数据画像

1.1 ACF

1.1.1 函数简介

本函数用于计算时间序列的自相关函数值，即序列与自身之间的互相关函数，详情参见 XCorr 函数文档。

函数名： ACF

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。序列中共包含 $2N-1$ 个数据点，每个值的具体含义参见 XCorr 函数文档。

提示：

- 序列中的 NaN 值会被忽略，在计算中表现为 0。

1.1.2 使用示例

输入序列：

```
+-----------------------------+-------------+
|                         Time|root.test.d1.s1|
+-----------------------------+-------------+
|2020-01-01T00:00:01.000+08:00|            1|
|2020-01-01T00:00:02.000+08:00|          NaN|
|2020-01-01T00:00:03.000+08:00|            3|
|2020-01-01T00:00:04.000+08:00|          NaN|
|2020-01-01T00:00:05.000+08:00|            5|
+-----------------------------+-------------+
```

用于查询的 SQL 语句：

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

输出序列：

```
+-----------------------------+--------------------+
|                         Time|acf(root.test.d1.s1)|
+-----------------------------+--------------------+
|1970-01-01T08:00:00.001+08:00|                 1.0|
|1970-01-01T08:00:00.002+08:00|                 0.0|
|1970-01-01T08:00:00.003+08:00|                 3.6|
|1970-01-01T08:00:00.004+08:00|                 0.0|
|1970-01-01T08:00:00.005+08:00|                 7.0|
|1970-01-01T08:00:00.006+08:00|                 0.0|
|1970-01-01T08:00:00.007+08:00|                 3.6|
```

```
|1970-01-01T08:00:00.008+08:00|                   0.0|
|1970-01-01T08:00:00.009+08:00|                   1.0|
+-----------------------------+----------------------+
```

1.1.2.1 Zeppelin 示例

链接: <http://101.6.15.213:18181/#/notebook/2GC91M5DY>

1.2 Distinct

1.2.1 函数简介

本函数可以返回输入序列中出现的所有不同的元素。

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。输出顺序是任意的。
- 缺失值和空值将被忽略，但 NaN 不会被忽略。
- 字符串区分大小写

1.2.2 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|
+-----------------------------+----------------------------+
```

1.3 Histogram

1.3.1 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：

- `min`：表示所求数据范围的下限，默认值为-Double.MAX_VALUE。
- `max`：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 $min + (i-1) \cdot \frac{max-min}{count}$，数据范围上界为 $min + i \cdot \frac{max-min}{count}$。

提示：

- 如果某个数据点的数值小于 `min`，它会被放入第 1 个桶；如果某个数据点的数值大于 `max`，它会被放入最后 1 个桶。
- 数据中的空值、缺失值和 `NaN` 将会被忽略。

1.3.2 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,"min"="1","max"="20","count"="10") from root.test.d1
```

输出序列：

```
+-----------------------------+-----------------------------------------------------------------------+
|                         Time|histogram(root.test.d1.s1, "min"="1", "max"="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|
+-----------------------------+-----------------------------------------------------------------------+
```

1.4 Integral

1.4.1 函数简介

本函数用于计算时间序列的数值积分，即以时间为横坐标、数值为纵坐标绘制的折线图中折线以下的面积。

函数名： INTEGRAL

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

参数：

- `unit`：积分求解所用的时间轴单位，取值为"1S", "1s", "1m", "1H", "1d"（区分大小写），分别表示以毫秒、秒、分钟、小时、天为单位计算积分。

缺省情况下取"1s"，以秒为单位。

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、值为积分结果的数据点。

提示：

- 积分值等于折线图中每相邻两个数据点和时间轴形成的直角梯形的面积之和，不同时间单位下相当于横轴进行不同倍数放缩，得到的积分值可直接按放缩倍数转换。
- 数据中 NaN 将会被忽略。折线将以临近两个有值数据点为准。

1.4.2 使用示例

1.4.2.1 参数缺省

缺省情况下积分以 1s 为时间单位。

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

其计算公式为：

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

1.4.2.2 指定时间单位

指定以分钟为时间单位。

输入序列同上，用于查询的 SQL 语句如下：

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

输出序列：

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

其计算公式为：

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

1.5 Mad

1.5.1 函数简介

本函数用于计算单列数值型数据的精确或近似绝对中位差，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 $0.99a \leq b \leq 1.01a$ 成立。当 `error` =0 时，计算结果为精确绝对中位差。

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、值为绝对中位差的数据点。

提示： 数据中的空值、缺失值和 `NaN` 将会被忽略。

1.5.2 使用示例

1.5.2.1 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)|
+-----------------------------+-----------------+
|1970-01-01T08:00:00.000+08:00|0.6806197166442871|
+-----------------------------+-----------------+
```

1.5.2.2 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")|
```

```
+-----------------------------+--------------------------------+
|1970-01-01T08:00:00.000+08:00|              0.6806616245859518|
+-----------------------------+--------------------------------+
```

1.6 Median

1.6.1 函数简介

本函数用于计算单列数值型数据的精确或近似中位数。中位数是顺序排列的乙组数据中居于中间位置的数。

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，取值范围 [0,1)，默认值为 0。如当 `error` =0.01 时，计算出的中位数的真实排名百分比在 0.49~0.51 之间。当 `error` =0 时，计算结果为精确中位数。

