IoTDB-Quality 用户文档

作者：数据质量组

组织：清华大学软件学院

时间：2021 年 4 月 2 日

目录

第 1 章 开始

1.1 概述

1.1.1 什么是 IoTDB-Quality

Apache IoTDB (Internet of Things Database) 是一个时序数据的数据管理系统，可以为用户提供数据收集、存储和分析等特定的服务。

对基于时序数据的应用而言，数据质量至关重要。**IoTDB-Quality** 基于 IoTDB 用户自定义函数 (UDF)，实现了一系列关于数据质量的函数，包括数据画像、数据质量评估与修复等，有效满足了工业领域对数据质量的需求。

1.1.2 快速开始

1. 下载包含全部依赖的 jar 包和注册脚本；
2. 将 jar 包复制到 IoTDB 程序目录的 `ext\udf` 目录下；
3. 运行 `sbin\start-server.bat`（在 Windows 下）或 `sbin\start-server.sh`（在 Linux 或 MacOS 下）以启动 IoTDB 服务器；
4. 将注册脚本复制到 IoTDB 的程序目录下，并运行注册脚本以注册 UDF。

1.2 系统对标

1.2.1 InfluxDB

InfluxDB是一个流行的时序数据库。InfluxQL 是它的查询语言，其部分通用函数与数据画像相关。这些函数与 IoTDB-Quality 数据画像函数的对比如下（*Native* 指该函数已经作为 IoTDB 的 Native 函数实现，*Built-in UDF* 指该函数已经作为 IoTDB 的内建 UDF 函数实现）：

IoTDB-Quality 的数据画像函数	InfluxQL 的通用函数
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 常见问题

第 2 章 数据画像

2.1 Distinct

2.1.1 函数简介

本函数可以返回输入序列中出现的所有不同的值。

函数名： DISTINCT

输入序列：仅支持单个输入序列，类型可以是任意的

输出序列：输出单个序列，类型与输入相同。

提示：输出序列的时间戳是无意义的，且不保证输出顺序。

2.1.2 使用示例

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

```
+-----------------------------+------------------------+
|                         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 函数简介

本函数用于计算单列数值型数据的分布直方图。

函数名： HISTOGRAM

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE。

参数：

- `start`：表示所求数据范围的下限，默认值为-Double.MAX_VALUE。
- `end`：表示所求数据范围的上限，默认值为 Double.MAX_VALUE，`start`的值必须小于或等于`end`。
- `count`: 表示直方图分桶的数量，默认值为 1，其值必须为正整数。

输出序列：直方图分桶的值，其中第 i 个桶（从 1 开始计数）表示的数据范围下界为 start+(i-1)(end-start)/count，数据范围上界为 start+i(end-start)/count。

提示：如果某个数据点的数值小于`start`，它会被放入第 1 个桶；如果某个数据点的数值大于`end`，它会被放入最后 1 个桶。

2.2.2 使用示例

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

2.4 Mad

2.4.1 函数简介

本函数用于计算单列数值型数据的近似或精确绝对中位差，绝对中位差为所有数值与其中位数绝对偏移量的中位数。

如有数据集 {1,3,3,5,5,6,7,8,9}，其中位数为 5，所有数值与中位数的偏移量的绝对值为 {0,0,1,2,2,2,3,4,4}，其中位数为 2，故而原数据集的绝对中位差为 2。

函数名： MAD

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE。

参数：

- `error`：近似绝对中位差的基于数值的误差百分比，取值范围为 [0,1]，默认值为 0。如当 `error` =0.01 时，记精确绝对中位差为 a，近似绝对中位差为 b，不等式 0.99a <= b <= 1.01a 成立。当 `error` =0 时，计算结果为精确绝对中位差。

输出序列：近似绝对中位差

2.4.2 使用示例

2.4.2.1 精确查询

当 `error` 参数缺省或为 0 时，本函数计算精确绝对中位差。

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

