Assignment 2 – Spark & SparkSQL

Due: Wednesday, March 11th at 11:55 PM

NO LATE SUBMISSIONS WILL BE ACCEPTED

THIS ASSIGNMENT IS LONGER THAN THE PREVIOUS ONE. START EARLY!

Goal

In this assignment, you will use Spark and SparkSQL to explore, analyze and identify issues in the NYC Taxi data. The assignment has 4 main tasks, each with a set of subtasks. You will implement all of them first in Spark, then in SparkSQL.

Instruction

You will submit 2 sets of code: one set in Spark (Python) and the other in SparkSQL. We will run both sets of your codes with our own datasets to check your codes and outputs.

You must use Spark 2.4.0 and Python 3.6.5.

Sort all results in ascending order unless stated otherwise.

Data

You will use 3 datasets in Dumbo HDFS under the **/user/hc2660/hw2data** directory:

```
Fares.csv  Trips.csv  Licenses.csv
```

To stream a CSV file in core Spark, thus reading it into an *RDD*, you would use, e.g.,

```
from csv import reader
lines = sc.textFile(sys.argv[1], 1)
lines = lines.mapPartitions(lambda x: reader(x))
```

To do this in SparkSQL, thus reading the data into a *DataFrame*, you would use, e.g.,

```
df = spark.read.format('csv').options(header='true',
     inferschema='true').load(sys.argv[1])
```

HINT: You may always use the smaller sets of data from assignment 1 to debug your codes.

Submission Instruction (-10 points if unfollowed)

Submit a .zip file named with your netID (e.g., hc2660.zip). The zip file includes 26 python files:

task1a.py, task1b.py, task1a-sql.py, task1b-sql.py
task2a.py, task2b.py, task2c.py, task2d.py, task2a-sql.py, task2b-sql.py, task2c-sql.py, task2d-sql.py
task3a.py, task3b.py, task3c.py, task3d.py, task3a-sql.py, task3b-sql.py, task3c-sql.py, task3d-sql.py
task4a.py, task4b.py, task4c.py, task4a-sql.py, task4b-sql.py, task4c-sql.py

Submit & Execute Spark Jobs on Dumbo

For each task, the script should read in the path to the input file on HDFS from the command line arguments. You should use the following commands to submit your jobs – make sure to order the inputs as directed (we will use the same commands, but with different data instances):

Task 1a/Task 1a-SQL

```
spark-submit --conf
spark.pyspark.python=/share/apps/python/3.6.5/bin/python
{your_netid}/task1a.py(task1a-sql.py) /user/hc2660/hw2data/Trips.csv
/user/hc2660/hw2data/Fares.csv
```

Task 1b/Task 1b-SQL

```
spark-submit --conf
spark.pyspark.python=/share/apps/python/3.6.5/bin/python
{your_netid}/task1b.py(task1b-sql.py) /user/hc2660/hw2data/Fares.csv
/user/hc2660/hw2data/Licenses.csv
```

Make sure your task1 outputs are named

task1a.out/task1a-sql.out/task1b.out/task1b-sql.out

And are *getmerged* into one file by using

hfs –getmerge task1a.out task1a.out

Then *put back* in HDFS *for task 2 to task 4* by using

hfs –put task1a.out

Task 2x/Task 2x-SQL. (x represents subtasks in task 2)

```
spark-submit --conf
spark.pyspark.python=/share/apps/python/3.6.5/bin/python
{your_netid}/task2x.py(task2x-sql.py)
/user/{your_netid}/task1a.out(task1a-sql.out)
```

Task 3x/Task 3x-SQL.

```
spark-submit --conf
spark.pyspark.python=/share/apps/python/3.6.5/bin/python
{your_netid}/task3x.py(task3x-sql.py)
/user/{your_netid}/task1a.out(task1a-sql.out)
```

Task 4x/Task 4x-SQL

```
spark-submit --conf
spark.pyspark.python=/share/apps/python/3.6.5/bin/python
{your_netid}/task4x.py(task4x-sql.py)
/user/{your_netid}/task1b.out(task1b-sql.out)
```

Remarks

- For this portion, you **do not** need to include output files in your submission.
- You should also **not** include any input files in your submission.
- Your code should output a directory named “taskx.out” or “taskx-sql.out” to HDFS, i.e., use the Spark RDD function `saveAsTextFile("taskx.out")` or the Spark DataFrame function `save("taskx-sql.out",format="text")` rather than python I/O. *(In order for the hw1tester script to work, you must name your output directories “taskx.out” and ”taskx-sql.out” where x is in 1a through 4c).*

Output Instruction

For simplicity, all attributes output in this homework are *comma separated*. (No need to separate key and values by tab and No header needed) This applies for both Spark tasks outputs and SparkSQL tasks outputs.