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、值为中位数的数据点。

1.6.2 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")|
+-----------------------------+-----------------------------------+
|1970-01-01T08:00:00.000+08:00|                 1.021884560585022|
+-----------------------------+-----------------------------------+
```

1.7 MinMax

1.7.1 函数简介

本函数将输入序列使用 min-max 方法进行标准化。最小值归一至 0，最大值归一至 1.

函数名： MINMAX

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

参数：

- `compute`：若设置为"batch"，则将数据全部读入后转换；若设置为"stream"，则需用户提供最大值及最小值进行流式计算转换。默认为"batch"。
- `min`：使用流式计算时的最小值。
- `max`：使用流式计算时的最大值。

输出序列： 输出单个序列，类型为 DOUBLE。

1.7.2 使用示例

1.7.2.1 全数据计算

输入序列：

```
+-----------------------------+------------+
|                         Time|root.test.s1|
```

```
+-----------------------------+------------+
|1970-01-01T08:00:00.100+08:00|         0.0|
|1970-01-01T08:00:00.200+08:00|         0.0|
|1970-01-01T08:00:00.300+08:00|         1.0|
|1970-01-01T08:00:00.400+08:00|        -1.0|
|1970-01-01T08:00:00.500+08:00|         0.0|
|1970-01-01T08:00:00.600+08:00|         0.0|
|1970-01-01T08:00:00.700+08:00|        -2.0|
|1970-01-01T08:00:00.800+08:00|         2.0|
|1970-01-01T08:00:00.900+08:00|         0.0|
|1970-01-01T08:00:01.000+08:00|         0.0|
|1970-01-01T08:00:01.100+08:00|         1.0|
|1970-01-01T08:00:01.200+08:00|        -1.0|
|1970-01-01T08:00:01.300+08:00|        -1.0|
|1970-01-01T08:00:01.400+08:00|         1.0|
|1970-01-01T08:00:01.500+08:00|         0.0|
|1970-01-01T08:00:01.600+08:00|         0.0|
|1970-01-01T08:00:01.700+08:00|        10.0|
|1970-01-01T08:00:01.800+08:00|         2.0|
|1970-01-01T08:00:01.900+08:00|        -2.0|
|1970-01-01T08:00:02.000+08:00|         0.0|
+-----------------------------+------------+
```

用于查询的 SQL 语句:

```
select minmax(s1) from root.test
```

输出序列:

```
+-----------------------------+--------------------+
|                         Time|minmax(root.test.s1)|
+-----------------------------+--------------------+
|1970-01-01T08:00:00.100+08:00| 0.16666666666666666|
|1970-01-01T08:00:00.200+08:00| 0.16666666666666666|
|1970-01-01T08:00:00.300+08:00|                0.25|
|1970-01-01T08:00:00.400+08:00| 0.08333333333333333|
|1970-01-01T08:00:00.500+08:00| 0.16666666666666666|
|1970-01-01T08:00:00.600+08:00| 0.16666666666666666|
|1970-01-01T08:00:00.700+08:00|                 0.0|
|1970-01-01T08:00:00.800+08:00|  0.3333333333333333|
|1970-01-01T08:00:00.900+08:00| 0.16666666666666666|
|1970-01-01T08:00:01.000+08:00| 0.16666666666666666|
|1970-01-01T08:00:01.100+08:00|                0.25|
|1970-01-01T08:00:01.200+08:00| 0.08333333333333333|
|1970-01-01T08:00:01.300+08:00| 0.08333333333333333|
|1970-01-01T08:00:01.400+08:00|                0.25|
|1970-01-01T08:00:01.500+08:00| 0.16666666666666666|
|1970-01-01T08:00:01.600+08:00| 0.16666666666666666|
|1970-01-01T08:00:01.700+08:00|                1.0|
```

```
|1970-01-01T08:00:01.800+08:00|  0.3333333333333333|
|1970-01-01T08:00:01.900+08:00|                 0.0|
|1970-01-01T08:00:02.000+08:00| 0.16666666666666666|
+-----------------------------+--------------------+
```

1.8 Mode

1.8.1 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，将会输出任意一个。
- 数据中的空值和缺失值将会被忽略，但 NaN 不会被忽略。

1.8.2 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|
+-----------------------------+--------------------+
```

1.9 MvAvg

1.9.1 函数简介

本函数计算序列的移动平均。

函数名： MVAVG

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

参数：

- `window`：移动窗口的长度。默认值为 10.

输出序列：输出单个序列，类型为 DOUBLE。

1.9.2 使用示例

1.9.2.1 指定窗口长度

输入序列：

```
+-----------------------------+-----------+
|                         Time|root.test.s1|
+-----------------------------+-----------+
|1970-01-01T08:00:00.100+08:00|        0.0|
|1970-01-01T08:00:00.200+08:00|        0.0|
|1970-01-01T08:00:00.300+08:00|        1.0|
|1970-01-01T08:00:00.400+08:00|       -1.0|
|1970-01-01T08:00:00.500+08:00|        0.0|
|1970-01-01T08:00:00.600+08:00|        0.0|
|1970-01-01T08:00:00.700+08:00|       -2.0|
|1970-01-01T08:00:00.800+08:00|        2.0|
|1970-01-01T08:00:00.900+08:00|        0.0|
|1970-01-01T08:00:01.000+08:00|        0.0|
|1970-01-01T08:00:01.100+08:00|        1.0|
|1970-01-01T08:00:01.200+08:00|       -1.0|
|1970-01-01T08:00:01.300+08:00|       -1.0|
|1970-01-01T08:00:01.400+08:00|        1.0|
|1970-01-01T08:00:01.500+08:00|        0.0|
|1970-01-01T08:00:01.600+08:00|        0.0|
|1970-01-01T08:00:01.700+08:00|       10.0|
|1970-01-01T08:00:01.800+08:00|        2.0|
|1970-01-01T08:00:01.900+08:00|       -2.0|
|1970-01-01T08:00:02.000+08:00|        0.0|
+-----------------------------+-----------+
```

