Q1 [15 points] Analyzing trips data with PySpark

Technology	PySpark, Docker
Allowed Libraries	NA
Max allowed runtime	NA
Deliverables	[Gradescope] **q1.ipynb:** your solution as a Jupyter Notebook file

Imagine that your boss gives you a large dataset which contains trip information of New York City Taxi and Limousine Commission (TLC). You are asked to provide summaries for the most common trips, as well as information related to fares and traffic. This information might help in positioning taxis depending on the demand at each location.

Follow these instructions to download and set up a preconfigured Docker image that you will use for this assignment.

> **Why use Docker?** In earlier iterations of this course, students installed software on their own machines, and we (both students and instructor team) ran into many issues that could not be resolved satisfactorily. Docker allows us to distribute a cross-platform, preconfigured image with all the requisite software and correct package versions. Once Docker is installed and the container is running, access Jupyter by browsing to http://localhost:6242. There is no need to install any additional Java or PySpark dependencies as they are all bundled as part of the Docker container.

Imagine that your boss gives you a large dataset which contains trip information of New York City Taxi and Limousine Commission (TLC). You are asked to provide summaries for the most common trips, as well as information related to fares and traffic. This information might help in positioning taxis depending on the demand at each location.

You are provided with a Jupyter notebook (q1.ipynb) file which you will complete using PySpark using the provided Docker image.

Note:
1. Regular PySpark Dataframe Operations and PySpark SQL operations can be used.
2. If you re-run cells, remember to **restart the kernel** to clear the Spark context, otherwise an existing Spark context may cause errors.
3. Be sure to save your work often! If you do not see your notebook in Jupyter, then double check that the file is present in the folder and that your Docker has been set up correctly. If, after checking both, the file still does not appear in Jupyter then you can still move forward by clicking the “upload” button in the Jupyter notebook and uploading the file – however, if you use this approach, then your file will **not** be saved to disk when you save in Jupyter, so you would need to download your work by going to File > Download as... > Notebook (.ipynb), so be sure to download often to save your work!

Tasks

You will use the yellow_tripdata_2019-01_short.csv dataset. This dataset is a modified record of the NYC Green Taxi trips and includes information about the pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, fare amounts, payment types, and driver-reported passenger counts. When processing the data or performing calculations, **do not round any values**.

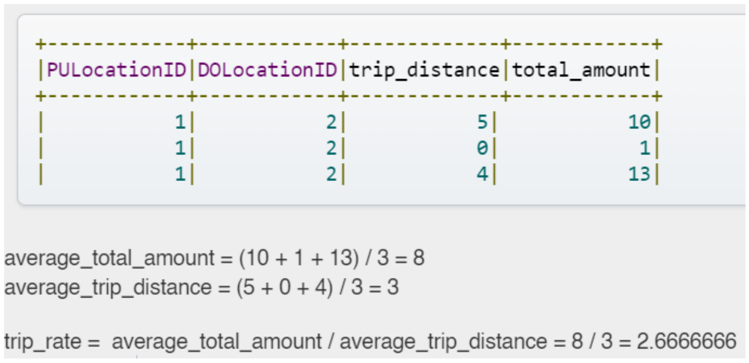
a. **[1 pt]** You will be modifying the function `clean_data` to clean the data. Cast the following columns into the specified data types:

a. `passenger_count` — integer
b. `total_amount` — float
c. `tip_amount` — float
d. `trip_distance` — float
e. `fare_amount` — float
f. `tpep_pickup_datetime` — timestamp
g. `tpep_dropoff_datetime` — timestamp

b. **[4 pts]** You will be modifying the function `common_pair`. Return the top 10 pickup-dropoff location pairs having the highest number of trips (`count`). The location pairs should be ordered by `count` in descending order. If two or more pairs have the same number of trips, break the tie using the trip amount per distance traveled (`trip_rate`) in descending order. Use columns `total_amount` and `trip_distance` to calculate the trip amount per distance. In certain situations, the pick-up and drop-off locations may be the same (include such entries as well).