Task 1a sample output

```
00005007A9F30E289E760362F69E4EAD,2C584442C9DC6740767CDE5672C12379,CMT,2013-08-07 00:55:11,1,N,2013-08-07 00:25:38,1,990,8.9,-73.981972,40.764397,-73.927887,40.865353,CRD,26.5,0.5,0.5,5.5,0,33
```

Task 2a sample output

```
0,5,123513
5,10,60358
```

Task 2b sample output

```
0,41
1,1691674
```

Task 3 sample output

```
10
```

Hint:

- **For core Spark: You may find using a final `map()` stage is helpful for formatting your output correctly. E.g., if output of reduce is named 'result', you could write:**

```
output=result.map(lambda r: ', '.join([KVPair for KVPair in r]))
output.saveAsTextFile('task1a.out')
```

- **For SparkSQL: Using a final `select()` which produces a single column using `format_string()` is helpful for formatting output and writing to a text file. E.g., if result in task 2-b is the dataframe that contains the query is named 'result', you could write:**

```
result.select(format_string('%d, %d', result.num_of_passengers, result.num_trips)).write.save('task2b-sql.out',format="text")
```

Tasks

Task 1: Inner Joins

a) Output all information related to the taxi trips (AllTrips) by performing an inner join of the 'Trips' and 'Fares' datasets.

These tables share 4 attributes: medallion, hack_license, vendor_id, pickup_datetime

Column order: medallion, hack_license, vendor_id, pickup_datetime
remaining attributes of Trips
remaining attributes of Fares

Sort rows by medallion, hack_license, pickup_datetime

b) Output all information related to the taxi fares and licenses by performing an inner join of the 'Fares' and 'Licenses' datasets.

Note that these two tables share only 1 attributes: medallion

Columns order: columns order in Fares, then remaining attributes Licenses

Sort rows by medallion, hack_license, pickup_datetime

Task 2: Analysis

For the following subtasks, use AllTrips.

a) Find the distribution of the amounts charged for taxi trips, i.e., for each range below, find the number of trips whose fare amounts (fare_amount) that fall in that range.

[0, 5] (5, 15] (15, 30] (30, 50] (50, 100] [>100)

Output schema: amount_range, num_trips

Sort: range

b) Find the distribution of the number of passengers, i.e., for each X number of passengers, find the number of trips that had X passengers.

Schema: num_of_passengers, num_trips

Sort: num_of_passengers

c) For *each* day (YYYY-MM-DD), find the total revenue (fare amount + tips + surcharges) and the total tolls. Use 2 decimal places.

Schema: date, total_revenue, total_tolls

Sort: date

d) For *each* taxi (medallion), find the *total* number of trips, and the *average* number of trips *per day* that the taxi was driven. Use 2 decimal places.

Schema: medallion, total_trips, days_driven, average

Sort: medallion

Task 3: Identify Data Issues

For the following subtasks, use AllTrips.

a) Are there trips with invalid fare amounts (less than 0)? If so, how many?

Schema: Number of trips with invalid fare amounts

b) Is there more than one record for a given taxi at the same time?

Schema: medallion, pickup_datetime

Sort: medallion, pickup_datetime

c) For each taxi, what is the percentage of trips without GPS coordinates (all 4 coordinates are recorded as 0's)?

Schema: medallion, percentage_of_trips

Sort: medallion

d) Find the number of different taxis (medallion) used by each driver (license).

Schema: hack_license, num_taxis_used

Sort: hack_license

Task 4: Analysis: Drivers and Vehicles

The Licenses dataset contains information about the actual vehicles used in a trip.

total_revenue = $\sum_{i=1}^{n} fare\_amount_i$

avg_tip_percentage = $\frac{100}{n}\sum_{i=1}^{n}\frac{fare\_amount_i}{tip\_amount_i}$

a) Compare trips based on vehicle_type (WAV, HYB, CNG, LV1, DSE, NRML).

Schema: vehicle_type, total_trips, total_revenue, avg_tip_percentage

Sort: vehicle_type

b) Compare trips based on medallion_type (NAMED DRIVER, OWNER MUST DRIVE).

Schema: medallion_type, total_trips, total_revenue, avg_tip_percentage

Sort: medallion_type

c) List the top 10 agents by total revenue.

Schema: agent_name, total_revenue

Sort: total_revenue (in descending order), agent_name

The End