用于查询的 SQL 语句：

```
select mvavg(s1, "window"="3") from root.test
```

输出序列：

```
+-----------------------------+-----------------------------------+
|                         Time|mvavg(root.test.s1, "window"="3")|
+-----------------------------+-----------------------------------+
|1970-01-01T08:00:00.300+08:00|                0.3333333333333333|
|1970-01-01T08:00:00.400+08:00|                               0.0|
|1970-01-01T08:00:00.500+08:00|               -0.3333333333333333|
|1970-01-01T08:00:00.600+08:00|                               0.0|
|1970-01-01T08:00:00.700+08:00|               -0.6666666666666666|
|1970-01-01T08:00:00.800+08:00|                               0.0|
|1970-01-01T08:00:00.900+08:00|                0.6666666666666666|
|1970-01-01T08:00:01.000+08:00|                               0.0|
|1970-01-01T08:00:01.100+08:00|                0.3333333333333333|
|1970-01-01T08:00:01.200+08:00|                               0.0|
|1970-01-01T08:00:01.300+08:00|               -0.6666666666666666|
|1970-01-01T08:00:01.400+08:00|                               0.0|
|1970-01-01T08:00:01.500+08:00|                0.3333333333333333|
|1970-01-01T08:00:01.600+08:00|                               0.0|
|1970-01-01T08:00:01.700+08:00|                3.3333333333333335|
|1970-01-01T08:00:01.800+08:00|                               4.0|
|1970-01-01T08:00:01.900+08:00|                               0.0|
|1970-01-01T08:00:02.000+08:00|               -0.6666666666666666|
+-----------------------------+-----------------------------------+
```

1.10 PACF

1.10.1 函数简介

本函数通过求解 Yule-Walker 方程，计算序列的偏自相关系数。对于特殊的输入序列，方程可能没有解，此时输出 NaN。

函数名： PACF

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

参数：

- `lag`：最大滞后阶数。默认值为 $\min(10\log_{10} n, n-1)$，n 表示数据点个数。

输出序列： 输出单个序列，类型为 DOUBLE。

1.10.2 使用示例

1.10.2.1 指定滞后阶数

输入序列：

```
+----------------------------+------------+
|                        Time|root.test.s1|
+----------------------------+------------+
|2019-12-27T00:00:00.000+08:00|         5.0|
|2019-12-27T00:05:00.000+08:00|         5.0|
|2019-12-27T00:10:00.000+08:00|         5.0|
|2019-12-27T00:15:00.000+08:00|         5.0|
|2019-12-27T00:20:00.000+08:00|         6.0|
|2019-12-27T00:25:00.000+08:00|         5.0|
|2019-12-27T00:30:00.000+08:00|         6.0|
|2019-12-27T00:35:00.000+08:00|         6.0|
|2019-12-27T00:40:00.000+08:00|         6.0|
|2019-12-27T00:45:00.000+08:00|         6.0|
|2019-12-27T00:50:00.000+08:00|         6.0|
|2019-12-27T00:55:00.000+08:00|    5.982609|
|2019-12-27T01:00:00.000+08:00|   5.9652176|
|2019-12-27T01:05:00.000+08:00|    5.947826|
|2019-12-27T01:10:00.000+08:00|   5.9304347|
|2019-12-27T01:15:00.000+08:00|   5.9130435|
|2019-12-27T01:20:00.000+08:00|   5.8956523|
|2019-12-27T01:25:00.000+08:00|    5.878261|
|2019-12-27T01:30:00.000+08:00|   5.8608694|
|2019-12-27T01:35:00.000+08:00|    5.843478|
.............
Total line number = 18066
```

用于查询的 SQL 语句:

```
select pacf(s1, "lag"="5") from root.test
```

输出序列:

```
+----------------------------+-------------------------------------+
|                        Time|pacf(root.test.s1, "lag"="5")|
+----------------------------+-------------------------------------+
|2019-12-27T00:00:00.000+08:00|                                  1.0|
|2019-12-27T00:05:00.000+08:00|                0.3528915091942786|
|2019-12-27T00:10:00.000+08:00|                0.1761346122516304|
|2019-12-27T00:15:00.000+08:00|                0.1492391973294682|
|2019-12-27T00:20:00.000+08:00|               0.03560059645868398|
|2019-12-27T00:25:00.000+08:00|                0.0366222998995286|
+----------------------------+-------------------------------------+
```

1.11 Percentile

1.11.1 函数简介

本函数用于计算单列数值型数据的精确或近似分位数。

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`：所求分位数在所有数据中的排名百分比，取值范围为 (0,1]，默认值为 0.5。如当设为 0.5 时则计算中位数。
- `error`：近似分位数的基于排名的误差百分比，取值范围为 [0,1)，默认值为 0。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。当 `error` =0 时，计算结果为精确分位数。

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、值为分位数的数据点。

提示： 数据中的空值、缺失值和 `NaN` 将会被忽略。

1.11.2 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")|
+----------------------------+-----------------------------------------------------+
|1970-01-01T08:00:00.000+08:00|                                0.1801469624042511|
+----------------------------+-----------------------------------------------------+
```

1.12 Period

1.12.1 函数简介

本函数用于计算单列数值型数据的周期。

函数名： PERIOD

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 INT32，序列仅包含一个时间戳为 0、值为周期的数据点。

1.12.2 使用示例

输入序列：

