Store aggregate value of a PySpark dataframe column into a variable

I am working with PySpark dataframes here. "test1" is my PySpark dataframe and event_date is a TimestampType. So when I try to get a distinct count of event_date, the result is a integer variable but when I try to get max of the same column the result is a dataframe. I would like to understand what operations result in a dataframe and variable. I would also like to know how to store the max of the event date as a variable

Code that results in an integer type:

```
loop_cnt=test1.select('event_date').distinct().count()
type(loop_cnt)
```

Code that results in dataframe type:

```
last_processed_dt=test1.select([max('event_date')])
type(last_processed_dt)
```

Edited to add a reproducible example:

```
schema = StructType([StructField("event_date", TimestampType(), True)])

df = sqlContext.createDataFrame([(datetime(2015, 8, 10, 2, 44, 15),),(datetime(2015, 8,
10, 3, 44, 15),)], schema)
```

Code that returns a dataframe:

```
last_processed_dt=df.select([max('event_date')])
type(last_processed_dt)
```

Code that returns a varible:

```
loop_cnt=df.select('event_date').distinct().count()
type(loop_cnt)
```

apache-spark pyspark

edited May 2 at 17:46

asked May 2 at 16:45
Sid Mandati
27 3

You should show a reproducible example! – Alberto Bonsanto May 2 at 16:52

2 Answers

I'm pretty sure `df.select([max('event_date')])` returns a DataFrame because there could be more than one row that has the max value in that column. In your particular use case no two rows may have the same value in that column, but it is easy to imagine a case where more than one row can have the same max `event_date` .

`df.select('event_date').distinct().count()` returns an integer because it is telling you how many distinct values there are in that particular column. It does NOT tell you which value is the largest.

If you want code to get the max `event_date` and store it as a variable, try the following `max_date = df.select([max('event_date')]).distinct().collect()`

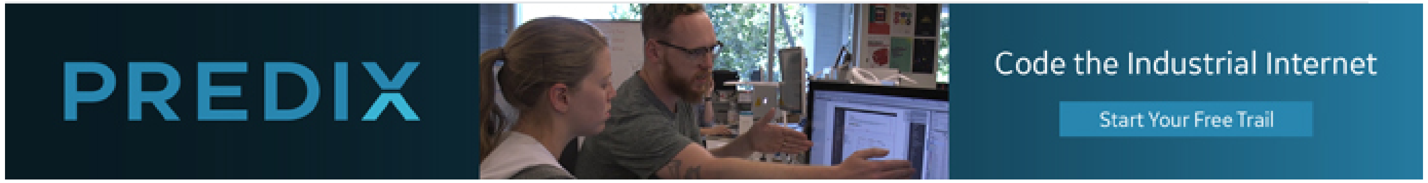
answered May 2 at 20:23
Katya Handler
732 4 23

I tried to using max_date = df.select([max('event_date')]).distinct().collect() and type of the max_date object is a list – Sid Mandati May 13 at 15:13

What does the list look like? – Katya Handler May 13 at 15:14

When I print the list I get [Row(max(event_date)=datetime.datetime(2015, 8, 10, 3, 44, 15))] – Sid Mandati May 13 at 15:25

That's just a single Row object. I'm not sure exactly but try indexing it. Something like `max_date[0][0]` . Index until you find the `datetime` object. – Katya Handler May 13 at 16:04



You cannot directly access the values in a dataframe. Dataframe returns a Row Object. Instead Dataframe gives you a option to convert it into a python dictionary. Go through the following example where I will calculate average wordcount:

```
wordsDF = sqlContext.createDataFrame([('cat',), ('elephant',), ('rat',), ('rat',), ('cat',
)], ['word'])
wordCountsDF = wordsDF.groupBy(wordsDF['word']).count()
wordCountsDF.show()
```

Here are the word count results:

```
+--------+-----+
|    word|count|
+--------+-----+
|     cat|    2|
|     rat|    2|
|elephant|    1|
+--------+-----+
```

Now I calculate the average of count column apply collect() operation on it. Remember collect()

returns a list.Here the list contains one element only.

```
averageCount = wordCountsDF.groupBy().avg('count').collect()
```

Result looks something like this.

```
[Row(avg(count)=1.6666666666666667)]
```

You cannot access directly the average value using some python variable. You have to convert it into a dictionary to access it.

```
results={}
for i in averageCount:
  results.update(i.asDict())
print results
```

Our final results look like these:

```
{'avg(count)': 1.6666666666666667}
```

Finally you can access average value using:

```
print results['avg(count)']

1.66666666667
```

answered Jun 23 at 13:51

sujit
79 7