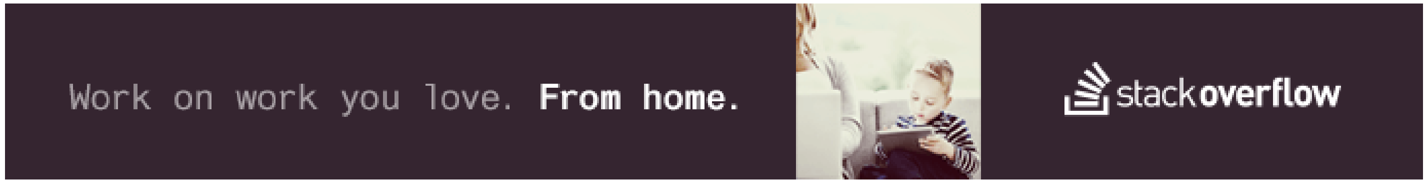
Dividing two columns of a different DataFrames



I am using Spark to do exploratory data analysis on a user log file. One of the analysis that I am doing is average requests on daily basis per host. So in order to figure out the average, I need to divide the total request column of the DataFrame by number unique Request column of the DataFrame.

```
total_req_per_day_df =
logs_df.select('host',dayofmonth('time').alias('day')).groupby('day').count()

avg_daily_req_per_host_df = total_req_per_day_df.select("day",
(total_req_per_day_df["count"] / daily_hosts_df["count"]).alias("count"))
```

This is what I have written using the PySpark to determine the average. And here is the log of error that I get

```
AnalysisException: u'resolved attribute(s) count#1993L missing from day#3628,count#3629L
in operator !Project [day#3628,(cast(count#3629L as double) / cast(count#1993L as double))
AS count#3630];
```

Note: daily_hosts_df and logs_df is cached in the memory. How do you divide the count column of both data frames?

python apache-spark pyspark apache-spark-sql

asked Jun 30 at 15:51
StackPointer
87 1 1 11

To any one who down voted: If you don't mind could you please specify the reason? Because I haven't seen any duplicates of the question. And even if it is a stupid mistake or as such at least guide me, what am I missing out on? – StackPointer Jun 30 at 17:01

4 Answers

It is not possible to reference column from another table. If you want to combine data you'll have to `join` first using something similar to this:

```
from pyspark.sql.functions import col

(total_req_per_day_df.alias("total")
    .join(daily_hosts_df.alias("host"), ["day"])
    .select(col("day"), (col("total.count") / col("host.count")).alias("count")))
```

answered Jun 30 at 18:26
zero323
69.1k 16 82 148

Nothing that can change much here. You can experiment with window functions but I doubt it will have a large impact. – zero323 Jun 30 at 18:34

It's a question from an edX Spark course assignment. Since the solution is public now I take the opportunity to share another, slower one and ask whether the performance of it could be improved or is totally anti-Spark?

```
daily_hosts_list = (daily_hosts_df.map(lambda r: (r[0], r[1])).take(30))
days_with_hosts, hosts = zip(*daily_hosts_list)
requests = (total_req_per_day_df.map(lambda r: (r[1])).take(30))
average_requests = [(days_with_hosts[n], float(l)) for n, l in
enumerate(list(np.array(requests, dtype=float) / np.array(hosts)))]
avg_daily_req_per_host_df = sqlContext.createDataFrame(average_requests, ('day',
'avg_reqs_per_host_per_day'))
```

answered Jul 9 at 19:33

marcindulak
21 3

The whole point of the assignment was to use sparkSQL and your solution has just avoided that. The solution given here would be good enough for the slightly light amount of data but imagine if you were doing this on 500 GB of data. In that case your solution will fail. Also this solution will not be parallel across the cluster of nodes, which is the point of spark. So for a better solution look at the answer that I accepted and the comment that I made on that answer. As it turns out that this is the most efficient solution to the problem as 2 different dataframes can't be divided in SparkSQL. – StackPointer Jul 12 at 15:02

Join the two data frames on column day, and then select the day and ratio of the count columns.

```
total_req_per_day_df = logs_df.select(dayofmonth('time')
                                      .alias('day')
                                     ).groupBy('day').count()

avg_daily_req_per_host_df = (
  total_req_per_day_df.join(daily_hosts_df,
                            total_req_per_day_df.day == daily_hosts_df.day
                           )
  .select(daily_hosts_df['day'],
          (total_req_per_day_df['count']/daily_hosts_df['count'])
           .alias('avg_reqs_per_host_per_day')
          )
  .cache()
)
```

edited Jul 2 at 20:21 answered Jul 2 at 19:25

aatishk
496 4 7

Solution, based on zero323 answer, but correctly works as OUTER join.

```
avg_daily_req_per_host_df = (
  total_req_per_day_df.join(
      daily_hosts_df, daily_hosts_df['day'] == total_req_per_day_df['day'], 'outer'
  ).select(
      total_req_per_day_df['day'],

(total_req_per_day_df['count']/daily_hosts_df['count']).alias('avg_reqs_per_host_per_day')
  )
).cache()
```

Without 'outer' param you lost data for days missings in one of dataframes. This is not critical for PySpark Lab2 task, becouse both dataframes contains same dates. But can create some pain in another tasks :)

answered Jul 19 at 12:48

Andy Al
13 3

Was just wondering about what you said computation wise. Suppose there is one day where there is no value given.... how are you going to calculate an average on that? And technically in Lab2 it doesn't matter because you will not get any data in dataframe2 if there were no data for a given day in dataframe1, if you think carefully about it. So we don't really need outer join here, but thanks for the tips. Will keep in mind in future. – StackPointer Jul 19 at 20:37

Also more over, I think you need to remove the 'null' / 'NaN' value anyways when you are doing arithmetic operations on the data. So again outer join in such situation outer join would break your program, rather then fixing it. You might want to look at how to remove/replace 'NaN' data. There are few techniques either remove them or either use Random Forest to fill them in. Although, I would suggest you to read more about it. So without taking care of 'Nan'/null values you have used outer join here. There is serious bug in your program. However, it is okay for lab 2. – StackPointer Jul 19 at 20:45

Thanks for comment, this is realy bug with NaN/zero. – Andy Al Jul 19 at 21:36