```
+----------------------------+--------------+
|                        Time|root.test.d3.s1|
+----------------------------+--------------+
|1970-01-01T08:00:00.001+08:00|           1.0|
|1970-01-01T08:00:00.002+08:00|           2.0|
|1970-01-01T08:00:00.003+08:00|           3.0|
|1970-01-01T08:00:00.004+08:00|           1.0|
|1970-01-01T08:00:00.005+08:00|           2.0|
|1970-01-01T08:00:00.006+08:00|           3.0|
|1970-01-01T08:00:00.007+08:00|           1.0|
|1970-01-01T08:00:00.008+08:00|           2.0|
|1970-01-01T08:00:00.009+08:00|           3.0|
+----------------------------+--------------+
```

用于查询的 SQL 语句：

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

输出序列：

```
+-----------------------------+-----------------------+
|                         Time|period(root.test.d3.s1)|
+-----------------------------+-----------------------+
|1970-01-01T08:00:00.000+08:00|                      3|
+-----------------------------+-----------------------+
```

1.12.2.1 Zeppelin 示例

链接: <http://101.6.15.213:18181/#/notebook/2GEJBUSZ9>

1.13 QLB

1.13.1 函数简介

本函数对输入序列计算 Q_{LB} 统计量，并计算对应的 p 值。p 值越小表明序列越有可能为非平稳序列。

函数名： QLB

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

参数：

- `lag`：计算时用到的最大延迟阶数，取值应为 1 至 n-2 之间的整数，n 为序列采样总数。默认取 n-2。

输出序列： 输出单个序列，类型为 DOUBLE。该序列是 Q_{LB} 统计量对应的 p 值，时间标签代表偏移阶数。

提示： Q_{LB} 统计量由自相关系数求得，如需得到统计量而非 p 值，可以使用 Auto-Correlation 函数。

1.13.2 使用示例

1.13.2.1 使用默认参数

输入序列：

```
+-----------------------------+--------------+
|                         Time|root.test.d1.s1|
+-----------------------------+--------------+
|1970-01-01T00:00:00.100+08:00|          1.22|
|1970-01-01T00:00:00.200+08:00|         -2.78|
|1970-01-01T00:00:00.300+08:00|          1.53|
|1970-01-01T00:00:00.400+08:00|          0.70|
```

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

用于查询的 SQL 语句:

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

输出序列:

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

1.14 Resample

1.14.1 函数简介

本函数对输入序列按照指定的频率进行重采样，包括上采样和下采样。目前，本函数支持的上采样方法包括 NaN 填充法 (NaN)、前值填充法 (FFill)、后值填充法 (BFill) 以及线性插值法 (Linear)；本函数支持的下采样方法为分组聚合，聚合方法包括最大值 (Max)、最小值 (Min)、首值 (First)、末值 (Last)、平均值 (Mean) 和中位数 (Median)。

函数名： RESAMPLE

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

参数：

- every：重采样频率，是一个有单位的正数。目前支持五种单位，分别是'ms'（毫秒）、's'（秒）、'm'（分钟）、'h'（小时）和'd'（天）。该参数不允许缺省。
- interp：上采样的插值方法，取值为'NaN'、'FFill'、'BFill' 或'Linear'。在缺省情况下，使用 NaN 填充法。
- aggr：下采样的聚合方法，取值为'Max'、'Min'、'First'、'Last'、'Mean' 或'Median'。在缺省情况下，使用平均数聚合。
- start：重采样的起始时间（包含），是一个格式为'yyyy-MM-dd HH:mm:ss' 的时间字符串。在缺省情况下，使用第一个有效数据点的时间戳。
- end：重采样的结束时间（不包含），是一个格式为'yyyy-MM-dd HH:mm:ss' 的时间字符串。在缺省情况下，使用最后一个有效数据点的时间戳。

输出序列： 输出单个序列，类型为 DOUBLE。该序列按照重采样频率严格等间隔分布。

提示： 数据中的 NaN 将会被忽略。

1.14.2 使用示例

1.14.2.1 上采样

当重采样频率高于数据原始频率时，将会进行上采样。

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

1.14.2.2 下采样

当重采样频率低于数据原始频率时，将会进行下采样。

输入序列同上，用于查询的 SQL 语句如下：

```
select resample(s1,'every'='30m','aggr'='first') from root.test.d1
```

输出序列：

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

1.14.2.3 指定重采样时间段

可以使用 `start` 和 `end` 两个参数指定重采样的时间段，超出实际时间范围的部分会被插值填补。

输入序列同上，用于查询的 SQL 语句如下：

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

输出序列：

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

1.15 Sample

1.15.1 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'reservoir' 或'isometric'。在缺省情况下，采用蓄水池采样法。
- `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。

1.15.2 使用示例

1.15.2.1 蓄水池采样

当`method`参数为'reservoir' 或缺省时，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：

```
+-----------------------------+---------------+
|                         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,'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|
+-----------------------------+-----------------------------------------------------------+
```

1.15.2.2 等距采样

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

输入序列同上，用于查询的 SQL 语句如下：

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

输出序列：

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

1.16 Segment

1.16.1 函数简介

本函数按照数据的线性变化趋势将数据划分为多个子序列，返回分段直线拟合后的子序列首值或所有拟合值。

函数名： SEGMENT

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

参数：

- `output`： "all" 输出所有拟合值；"first" 输出子序列起点拟合值。默认为"first"。
- `error`： 判定存在线性趋势的误差允许阈值。误差的定义为子序列进行线性拟合的误差的绝对值的均值。默认为 0.1.

输出序列： 输出单个序列，类型为 DOUBLE。

提示： 函数默认所有数据等时间间隔分布。函数读取所有数据，若原始数据过多，请先进行降采样处理。拟合采用自底向上方法，子序列的尾值可能会被认作子序列首值输出。

1.16.2 使用示例

输入序列：

