

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11日

目录

1	开始	
	1.1	概述
	1.2	系统对标
	1.3	常见问题 2
2	数据	画像
	2.1	Cov
	2.2	Distinct
	2.3	Histogram
	2.4	Integral
	2.5	Mean
	2.6	Median
	2.7	Mode
	2.8	Pearson
	2.9	Percentile
	2.10	Sample
		Skew
	2.12	Spread
		Stddev
3	数据	质量
J	3.1	Completeness
	3.1	Consistency
	3.3	Timeliness
	3.4	Validity
	Э. -	validity
4	数据	修复 14
	4.1	Fill
	4.2	ValueRepair
	4.3	TimestampRepair
5	数据	<u>匹配</u>
	5.1	SeriesAlign
	5.2	SeriesSimilarity
	5.3	DTW

		E	录
6	异常	· 检测	16
	6.1	Range	16
	6.2	KSigma	16
	6.3	LOF	16
7	复杂	事件处理	17
	7.1	EventMatching	17
	7.2	MissingEventRecovery	17
	7.3	EventNameRepair	17
	7.4	EventTimeRepair	17
	7.5	SEQ	17
	7.6	AND	17

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 Mean
- 2.6 Median
- **2.7 Mode**
- 2.8 Pearson
- 2.9 Percentile
- **2.10** Sample
- **2.11 Skew**
- 2.12 Spread
- **2.13 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

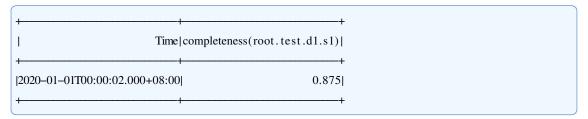
3.1.2.1 参数缺省

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

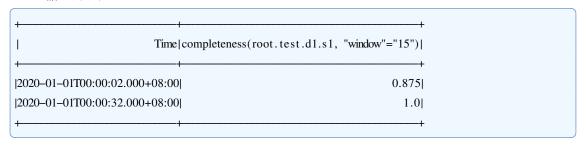
在指定窗口大小的情况下,本函数会把输入数据划分为若干个窗口计算完整性。 输入序列:

1	
Time 1	coot.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
+	t

```
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。

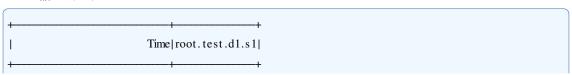
输出序列: 输出单个序列,类型为 DOUBLE,其中每一个数据点的值的范围都是 [0,1]。

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

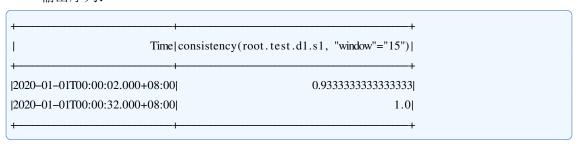
在指定窗口大小的情况下,本函数会把输入数据划分为若干个窗口计算一致性。 输入序列:

```
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。

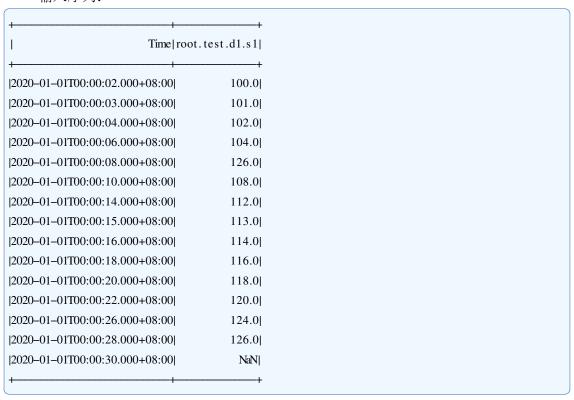
输出序列: 输出单个序列,类型为 DOUBLE,其中每一个数据点的值的范围都是 [0,1]。

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

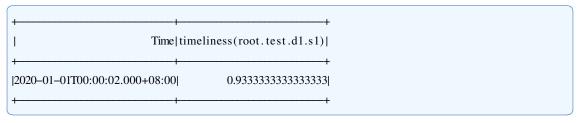
在参数缺省的情况下,本函数将会把全部输入数据都作为同一个窗口计算时效性。 输入序列:



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

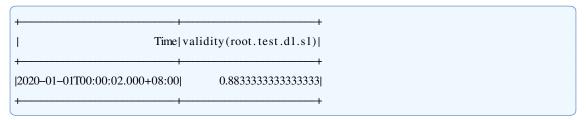
在参数缺省的情况下,本函数将会把全部输入数据都作为同一个窗口计算有效性。 输入序列:

+	-+
Tin	ne 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:	NaN
+	-+

用于查询的 SQL 语句:

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

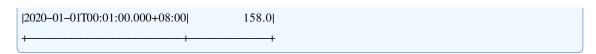
输出序列:



3.4.2.2 指定窗口大小

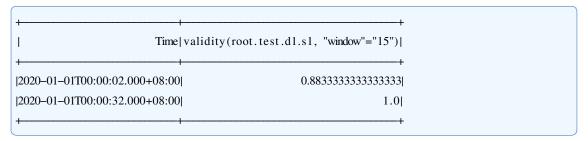
在指定窗口大小的情况下,本函数会把输入数据划分为若干个窗口计算有效性。 输入序列:

+	
Time	root.test.d1.s1
+	
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
- 4.2 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
- 6.2 KSigma
- **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**
- **7.6 AND**