```
+-----------------------------+------------------+
|                         Time| mad(root.test.s0)|
+-----------------------------+------------------+
|2021-03-17T10:32:17.054+08:00|0.6806197166442871|
+-----------------------------+------------------+
```

2.4.2.2 近似查询

当 error 参数取值不为 0 时，本函数计算近似绝对中位差。

输入序列同上，用于查询的 SQL 语句如下：

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

输出序列：

```
+-----------------------------+--------------------------------+
|                         Time|mad(root.test.s0, "error"="0.01")|
+-----------------------------+--------------------------------+
|2021-03-17T10:32:17.054+08:00|              0.6806616245859518|
+-----------------------------+--------------------------------+
```

2.5 Median

2.5.1 函数简介

本函数用于计算单列数值型数据的近似中位数。

函数名： MEDIAN

输入序列： 仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE

参数：

- `error`：近似中位数的基于排名的误差百分比，如`error`=0.01，则计算出的中位数的真实排名百分比在 0.49~0.51 之间。

输出序列： 近似中位数

2.5.2 使用示例

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

```
+----------------------------+-----------------------------------+
|                        Time|median(root.test.s0, "error"="0.01")|
+----------------------------+-----------------------------------+
|2021-03-17T10:32:17.054+08:00|                1.021884560585022|
+----------------------------+-----------------------------------+
```

2.6 Mode

2.6.1 函数简介

本函数用于计算时间序列的众数，即出现次数最多的元素。

函数名： MODE

输入序列：仅支持单个输入序列，类型可以是任意的。

输出序列：输出单个序列，类型与输入相同，序列仅包含一个时间戳为 0、值为众数的数据点。

提示：如果有多个出现次数最多的元素，将会输出最先出现的一个。

2.6.2 使用示例

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

2.7 Percentile

2.7.1 函数简介

本函数用于计算单列数值型数据的近似分位数。

函数名： PERCENTILE

输入序列： 仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE。

参数：

- `rank`：[0,1] 范围内浮点数，代表所求分位数在所有数据中的排名百分比，如当设为 0.5 时则计算中位数，默认值为 0.5。
- `error`：(0,1) 范围内浮点数，代表近似分位数的基于排名的误差百分比，如 `rank` =0.5 且 `error` =0.01，则计算出的分位数的真实排名百分比在 0.49~0.51 之间，默认值为 0.01。

输出序列： 近似分位数

2.7.2 使用示例

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

```
+-----------------------------+-----------------------------------------------------+
|                         Time|percentile(root.test.s0, "rank"="0.2", "error"="0.01")|
+-----------------------------+-----------------------------------------------------+
|2021-03-17T13:18:56.054+08:00|                                 0.1801469624042511|
+-----------------------------+-----------------------------------------------------+
```

2.8 Sample

2.8.1 函数简介

本函数对输入序列进行采样，即从输入序列中选取指定数量的数据点并输出。目前，本函数支持两种采样方法：**蓄水池采样法（reservoir sampling）**对数据进行随机采样，所有数据点被采样的概率相同；**等距采样法（isometric sampling）**按照相等的索引间隔对数据进行采样。

函数名： SAMPLE

输入序列：仅支持单个输入序列，类型可以是任意的。

参数：

- `method`：采样方法，取值为'isometric' 或'reservoir'。在缺省情况下，采用等距采样法。
- `k`：采样数，它是一个正整数，在缺省情况下为 1。

输出序列：输出单个序列，类型与输入序列相同。该序列的长度为采样数，序列中的每一个数据点都来自于输入序列。

提示：如果采样数大于序列长度，那么输入序列中所有的数据点都会被输出。

2.8.2 使用示例

2.8.2.1 等距采样

当 `method` 参数为'isometric' 或缺省时，采用等距采样法对输入序列进行采样。

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