```
+-----------------------------+------------+
|                         Time|root.test.s1|
+-----------------------------+------------+
|1970-01-01T08:00:00.000+08:00|         5.0|
|1970-01-01T08:00:00.100+08:00|         0.0|
|1970-01-01T08:00:00.200+08:00|         1.0|
|1970-01-01T08:00:00.300+08:00|         2.0|
|1970-01-01T08:00:00.400+08:00|         3.0|
|1970-01-01T08:00:00.500+08:00|         4.0|
|1970-01-01T08:00:00.600+08:00|         5.0|
|1970-01-01T08:00:00.700+08:00|         6.0|
|1970-01-01T08:00:00.800+08:00|         7.0|
|1970-01-01T08:00:00.900+08:00|         8.0|
|1970-01-01T08:00:01.000+08:00|         9.0|
|1970-01-01T08:00:01.100+08:00|         9.1|
|1970-01-01T08:00:01.200+08:00|         9.2|
|1970-01-01T08:00:01.300+08:00|         9.3|
|1970-01-01T08:00:01.400+08:00|         9.4|
|1970-01-01T08:00:01.500+08:00|         9.5|
|1970-01-01T08:00:01.600+08:00|         9.6|
|1970-01-01T08:00:01.700+08:00|         9.7|
|1970-01-01T08:00:01.800+08:00|         9.8|
|1970-01-01T08:00:01.900+08:00|         9.9|
|1970-01-01T08:00:02.000+08:00|        10.0|
```

```
|1970-01-01T08:00:02.100+08:00|               8.0|
|1970-01-01T08:00:02.200+08:00|               6.0|
|1970-01-01T08:00:02.300+08:00|               4.0|
|1970-01-01T08:00:02.400+08:00|               2.0|
|1970-01-01T08:00:02.500+08:00|               0.0|
|1970-01-01T08:00:02.600+08:00|              -2.0|
|1970-01-01T08:00:02.700+08:00|              -4.0|
|1970-01-01T08:00:02.800+08:00|              -6.0|
|1970-01-01T08:00:02.900+08:00|              -8.0|
|1970-01-01T08:00:03.000+08:00|             -10.0|
|1970-01-01T08:00:03.100+08:00|              10.0|
|1970-01-01T08:00:03.200+08:00|              10.0|
|1970-01-01T08:00:03.300+08:00|              10.0|
|1970-01-01T08:00:03.400+08:00|              10.0|
|1970-01-01T08:00:03.500+08:00|              10.0|
|1970-01-01T08:00:03.600+08:00|              10.0|
|1970-01-01T08:00:03.700+08:00|              10.0|
|1970-01-01T08:00:03.800+08:00|              10.0|
|1970-01-01T08:00:03.900+08:00|              10.0|
+-----------------------------+-------------------+
```

用于查询的 SQL 语句：

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

输出序列：

```
+-----------------------------+-----------------------------------------+
|                         Time|segment(root.test.s1, "error"="0.1")|
+-----------------------------+-----------------------------------------+
|1970-01-01T08:00:00.000+08:00|                                      5.0|
|1970-01-01T08:00:00.200+08:00|                                      1.0|
|1970-01-01T08:00:01.000+08:00|                                      9.0|
|1970-01-01T08:00:02.000+08:00|                                     10.0|
|1970-01-01T08:00:03.000+08:00|                                    -10.0|
|1970-01-01T08:00:03.200+08:00|                                     10.0|
+-----------------------------+-----------------------------------------+
```

1.17 Skew

1.17.1 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：输出单个序列，类型为 DOUBLE，序列仅包含一个时间戳为 0、值为总体偏度的数据点。

提示：数据中的空值、缺失值和 NaN 将会被忽略。

1.17.2 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|
+-----------------------------+--------------------+
```

1.18 Spline

1.18.1 函数简介

本函数提供对原始序列进行三次样条曲线拟合后的插值重采样。

函数名： SPLINE

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

参数：

- points：重采样个数。

输出序列：输出单个序列，类型为 DOUBLE。

提示：输出序列保留输入序列的首尾值，等时间间隔采样。仅当输入点个数不少于 4 个时才计算插值。

1.18.2 使用示例

1.18.2.1 指定插值个数

输入序列：

```
+-----------------------------+-----------+
|                         Time|root.test.s1|
+-----------------------------+-----------+
|1970-01-01T08:00:00.000+08:00|        0.0|
|1970-01-01T08:00:00.300+08:00|        1.2|
|1970-01-01T08:00:00.500+08:00|        1.7|
|1970-01-01T08:00:00.700+08:00|        2.0|
|1970-01-01T08:00:00.900+08:00|        2.1|
|1970-01-01T08:00:01.100+08:00|        2.0|
|1970-01-01T08:00:01.200+08:00|        1.8|
|1970-01-01T08:00:01.300+08:00|        1.2|
|1970-01-01T08:00:01.400+08:00|        1.0|
|1970-01-01T08:00:01.500+08:00|        1.6|
+-----------------------------+-----------+
```

