

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3月12日

目录

1	廾始	1
	1.1	概述
	1.2	系统对标
	1.3	常见问题 2
2	数据	画像 3
	2.1	Cov
	2.2	Distinct
	2.3	Histogram
	2.4	Integral
	2.5	Mad
	2.6	Mean
	2.7	Median
	2.8	Mode
	2.9	Pearson
	2.10	Percentile
	2.11	Sample
	2.12	Skew
	2.13	Spread
	2.14	Stddev
3	数据	质量 5
	3.1	Completeness
	3.2	Consistency
	3.3	Timeliness
	3.4	Validity
4	数据	修复
	4.1	Fill
	4.2	ValueRepair
	4.3	TimestampRepair
5	数据	匹配 <u>16</u>
	5.1	SeriesAlign
	5.2	SeriesSimilarity
	5.3	DTW

		E CONTRACTOR DE LA CONT	目录
6	异常	· ·检测	17
	6.1	Range	17
	6.2	KSigma	18
	6.3	LOF	19
7	复杂	· ·事件处理	20
	7.1	EventMatching	20
	7.2	MissingEventRecovery	20
	7.3	EventNameRepair	20
	7.4	EventTimeRepair	20
	7.5	SEQ	20
	7.6	AND	20

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
- 2. 将 jar 包复制到 IoTDB 程序目录的 ext\udf 目录下
- 3. 在 IoTDB 中使用下面的 SQL 语句注册 UDF

```
create function completeness as 'cn.edu.thu.dquality.udf.UDTFCompleteness'
create function consistency as 'cn.edu.thu.dquality.udf.UDTFConsistency'
create function timeliness as 'cn.edu.thu.dquality.udf.UDTFTimeliness'
create function validity as 'cn.edu.thu.dquality.udf.UDTFValidity'
```

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 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 Cov
- 2.2 Distinct
- 2.3 Histogram
- 2.4 Integral
- 2.5 Mad

2.5.1 函数简介

本函数用于计算单列数值型数据的近似绝对中位差,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,如且 error =0.01,即精确绝对中位差为 a,近似绝对中位差为 b,则必然有以下不等式成立:

 $0.99a \leq b \leq 1.01a$

输出序列: 近似绝对中位差

2.6 Mean

2.7 Median

2.7.1 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: PERCENTILE

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

- **2.8** Mode
- 2.9 Pearson
- 2.10 Percentile

2.10.1 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: 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。
- error: 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。

输出序列: 近似分位数

- **2.11 Sample**
- **2.12** Skew
- 2.13 Spread
- 2.14 Stddev

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

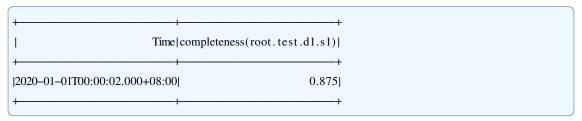
3.1.2.1 参数缺省

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

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

输出序列:



3.1.2.2 指定窗口大小

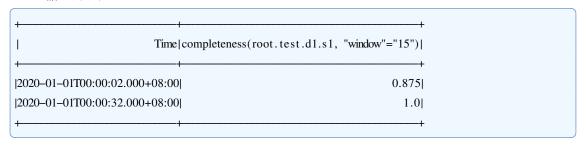
在指定窗口大小的情况下,本函数会把输入数据划分为若干个窗口计算完整性。 输入序列:

Tim	e[root.test.d1.s1]
2020-01-01T00:00:02.000+08:0	00 100.0
2020-01-01T00:00:03.000+08:0	
2020-01-01T00:00:04.000+08:0	
2020-01-01T00:00:06.000+08:0	
2020-01-01T00:00:08.000+08:0	
2020-01-01T00:00:10.000+08:0	00 108.0
2020-01-01T00:00:14.000+08:0	00 112.0
2020-01-01T00:00:15.000+08:0	00 113.0
2020-01-01T00:00:16.000+08:0	00 114.0
2020-01-01T00:00:18.000+08:0	00 116.0
2020-01-01T00:00:20.000+08:0	00 118.0
2020-01-01T00:00:22.000+08:0	00 120.0
2020-01-01T00:00:26.000+08:0	00 124.0
2020-01-01T00:00:28.000+08:0	00 126.0
2020-01-01T00:00:30.000+08:0	00 NaN
2020-01-01T00:00:32.000+08:0	00 130.0
2020-01-01T00:00:34.000+08:0	00 132.0
2020-01-01T00:00:36.000+08:0	00 134.0
2020-01-01T00:00:38.000+08:0	00 136.0
2020-01-01T00:00:40.000+08:0	00 138.0
2020-01-01T00:00:42.000+08:0	00 140.0
2020-01-01T00:00:44.000+08:0	00 142.0
2020-01-01T00:00:46.000+08:0	00 144.0
2020-01-01T00:00:48.000+08:0	00 146.0
l control of the cont	
2020-01-01T00:00:50.000+08:0	00 148.0

 	+
2020-01-01T00:01:00.000+08:00	158.0
2020-01-01T00:00:58.000+08:00	156.0
2020-01-01T00:00:56.000+08:00	154.0
2020-01-01T00:00:54.000+08:00	152.0

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

输出序列:



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。

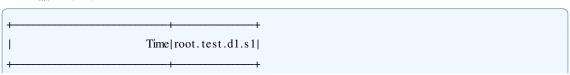
输出序列: 输出单个序列,类型为 DOUBLE,其中每一个数据点的值的范围都是 [0,1]。

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:



```
|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|
```

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

输出序列:



3.2.2.2 指定窗口大小

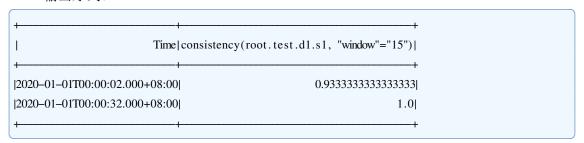
在指定窗口大小的情况下,本函数会把输入数据划分为若干个窗口计算一致性。 输入序列:

```
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
```

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

输出序列:



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。

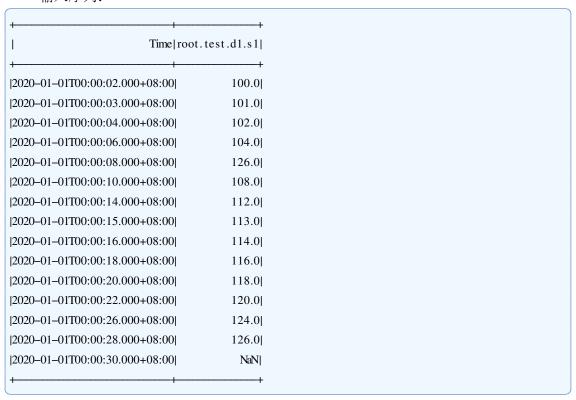
输出序列: 输出单个序列,类型为 DOUBLE,其中每一个数据点的值的范围都是 [0,1]。

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

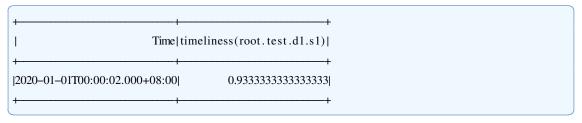
在参数缺省的情况下,本函数将会把全部输入数据都作为同一个窗口计算时效性。 输入序列:



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:



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
```

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

输出序列:



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:

<u> </u>	+	+
1	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:



3.4.2.2 指定窗口大小

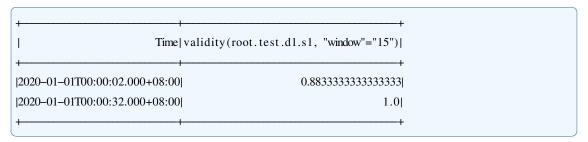
在指定窗口大小的情况下,本函数会把输入数据划分为若干个窗口计算有效性。 输入序列:

+	
Time root.	test.d1.s1
12020 01 01700 00 02 000 00 00	100.01
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



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

输出序列:



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 ValueRepair

4.3 TimestampRepair

第5章 数据匹配

- 5.1 SeriesAlign
- **5.2** SeriesSimilarity
- **5.3 DTW**

第6章 异常检测

6.1 Range

6.1.1 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。

6.1.2 使用示例

6.1.2.1 指定上界与下界

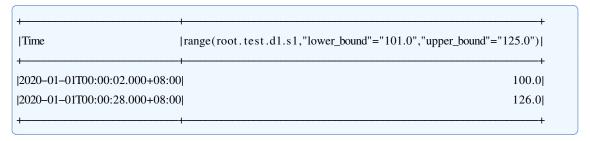
输入序列:

+		+
1	Time root.t	est.d1.s1
12020 01 01770 00 02 0	00.00.00	100.01
2020-01-01T00:00:02.0	•	100.0
2020-01-01T00:00:03.0	00+08:00	101.0
2020-01-01T00:00:04.0	00+08:00	102.0
2020-01-01T00:00:06.0	00+08:00	104.0
2020-01-01T00:00:08.0	00+08:00	126.0
2020-01-01T00:00:10.0	00+08:00	108.0
2020-01-01T00:00:14.0	00+08:00	112.0
2020-01-01T00:00:15.0	00+08:00	113.0
2020-01-01T00:00:16.0	00+08:00	114.0
2020-01-01T00:00:18.0	00+08:00	116.0
2020-01-01T00:00:20.0	00+08:00	118.0
2020-01-01T00:00:22.0	00+08:00	120.0
2020-01-01T00:00:26.0	00+08:00	124.0
2020-01-01T00:00:28.0	00+08:00	126.0
2020-01-01T00:00:30.0	00+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:



6.2 KSigma

6.2.1 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。

6.2.2 使用示例

6.2.2.1 指定 k

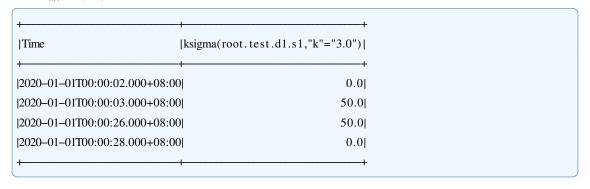
输入序列:

+	-1
Tim	e root.test.d1.s1
+	++
2020-01-01T00:00:02.000+08:0	0.0
2020-01-01T00:00:03.000+08:0	0 50.0
2020-01-01T00:00:04.000+08:0	0 100.0
2020-01-01T00:00:06.000+08:0	0 150.0
2020-01-01T00:00:08.000+08:0	0 200.0
2020-01-01T00:00:10.000+08:0	0 200.0
2020-01-01T00:00:14.000+08:0	0 200.0
2020-01-01T00:00:15.000+08:0	0 200.0
2020-01-01T00:00:16.000+08:0	0 200.0
2020-01-01T00:00:18.000+08:0	0 200.0
2020-01-01T00:00:20.000+08:0	0 150.0

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
+	
	·

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

输出序列:



6.3 LOF

第7章 复杂事件处理

7.1 EventMatching

7.2 MissingEventRecovery

7.3 EventNameRepair

7.4 EventTimeRepair

7.5 SEQ

7.5.1 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。

7.5.2 使用示例

TODO

7.6 AND

7.6.1 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。

7.6.2 使用示例

TODO