----------------------------+-----------------------+
```

2.12 Stddev

2.12.1 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.

2.12.2 Examples

Input series:

```
+-----------------------------+-------------+
|                         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|
|2020-01-01T00:00:13.000+08:00|          14.0|
|2020-01-01T00:00:14.000+08:00|          15.0|
|2020-01-01T00:00:15.000+08:00|          16.0|
|2020-01-01T00:00:16.000+08:00|          17.0|
|2020-01-01T00:00:17.000+08:00|          18.0|
|2020-01-01T00:00:18.000+08:00|          19.0|
|2020-01-01T00:00:19.000+08:00|          20.0|
+-----------------------------+--------------+
```

SQL for query:

```sql
select stddev(s1) from root.test.d1
```

Output series:

```
+-----------------------------+-----------------------+
|                         Time|stddev(root.test.d1.s1)|
+-----------------------------+-----------------------+
|1970-01-01T08:00:00.000+08:00|     5.7662812973353965|
+-----------------------------+-----------------------+
```

2.13 TimeWeightedAvg

2.13.1 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.

2.13.2 Examples

Input series:

```
+-----------------------------+------------+
|                         Time|root.test.d1.s1|
+-----------------------------+------------+
|2020-01-01T00:00:01.000+08:00|           1|
|2020-01-01T00:00:02.000+08:00|           2|
|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|           7|
|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:

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

Output series:

```
+-----------------------------+---------------------------------+
|                         Time|timeweightedavg(root.test.d1.s1)|
+-----------------------------+---------------------------------+
|1970-01-01T08:00:00.000+08:00|                             5.75|
+-----------------------------+---------------------------------+
```

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 number of data points in each window. The number of data points in the last window may be less than it. 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.

3.1.2 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|
|2020-01-01T00:00:08.000+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|
+-----------------------------+---------------+
```

SQL for query:

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

Output series:

```
+-----------------------------+-----------------------------+
|                         Time|completeness(root.test.d1.s1)|
+-----------------------------+-----------------------------+
|2020-01-01T00:00:02.000+08:00|                        0.875|
+-----------------------------+-----------------------------+
```

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

Output series:

```
+-----------------------------+--------------------------------------------+
|                         Time|completeness(root.test.d1.s1, "window"="15")|
+-----------------------------+--------------------------------------------+
|2020-01-01T00:00:02.000+08:00|                                       0.875|
|2020-01-01T00:00:32.000+08:00|                                         1.0|
+-----------------------------+--------------------------------------------+
```

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 number of data points in each window. The number of data points in the last window may be less than it. 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:

```
+-----------------------------+-------------+
|                         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|
+-----------------------------+-------------+
```

SQL for query:

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

Output series:

```
+-----------------------------+--------------------------+
|                         Time|consistency(root.test.d1.s1)|
+-----------------------------+--------------------------+
|2020-01-01T00:00:02.000+08:00|        0.9333333333333333|
+-----------------------------+--------------------------+
```

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:

```
+-----------------------------+-------------+
|                         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
```

Output series:

```
+-----------------------------+-----------------------------------------+
|                         Time|consistency(root.test.d1.s1, "window"="15")|
+-----------------------------+-----------------------------------------+
|2020-01-01T00:00:02.000+08:00|                       0.9333333333333333|
|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 number of data points in each window. The number of data points in the last window may be less than it. 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:

```
+-----------------------------+------------+
|                         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|
+-----------------------------+------------+
```

SQL for query:

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

Output series:

```
+-----------------------------+--------------------------+
|                         Time|timeliness(root.test.d1.s1)|
+-----------------------------+--------------------------+
|2020-01-01T00:00:02.000+08:00|        0.9333333333333333|
+-----------------------------+-----------------------------+
```

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|
|2020-01-01T00:00:40.000+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 timeliness(s1,"window"="15") from root.test.d1 where time <= 2020-01-01 00:01:00
```

Output series:

```
+-----------------------------+------------------------------------------+
|                         Time|timeliness(root.test.d1.s1, "window"="15")|
+-----------------------------+------------------------------------------+
|2020-01-01T00:00:02.000+08:00|                        0.9333333333333333|
|2020-01-01T00:00:32.000+08:00|                                       1.0|
+-----------------------------+------------------------------------------+
```

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 number of data points in each window. The number of data points in the last window may be less than it. 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:

```
+-----------------------------+---------------+
|                         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|
+-----------------------------+---------------+
```

SQL for query:

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

Output series:

```
+-----------------------------+-------------------------+
|                         Time|validity(root.test.d1.s1)|
+-----------------------------+-------------------------+
|2020-01-01T00:00:02.000+08:00|       0.8833333333333333|
+-----------------------------+-------------------------+
```

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:

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

Output series:

```
+-----------------------------+----------------------------------------+
|                         Time|validity(root.test.d1.s1, "window"="15")|
+-----------------------------+----------------------------------------+
|2020-01-01T00:00:02.000+08:00|                      0.8833333333333333|
|2020-01-01T00:00:32.000+08:00|                                     1.0|
+-----------------------------+----------------------------------------+
```

Chapter 4 Data Repairing

4.1 Fill(TODO)

4.2 TimestampRepair(TODO)

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:

```
+-----------------------------+---------------+
|                         Time|root.test.d2.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|          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|
+-----------------------------+---------------+
```

SQL for query:

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

Output series:

```
+-----------------------------+----------------------------+
|                         Time|valuerepair(root.test.d2.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|                       106.0|
|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|                      128.0|
+-----------------------------+---------------------------+
```

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
```

Output series:

```
+-----------------------------+-------------------------------------------------+
|                         Time|valuerepair(root.test.d2.s1, "method"="LsGreedy")|
+-----------------------------+-------------------------------------------------+
|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|                                            106.0|
|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|                                            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:

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

Output series:

```
+-----------------------------+-------------------------------------+
|                         Time|cov(root.test.d2.s1, root.test.d2.s2)|
+-----------------------------+-------------------------------------+
|1970-01-01T08:00:00.000+08:00|                   12.291666666666666|
+-----------------------------+-------------------------------------+
```

5.2 DTW(TODO)

5.3 Pearson

5.3.1 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.

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

Output series:

```
+-----------------------------+-------------------------------------+
|                         Time|pearson(root.test.d2.s1, root.test.d2.s2)|
+-----------------------------+-------------------------------------+
|1970-01-01T08:00:00.000+08:00|                   0.5630881927754872|
+-----------------------------+-------------------------------------+
```

5.4 SeriesAlign(TODO)

5.5 SeriesSimilarity(TODO)

5.6 ValueAlign(TODO)

Chapter 6 Anomaly Detection

6.1 KSigma

6.1.1 Usage

This function is used to detect distribution anomaly of time series. According to k parameter, the function judges if a input value is an extreme value beyond k-sigma, aka distribution anomaly, and a new time series of anomaly will be output.

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 extreme value.

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.

6.1.2 Examples

6.1.2.1 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
```

Output series:

```
+-----------------------------+---------------------------------+
|Time                         |ksigma(root.test.d1.s1,"k"="3.0")|
+-----------------------------+---------------------------------+
|2020-01-01T00:00:02.000+08:00|                              0.0|
|2020-01-01T00:00:03.000+08:00|                             50.0|
|2020-01-01T00:00:26.000+08:00|                             50.0|
|2020-01-01T00:00:28.000+08:00|                              0.0|
+-----------------------------+---------------------------------+
```

6.2 LOF

6.2.1 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.

6.2.2 Examples

6.2.2.1 Using default parameters

Input series:

```
+-----------------------------+---------------+---------------+
|                         Time|root.test.d1.s1|root.test.d1.s2|
+-----------------------------+---------------+---------------+
|1970-01-01T08:00:00.100+08:00|            0.0|            0.0|
|1970-01-01T08:00:00.200+08:00|            0.0|            1.0|
|1970-01-01T08:00:00.300+08:00|           1.0|           1.0|
|1970-01-01T08:00:00.400+08:00|           1.0|           0.0|
|1970-01-01T08:00:00.500+08:00|           0.0|          -1.0|
|1970-01-01T08:00:00.600+08:00|          -1.0|          -1.0|
|1970-01-01T08:00:00.700+08:00|          -1.0|           0.0|
|1970-01-01T08:00:00.800+08:00|           2.0|           2.0|
|1970-01-01T08:00:00.900+08:00|           0.0|          null|
+-----------------------------+--------------+--------------+
```