用于查询的 SQL 语句：

```
select spline(s1, "points"="151") from root.test
```

输出序列：

```
+-----------------------------+-----------------------------------+
|                         Time|spline(root.test.s1, "points"="151")|
+-----------------------------+-----------------------------------+
|1970-01-01T08:00:00.000+08:00|                                0.0|
|1970-01-01T08:00:00.010+08:00|                0.04870000251134237|
|1970-01-01T08:00:00.020+08:00|                0.09680000495910646|
|1970-01-01T08:00:00.030+08:00|                0.14430000734329226|
```

|1970–01–01T08:00:00.040+08:00| 0.19120000966389972|
|1970–01–01T08:00:00.050+08:00| 0.23750001192092896|
|1970–01–01T08:00:00.060+08:00| 0.2832000141143799|
|1970–01–01T08:00:00.070+08:00| 0.32830001624425253|
|1970–01–01T08:00:00.080+08:00| 0.3728000183105469|
|1970–01–01T08:00:00.090+08:00| 0.416700020313263|
|1970–01–01T08:00:00.100+08:00| 0.4600000222524008|
|1970–01–01T08:00:00.110+08:00| 0.5027000241279602|
|1970–01–01T08:00:00.120+08:00| 0.5448000259399414|
|1970–01–01T08:00:00.130+08:00| 0.5863000276883443|
|1970–01–01T08:00:00.140+08:00| 0.627200029373169|
|1970–01–01T08:00:00.150+08:00| 0.6675000309944153|
|1970–01–01T08:00:00.160+08:00| 0.7072000325520833|
|1970–01–01T08:00:00.170+08:00| 0.7463000340461731|
|1970–01–01T08:00:00.180+08:00| 0.7848000354766846|
|1970–01–01T08:00:00.190+08:00| 0.8227000368436178|
|1970–01–01T08:00:00.200+08:00| 0.8600000381469728|
|1970–01–01T08:00:00.210+08:00| 0.8967000393867494|
|1970–01–01T08:00:00.220+08:00| 0.9328000405629477|
|1970–01–01T08:00:00.230+08:00| 0.9683000416755676|
|1970–01–01T08:00:00.240+08:00| 1.0032000427246095|
|1970–01–01T08:00:00.250+08:00| 1.037500043710073|
|1970–01–01T08:00:00.260+08:00| 1.071200044631958|
|1970–01–01T08:00:00.270+08:00| 1.1043000454902647|
|1970–01–01T08:00:00.280+08:00| 1.1368000462849934|
|1970–01–01T08:00:00.290+08:00| 1.1687000470161437|
|1970–01–01T08:00:00.300+08:00| 1.2000000476837158|
|1970–01–01T08:00:00.310+08:00| 1.2307000483103594|
|1970–01–01T08:00:00.320+08:00| 1.2608000489139557|
|1970–01–01T08:00:00.330+08:00| 1.2903000494873524|
|1970–01–01T08:00:00.340+08:00| 1.3192000500233967|
|1970–01–01T08:00:00.350+08:00| 1.3475000505149364|
|1970–01–01T08:00:00.360+08:00| 1.3752000509548186|
|1970–01–01T08:00:00.370+08:00| 1.402300051335891|
|1970–01–01T08:00:00.380+08:00| 1.4288000516510009|
|1970–01–01T08:00:00.390+08:00| 1.4547000518929958|
|1970–01–01T08:00:00.400+08:00| 1.480000052054723|
|1970–01–01T08:00:00.410+08:00| 1.5047000521290301|
|1970–01–01T08:00:00.420+08:00| 1.5288000521087646|
|1970–01–01T08:00:00.430+08:00| 1.5523000519867738|
|1970–01–01T08:00:00.440+08:00| 1.575200051755905|
|1970–01–01T08:00:00.450+08:00| 1.597500051409006|
|1970–01–01T08:00:00.460+08:00| 1.619200050938924|
|1970–01–01T08:00:00.470+08:00| 1.6403000503385066|
|1970–01–01T08:00:00.480+08:00| 1.660800049600601|
|1970–01–01T08:00:00.490+08:00| 1.680700048718055|
|1970–01–01T08:00:00.500+08:00| 1.7000000476837158|