```
+-----------------------------+-------------------------------------+
|                         Time|sample(root.test.d1.s1, "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|
+-----------------------------+-------------------------------------+
```

2.8.2.2 蓄水池采样

当 method 参数为'reservoir' 时，采用蓄水池采样法对输入序列进行采样。由于该采样方法具有随机性，下面展示的输出序列只是一种可能的结果。

输入序列同上，用于查询的 SQL 语句如下：

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

输出序列：

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

2.9.1 函数简介

本函数用于计算单列数值型数据的总体偏度

函数名： SKEW

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE

输出序列：总体偏度

2.9.2 使用示例

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

2.10 Spread

2.10.1 函数简介

本函数用于计算时间序列的极差，即最大值减去最小值的结果。

函数名： SPREAD

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE。

输出序列：输出单个序列，类型与输入相同，序列仅包含一个时间戳为 0、值为极差的数据点。

提示：输入序列中的 NaN 将被忽略。

2.10.2 使用示例

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

2.11 Stddev

2.11.1 函数简介

本函数用于计算单列数值型数据的总体标准差。

函数名： STDDEV

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE。

输出序列：输出单个序列，类型为 DOUBLE。序列仅包含一个时间戳为 0、值为总体标准差的数据点。

2.11.2 使用示例

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

第 3 章 数据质量

3.1 Completeness

3.1.1 函数简介

本函数用于计算时间序列的完整性。将输入序列划分为若干个连续且不重叠的窗口，分别计算每一个窗口的完整性，并输出窗口第一个数据点的时间戳和窗口的完整性。

函数名： COMPLETENESS

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE。

参数：

- window：每一个窗口包含的数据点数目（一个大于 0 的整数），最后一个窗口的数据点数目可能会不足。缺省情况下，全部输入数据都属于同一个窗口。

输出序列：输出单个序列，类型为 DOUBLE，其中每一个数据点的值的范围都是[0,1]。

提示：只有当窗口内的数据点数目超过 10 时，才会进行完整性计算。否则，该窗口将被忽略，不做任何输出。

3.1.2 使用示例

3.1.2.1 参数缺省

在参数缺省的情况下，本函数将会把全部输入数据都作为同一个窗口计算完整性。

输入序列：

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

```
+-----------------------------+-------------+
```

用于查询的 SQL 语句：

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

输出序列：

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

3.1.2.2 指定窗口大小

在指定窗口大小的情况下，本函数会把输入数据划分为若干个窗口计算完整性。

输入序列:

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

用于查询的 SQL 语句：

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

输出序列：

```
+-----------------------------+---------------------------------------------+
|                         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 函数简介

本函数用于计算时间序列的一致性。将输入序列划分为若干个连续且不重叠的窗口，分别计算每一个窗口的一致性，并输出窗口第一个数据点的时间戳和窗口的时效性。

函数名： CONSISTENCY

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE

参数：

- `window`：每一个窗口包含的数据点数目（一个大于 0 的整数），最后一个窗口的数据点数目可能会不足。缺省情况下，全部输入数据都属于同一个窗口。

输出序列：输出单个序列，类型为 DOUBLE，其中每一个数据点的值的范围都是 [0,1]。

提示：只有当窗口内的数据点数目超过 10 时，才会进行一致性计算。否则，该窗口将被忽略，不做任何输出。

3.2.2 使用示例

3.2.2.1 参数缺省

在参数缺省的情况下，本函数将会把全部输入数据都作为同一个窗口计算一致性。

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

3.2.2.2 指定窗口大小

在指定窗口大小的情况下，本函数会把输入数据划分为若干个窗口计算一致性。

输入序列:

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

用于查询的 SQL 语句：

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

输出序列：

```
+-----------------------------+-----------------------------------------------+
|                         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 函数简介

本函数用于计算时间序列的时效性。将输入序列划分为若干个连续且不重叠的窗口，分别计算每一个窗口的时效性，并输出窗口第一个数据点的时间戳和窗口的时效性。

函数名： TIMELINESS

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE

参数：

- `window`：每一个窗口包含的数据点数目（一个大于 0 的整数），最后一个窗口的数据点数目可能会不足。缺省情况下，全部输入数据都属于同一个窗口。

输出序列：输出单个序列，类型为 DOUBLE，其中每一个数据点的值的范围都是[0,1]。

提示：只有当窗口内的数据点数目超过 10 时，才会进行时效性计算。否则，该窗口将被忽略，不做任何输出。

3.3.2 使用示例

3.3.2.1 参数缺省

在参数缺省的情况下，本函数将会把全部输入数据都作为同一个窗口计算时效性。

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

3.3.2.2 指定窗口大小

在指定窗口大小的情况下，本函数会把输入数据划分为若干个窗口计算时效性。

输入序列:

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

用于查询的 SQL 语句：

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

输出序列：

```
+-----------------------------+-----------------------------------------------+
|                         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 函数简介

本函数用于计算时间序列的有效性。将输入序列划分为若干个连续且不重叠的窗口，分别计算每一个窗口的有效性，并输出窗口第一个数据点的时间戳和窗口的有效性。

函数名： VALIDITY

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE

参数：

- `window`：每一个窗口包含的数据点数目（一个大于 0 的整数），最后一个窗口的数据点数目可能会不足。缺省情况下，全部输入数据都属于同一个窗口。

输出序列：输出单个序列，类型为 DOUBLE，其中每一个数据点的值的范围都是[0,1]。

提示：只有当窗口内的数据点数目超过 10 时，才会进行有效性计算。否则，该窗口将被忽略，不做任何输出。

3.4.2 使用示例

3.4.2.1 参数缺省

在参数缺省的情况下，本函数将会把全部输入数据都作为同一个窗口计算有效性。

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

3.4.2.2 指定窗口大小

在指定窗口大小的情况下，本函数会把输入数据划分为若干个窗口计算有效性。

输入序列:

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

用于查询的 SQL 语句：

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

输出序列：

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

第 4 章 数据修复

4.1 Fill

4.1.1 函数简介

函数名： FILL

输入序列：支持多维输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE

参数：

- method: “mean" 指使用均值方法；“median” 使用中值填补；"previous" 指使用前值方法；“MICE" 使用 multivariate imputation of chained equation 方法填补；” ARIMA “使用回归滑动平均方法（默认）；"KNN" 使用 K 近邻方法；“EM” 使用期望最大化方法；
- regression: 当 method 指定为 mice 时使用，“lr” /"linear" 表示线性回归，“rf" 指随机森林；其他方式待完成中

输出序列：即修复后的多维序列。

4.2 TimestampRepair

4.2.1 函数简介

本函数用于时间戳的等间隔修复。将根据提供的参考时间间隔 k，采用最小化修复代价修复成等间隔的时间序列；不给定的话，根据全局间隔中位数确定时间间隔。

函数名： TIMESTAMPREPAIR

输入序列：仅支持单个输入序列，数据类型无要求

参数：

- k: 参考时间间隔，可选。

输出序列：输出单个序列。

4.3 ValueRepair

4.3.1 函数简介

本函数用于对时间序列的数值进行修复。目前，本函数支持两种修复方法：**Screen** 是一种基于速度阈值的方法，在最小改动的前提下使得所有的速度符合阈值要求；**LsGreedy** 是一种基于速度变化似然的方法，将速度变化建模为高斯分布，并采用贪心算法极大化似然函数。

函数名： VALUEREPAIR

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE。

参数：

- method：修复时采用的方法，取值为'Screen' 或'LsGreedy'。在缺省情况下，使用 Screen 方法进行修复。
- minSpeed：该参数仅在使用 Screen 方法时有效。当速度小于该值时会被视作数值异常点加以修复。在缺省情况下为中位数减去三倍绝对中位差。
- maxSpeed：该参数仅在使用 Screen 方法时有效。当速度大于该值时会被视作数值异常点加以修复。在缺省情况下为中位数加上三倍绝对中位差。
- center：该参数仅在使用 LsGreedy 方法时有效。对速度变化分布建立的高斯模型的中心。在缺省情况下为 0。
- sigma：该参数仅在使用 LsGreedy 方法时有效。对速度变化分布建立的高斯模型的标准差。在缺省情况下为绝对中位差。

