User Manual of IoTDB-Quality

Author: Data Quality Group

Institute: School of Software, Tsinghua University

Date: March 26, 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.

1.2 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()
Mean	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()
Cov	
Histogram	
Pearson	
Skew	

1.3 Q&A

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 without the guarantee on the order.

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

2.3 Integral

2.4 Mad

2.5 Median

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 one that appears first will be output.

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

2.8 Sample

2.9 Skew

2.10 Spread

2.10.1 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: NaN in the input series will be ignored.

2.10.2 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|
+-----------------------------+-----------------------+
```

2.11 Stddev

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:

```sql
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:

```sql
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:

```
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

4.2 TimestampRepair

4.3 ValueRepair

Chapter 5 Data Matching

5.1 Cov

5.2 DTW

5.3 Pearson

5.4 SeriesAlign

5.5 SeriesSimilarity

5.6 ValueAlign

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 DOUBLE.

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.

- k :use the k-th distance to calculate lof.
- threshold :sets of values of lof larger than this threshold will be recognized as outlier. Lof larger than 1 indicates density near the set of value is below nearby sets, which is likely to be an outlier.

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

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

6.2.2 Examples

6.2.2.1 Assigning k

6.2.2.2 Assigning k and threshold

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 DOUBLE.

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 Complex Event Processing

7.1 AND

7.2 EventMatching

7.3 EventNameRepair

7.4 EventTag

7.5 EventTimeRepair

7.6 MissingEventRecovery

7.7 SEQ