|1970–01–01T08:00:00.510+08:00| 1.7188475466453037|
|1970–01–01T08:00:00.520+08:00| 1.7373800457262996|
|1970–01–01T08:00:00.530+08:00| 1.7555825448831923|
|1970–01–01T08:00:00.540+08:00| 1.7734400440724702|
|1970–01–01T08:00:00.550+08:00| 1.790937543250622|
|1970–01–01T08:00:00.560+08:00| 1.8080600423741364|
|1970–01–01T08:00:00.570+08:00| 1.8247925413995016|
|1970–01–01T08:00:00.580+08:00| 1.8411200402832066|
|1970–01–01T08:00:00.590+08:00| 1.8570275389817397|
|1970–01–01T08:00:00.600+08:00| 1.8725000374515897|
|1970–01–01T08:00:00.610+08:00| 1.8875225356492449|
|1970–01–01T08:00:00.620+08:00| 1.902080033531194|
|1970–01–01T08:00:00.630+08:00| 1.9161575310539258|
|1970–01–01T08:00:00.640+08:00| 1.9297400281739288|
|1970–01–01T08:00:00.650+08:00| 1.9428125248476913|
|1970–01–01T08:00:00.660+08:00| 1.9553600210317021|
|1970–01–01T08:00:00.670+08:00| 1.96736751668245|
|1970–01–01T08:00:00.680+08:00| 1.9788200117564232|
|1970–01–01T08:00:00.690+08:00| 1.9897025062101101|
|1970–01–01T08:00:00.700+08:00| 2.0|
|1970–01–01T08:00:00.710+08:00| 2.0097024933913334|
|1970–01–01T08:00:00.720+08:00| 2.0188199867081615|
|1970–01–01T08:00:00.730+08:00| 2.027367479995188|
|1970–01–01T08:00:00.740+08:00| 2.0353599732971155|
|1970–01–01T08:00:00.750+08:00| 2.0428124666586482|
|1970–01–01T08:00:00.760+08:00| 2.049739960124489|
|1970–01–01T08:00:00.770+08:00| 2.056157453739342|
|1970–01–01T08:00:00.780+08:00| 2.06207994754791|
|1970–01–01T08:00:00.790+08:00| 2.067522441594897|
|1970–01–01T08:00:00.800+08:00| 2.072499935925006|
|1970–01–01T08:00:00.810+08:00| 2.07702743058294|
|1970–01–01T08:00:00.820+08:00| 2.081119925613404|
|1970–01–01T08:00:00.830+08:00| 2.0847924210611|
|1970–01–01T08:00:00.840+08:00| 2.0880599169707317|
|1970–01–01T08:00:00.850+08:00| 2.0909374133870027|
|1970–01–01T08:00:00.860+08:00| 2.0934399103546166|
|1970–01–01T08:00:00.870+08:00| 2.0955824079182768|
|1970–01–01T08:00:00.880+08:00| 2.0973799061226863|
|1970–01–01T08:00:00.890+08:00| 2.098847405012549|
|1970–01–01T08:00:00.900+08:00| 2.0999999046325684|
|1970–01–01T08:00:00.910+08:00| 2.1005574051201332|
|1970–01–01T08:00:00.920+08:00| 2.1002599065303778|
|1970–01–01T08:00:00.930+08:00| 2.0991524087846245|
|1970–01–01T08:00:00.940+08:00| 2.0972799118041947|
|1970–01–01T08:00:00.950+08:00| 2.0946874155104105|
|1970–01–01T08:00:00.960+08:00| 2.0914199198245944|
|1970–01–01T08:00:00.970+08:00| 2.0875224246680673|

|1970–01–01T08:00:00.980+08:00| 2.083039929962151|
|1970–01–01T08:00:00.990+08:00| 2.0780174356281687|
|1970–01–01T08:00:01.000+08:00| 2.0724999415874406|
|1970–01–01T08:00:01.010+08:00| 2.06653244776129|
|1970–01–01T08:00:01.020+08:00| 2.060159954071038|
|1970–01–01T08:00:01.030+08:00| 2.053427460438006|
|1970–01–01T08:00:01.040+08:00| 2.046379966783517|
|1970–01–01T08:00:01.050+08:00| 2.0390624730288924|
|1970–01–01T08:00:01.060+08:00| 2.031519979095454|
|1970–01–01T08:00:01.070+08:00| 2.0237974849045237|
|1970–01–01T08:00:01.080+08:00| 2.015939990377423|
|1970–01–01T08:00:01.090+08:00| 2.0079924954354746|
|1970–01–01T08:00:01.100+08:00| 2.0|
|1970–01–01T08:00:01.110+08:00| 1.9907018211101906|
|1970–01–01T08:00:01.120+08:00| 1.9788509124245144|
|1970–01–01T08:00:01.130+08:00| 1.9645127287932083|
|1970–01–01T08:00:01.140+08:00| 1.9477527250665083|
|1970–01–01T08:00:01.150+08:00| 1.9286363560946513|
|1970–01–01T08:00:01.160+08:00| 1.9072290767278735|
|1970–01–01T08:00:01.170+08:00| 1.8835963418164114|
|1970–01–01T08:00:01.180+08:00| 1.8578036062105014|
|1970–01–01T08:00:01.190+08:00| 1.8299163247603802|
|1970–01–01T08:00:01.200+08:00| 1.7999999523162842|
|1970–01–01T08:00:01.210+08:00| 1.7623635841923329|
|1970–01–01T08:00:01.220+08:00| 1.7129696477516976|
|1970–01–01T08:00:01.230+08:00| 1.6543635959181928|
|1970–01–01T08:00:01.240+08:00| 1.5890908816156328|
|1970–01–01T08:00:01.250+08:00| 1.5196969577678319|
|1970–01–01T08:00:01.260+08:00| 1.4487272772986044|
|1970–01–01T08:00:01.270+08:00| 1.3787272931317647|
|1970–01–01T08:00:01.280+08:00| 1.3122424581911272|
|1970–01–01T08:00:01.290+08:00| 1.251818225400506|
|1970–01–01T08:00:01.300+08:00| 1.2000000476837158|
|1970–01–01T08:00:01.310+08:00| 1.1548000470995912|
|1970–01–01T08:00:01.320+08:00| 1.1130667107899999|
|1970–01–01T08:00:01.330+08:00| 1.0756000393033045|
|1970–01–01T08:00:01.340+08:00| 1.043200033187868|
|1970–01–01T08:00:01.350+08:00| 1.016666692992053|
|1970–01–01T08:00:01.360+08:00| 0.9968000192642223|
|1970–01–01T08:00:01.370+08:00| 0.9844000125527389|
|1970–01–01T08:00:01.380+08:00| 0.9802666734059655|
|1970–01–01T08:00:01.390+08:00| 0.9852000023722649|
|1970–01–01T08:00:01.400+08:00| 1.0|
|1970–01–01T08:00:01.410+08:00| 1.023999999165535|
|1970–01–01T08:00:01.420+08:00| 1.0559999990463256|
|1970–01–01T08:00:01.430+08:00| 1.0959999996423722|
|1970–01–01T08:00:01.440+08:00| 1.1440000009536744|

```
|1970-01-01T08:00:01.450+08:00|                1.2000000029802322|
|1970-01-01T08:00:01.460+08:00|                 1.264000005722046|
|1970-01-01T08:00:01.470+08:00|                1.3360000091791153|
|1970-01-01T08:00:01.480+08:00|                1.4160000133514405|
|1970-01-01T08:00:01.490+08:00|                1.5040000182390214|
|1970-01-01T08:00:01.500+08:00|                 1.600000023841858|
+-----------------------------+----------------------------------+
```