输出序列：输出单个序列，类型与输入序列相同。该序列是修复后的输入序列。

提示： NaN 在修复之前会先进行线性插值填补。

4.3.2 使用示例

4.3.2.1 使用 Screen 方法进行修复

4.3.2.2 使用 LsGreedy 方法进行修复

第 5 章 数据匹配

5.1 Cov

5.2 DTW

5.3 Pearson

5.4 SeriesAlign

5.5 SeriesSimilarity

5.6 ValueAlign

第 6 章 异常检测

6.1 KSigma

6.1.1 函数简介

本函数用于查找时间序列的 k 倍标准差分布异常。将根据提供的 k，判断输入数据是否为超过 k-sigma 的极端分布，即分布异常，并输出所有异常点为新的时间序列。

函数名： KSIGMA

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE

参数：

- `k`: 确定极端分布时标准差 sigma 的倍数。

输出序列：输出单个序列，类型为 DOUBLE。

提示： k 应大于 0，否则将不做输出。

6.1.2 使用示例

6.1.2.1 指定 k

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

```
+-----------------------------+-----------------------------------------+
|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 函数简介

本函数使用局部离群点检测方法用于查找序列的密度异常。将根据提供的第 k 距离数及局部离群点因子 (lof) 阈值，判断输入数据是否为离群点，即异常，并输出各点的判别结果。

函数名： LOF

输入序列：多个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE

参数：

- `k`: 使用第 k 距离计算局部离群点因子。
- `threshold`: 判断输入数据为局部离群点的局部离群点因子的下界，默认为 1。局部离群点因子大于 1 表明其密度低于附近点，更可能为离群点。

输出序列：输出单个序列，类型为 BOOLEAN。

提示：不完整的数据行会被忽略，不参与计算，也不标记为离群点。

6.2.2 使用示例

6.2.2.1 指定第 k 距离数

6.2.2.2 指定第 k 距离数与局部离群点因子阈值

6.3 Range

6.3.1 函数简介

本函数用于查找时间序列的范围异常。将根据提供的上界与下界，判断输入数据是否越界，即异常，并输出所有异常点为新的时间序列。

函数名： RANGE

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE

参数：

- `lower_bound`: 范围异常检测的下界。

- upper_bound : 范围异常检测的上界。

输出序列：输出单个序列，类型为 DOUBLE。

提示：应满足给定上界大于下界，否则将不做输出。

6.3.2 使用示例

6.3.2.1 指定上界与下界

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

第 7 章 复杂事件处理

7.1 AND

7.1.1 函数简介

本函数用于查询时间序列中具有并行关系的模式匹配，并输出匹配的个数。

函数名： SEQ

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE

参数：

- `WITHIN`：匹配的时间序列的时间间隔的最大值。
- `ATLEAST`：匹配的时间序列的时间间隔的最小值。

输出序列：输出匹配的个数，类型为 INT32。

7.1.2 使用示例

TODO

7.2 EventMatching

7.3 EventNameRepair

7.4 EventTag

7.5 EventTimeRepair

7.6 MissingEventRecovery

7.7 SEQ

7.7.1 函数简介

本函数用于查询时间序列中具有顺序关系的模式匹配，并输出匹配的个数。

函数名： SEQ

输入序列：仅支持单个输入序列，类型为 INT32 / INT64 / FLOAT / DOUBLE

参数：

- `WITHIN`：匹配的时间序列的时间间隔的最大值。
- `ATLEAST`：匹配的时间序列的时间间隔的最小值。

输出序列：输出匹配的个数，类型为 INT32。

7.7.2 使用示例

TODO