SQL for query:

```
select lof(s1,s2) from root.test.d1 where time<1000
```

Output series:

```
+-----------------------------+-------------------------------------+
|                         Time|lof(root.test.d1.s1, root.test.d1.s2)|
+-----------------------------+-------------------------------------+
|1970-01-01T08:00:00.100+08:00|                   3.8274824267668244|
|1970-01-01T08:00:00.200+08:00|                   3.0117631741126156|
|1970-01-01T08:00:00.300+08:00|                    2.838155437762879|
|1970-01-01T08:00:00.400+08:00|                   3.0117631741126156|
|1970-01-01T08:00:00.500+08:00|                     2.73518261244453|
|1970-01-01T08:00:00.600+08:00|                    2.371440975708148|
|1970-01-01T08:00:00.700+08:00|                     2.73518261244453|
|1970-01-01T08:00:00.800+08:00|                   1.7561416374270742|
+-----------------------------+-------------------------------------+
```

6.2.2.2 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|    3.77777777777778|
|1970-01-01T08:00:00.200+08:00|    4.32727272727273|
|1970-01-01T08:00:00.300+08:00|    4.85714285714286|
|1970-01-01T08:00:00.400+08:00|    5.40909090909091|
|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|    4.32727272727273|
|1970-01-01T08:00:01.600+08:00|    3.77777777777778|
+-----------------------------+--------------------+
```

6.2.2.3

6.3 Range

6.3.1 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.

6.3.2 Examples

6.3.2.1 Assigning Lower and Upper Bound

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|
+-----------------------------+------------+
```

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:

```
+-----------------------------+------------------------------------------------------------------+
|Time                         |range(root.test.d1.s1,"lower_bound"="101.0","upper_bound"="125.0")|
+-----------------------------+------------------------------------------------------------------+
|2020-01-01T00:00:02.000+08:00|                                                             100.0|
|2020-01-01T00:00:28.000+08:00|                                                             126.0|
+-----------------------------+------------------------------------------------------------------+
```

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(TODO)

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', 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 timestamps starting from 0 only indicate the order.

Note: `NaN` in the input series will be ignored.

7.3.2 Examples

7.4 Filter(TODO)

7.4.1 Usage

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

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

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:

```
+-----------------------------+---------------+---------------+
|                         Time|root.test.d1.s1|root.test.d1.s2|
+-----------------------------+---------------+---------------+
|2020-01-01T00:00:00.000+08:00|            1.0|            1.0|
|2020-01-01T00:05:00.000+08:00|            1.0|            1.0|
|2020-01-01T00:10:00.000+08:00|            1.0|            1.0|
|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|
+-----------------------------+-------------+-------------+
```

SQL for query:

```sql
select consecutivesequences(s1,s2,'gap'='5m') from root.test.d1
```

Output series:

```
+-----------------------------+------------------------------------------------------------------+
|                         Time|consecutivesequences(root.test.d1.s1, root.test.d1.s2, "gap"="5m")|
+-----------------------------+------------------------------------------------------------------+
|2020-01-01T00:00:00.000+08:00|                                                                 3|
|2020-01-01T00:20:00.000+08:00|                                                                 4|
|2020-01-01T00:45:00.000+08:00|                                                                 2|
+-----------------------------+------------------------------------------------------------------+
```

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:

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

Output series:

```
+-----------------------------+------------------------------------------------------+
|                         Time|consecutivesequences(root.test.d1.s1, root.test.d1.s2)|
+-----------------------------+------------------------------------------------------+
|2020-01-01T00:00:00.000+08:00|                                                     3|
|2020-01-01T00:20:00.000+08:00|                                                     4|
|2020-01-01T00:45:00.000+08:00|                                                     2|
+-----------------------------+------------------------------------------------------+
```

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:

```
+-----------------------------+---------------+---------------+
|                         Time|root.test.d1.s1|root.test.d1.s2|
+-----------------------------+---------------+---------------+
|2020-01-01T00:00:00.000+08:00|            1.0|            1.0|
|2020-01-01T00:05:00.000+08:00|            1.0|            1.0|
|2020-01-01T00:10:00.000+08:00|            1.0|            1.0|
|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|
+-----------------------------+---------------+---------------+
```

SQL for query:

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

Output series:

```
+-----------------------------+------------------------------------------------------------------+
|                         Time|consecutivewindows(root.test.d1.s1, root.test.d1.s2, "length"="10m")|
+-----------------------------+------------------------------------------------------------------+
|2020-01-01T00:00:00.000+08:00|                                                                 3|
|2020-01-01T00:20:00.000+08:00|                                                                 3|
|2020-01-01T00:25:00.000+08:00|                                                                 3|
+-----------------------------+------------------------------------------------------------------+
```

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)