1.19 Spread

1.19.1 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： 数据中的空值、缺失值和 NaN 将会被忽略。

1.19.2 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|
+-----------------------------+-----------------------+
```

1.20 Stddev

1.20.1 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。

提示： 数据中的空值、缺失值和 NaN 将会被忽略。

1.20.2 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|
+-----------------------------+------------------------+
```

1.21 IntegralAvg

1.21.1 函数简介

本函数用于计算时间序列的函数均值，即在相同时间单位下的数值积分除以序列总的时间跨度。更多关于数值积分计算的信息请参考 Integral 函数。

函数名： INTEGRALAVG

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、值为时间加权平均结果的数据点。

提示：

- 时间加权的平均值等于在任意时间单位 unit 下计算的数值积分（即折线图中每相邻两个数据点和时间轴形成的直角梯形的面积之和），

除以相同时间单位下输入序列的时间跨度，其值与具体采用的时间单位无关，默认与 IoTDB 时间单位一致。

- 数据中的 NaN 将会被忽略。折线将以临近两个有值数据点为准。
- 输入序列为空时，函数输出结果为 0；仅有一个数据点时，输出结果为该点数值。

1.21.2 使用示例

输入序列：

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

用于查询的 SQL 语句：

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

输出序列：

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

其计算公式为：

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

1.22 ZScore

1.22.1 函数简介

本函数将输入序列使用 z-score 方法进行归一化。

函数名： ZSCORE

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

参数：

- `compute`：若设置为"batch"，则将数据全部读入后转换；若设置为"stream"，则需用户提供均值及方差进行流式计算转换。默认为"batch"。
- `avg`：使用流式计算时的均值。
- `sd`：使用流式计算时的标准差。

输出序列： 输出单个序列，类型为 DOUBLE。

1.22.2 使用示例

1.22.2.1 全数据计算

输入序列：

```
+-----------------------------+------------+
|                         Time|root.test.s1|
+-----------------------------+------------+
|1970-01-01T08:00:00.100+08:00|         0.0|
```

```
|1970-01-01T08:00:00.200+08:00|                0.0|
|1970-01-01T08:00:00.300+08:00|                1.0|
|1970-01-01T08:00:00.400+08:00|               -1.0|
|1970-01-01T08:00:00.500+08:00|                0.0|
|1970-01-01T08:00:00.600+08:00|                0.0|
|1970-01-01T08:00:00.700+08:00|               -2.0|
|1970-01-01T08:00:00.800+08:00|                2.0|
|1970-01-01T08:00:00.900+08:00|                0.0|
|1970-01-01T08:00:01.000+08:00|                0.0|
|1970-01-01T08:00:01.100+08:00|                1.0|
|1970-01-01T08:00:01.200+08:00|               -1.0|
|1970-01-01T08:00:01.300+08:00|               -1.0|
|1970-01-01T08:00:01.400+08:00|                1.0|
|1970-01-01T08:00:01.500+08:00|                0.0|
|1970-01-01T08:00:01.600+08:00|                0.0|
|1970-01-01T08:00:01.700+08:00|               10.0|
|1970-01-01T08:00:01.800+08:00|                2.0|
|1970-01-01T08:00:01.900+08:00|               -2.0|
|1970-01-01T08:00:02.000+08:00|                0.0|
+-----------------------------+--------------------+
```

用于查询的 SQL 语句:

```
select zscore(s1) from root.test
```

输出序列:

```
+-----------------------------+--------------------+
|                         Time|zscore(root.test.s1)|
+-----------------------------+--------------------+
|1970-01-01T08:00:00.100+08:00|-0.20672455764868078|
|1970-01-01T08:00:00.200+08:00|-0.20672455764868078|
|1970-01-01T08:00:00.300+08:00| 0.20672455764868078|
|1970-01-01T08:00:00.400+08:00| -0.6201736729460423|
|1970-01-01T08:00:00.500+08:00|-0.20672455764868078|
|1970-01-01T08:00:00.600+08:00|-0.20672455764868078|
|1970-01-01T08:00:00.700+08:00|  -1.033622788243404|
|1970-01-01T08:00:00.800+08:00|  0.6201736729460423|
|1970-01-01T08:00:00.900+08:00|-0.20672455764868078|
|1970-01-01T08:00:01.000+08:00|-0.20672455764868078|
|1970-01-01T08:00:01.100+08:00| 0.20672455764868078|
|1970-01-01T08:00:01.200+08:00| -0.6201736729460423|
|1970-01-01T08:00:01.300+08:00| -0.6201736729460423|
|1970-01-01T08:00:01.400+08:00| 0.20672455764868078|
|1970-01-01T08:00:01.500+08:00|-0.20672455764868078|
|1970-01-01T08:00:01.600+08:00|-0.20672455764868078|
|1970-01-01T08:00:01.700+08:00|  3.9277665953249348|
|1970-01-01T08:00:01.800+08:00|  0.6201736729460423|
|1970-01-01T08:00:01.900+08:00|  -1.033622788243404|
```

```
|1970-01-01T08:00:02.000+08:00|-0.20672455764868078|
+-----------------------------+--------------------+
```