While calculating trip_rate, first get the average `trip_distance` and the average `total_amount` for each pair of `PULocationID` and `DOLocationID` (using group by). Then take their ratio to get the `trip_rate` for a pickup-drop pair.

Example:



```
+------------+------------+-------------+------------+
|PULocationID|DOLocationID|trip_distance|total_amount|
+------------+------------+-------------+------------+
|           1|           2|            5|          10|
|           1|           2|            0|           1|
|           1|           2|            4|          13|
```

average_total_amount = (10 + 1 + 13) / 3 = 8
average_trip_distance = (5 + 0 + 4) / 3 = 3

trip_rate = average_total_amount / average_trip_distance = 8 / 3 = 2.6666666

Sample Output Format (values are examples *only*):

PULocationID	DOLocationID	Count	trip_rate
1	2	23	5.242345
3	3	5	6.61345634

c. **[4 pts]** You will be modifying the function `time_of_cheapest_fare.` Divide each day into two periods: Day (from 9am to 8:59:59pm, both inclusive), and Night (from 9pm to 8:59:59am, both inclusive). Calculate the average total amount per unit distance traveled (use column `total_amount`) for both time periods. Sort the result by `trip_rate` in ascending order to determine when the fare rate is the cheapest. Use `tpep_pickup_datetime` to divide trips into Day and Night.

Output:

day_night	trip_rate
Day	4.2632344561
Night	6.42342882

d. **[4 pts]** You will be modifying the function `passenger_count_for_most_tip`. Filter the data for trips having fares (`fare_amount`) greater than $2 and the number of passengers (`passenger_count`) greater than 0. Calculate the average fare and tip (`tip_amount`) for all passenger group sizes and calculate the tip percent (`tip_amount * 100 / fare_amount`). Sort the result in descending order of tip percent to obtain the group size that tips the most generously.

Output:

passenger_count	tip_percent
2	14.22345234
1	12.523334576
3	12.17345231

e. **[3 pts]** You will be modifying the function `day_with_traffic` . Sort the days of the week (using `tpep_pickup_datetime`) in descending order of traffic (day having the highest traffic should be at the top). Calculate traffic for a particular day using the average speed of all taxi trips on that day of the week. Calculate the average speed as the average trip distance divided by the average trip time, as distance per hour. If the `average_speed` is equal for multiple days, order the days alphabetically. A day with low average speed indicates high levels of traffic. The average speed may be 0, indicating very high levels of traffic. Not all days of the week may be present in the data (do not include the missing days of the week in your output). Use `date_format` along with the appropriate pattern letters to format the day of the week so that it matches the example output below.

Output:

day_of_week	average_speed
Fri	0.953452345
Mon	5.2424622
Tue	9.23345272

IMPORTANT: Strictly follow the requirements below, or your answers may not be graded.

1. Do not add any cells to the notebook.
2. Remove all "testing" code that renders output, or Gradescope will crash. For instance, any additional print, display, and show statements used for debugging must be removed.

Q2 [30 pts] Analyzing dataset with Spark/Scala on Databricks

Technology	Spark/Scala, Databricks
Allowed Libraries	NA
Max allowed runtime	NA
Deliverables	[Gradescope] • **q2.dbc:** Your solution as Scala Notebook archive file (.dbc) exported from Databricks (see Databricks Setup Guide below) • **q2.scala**: Your solution as a Scala source file exported from Databricks (see Databricks Setup Guide below) • **q2_results.csv:** The output results from your Scala code in the Databricks q2 notebook file. You must carefully copy the outputs of the *display()/show()* function into a file titled ***q2_results.csv*** under the relevant sections. Please double-check and compare your actual output with the results you copied.

Tutorial

Firstly, go over this Spark on Databricks Tutorial, to learn the basics of creating Spark jobs, loading data, and working with data.

You will analyze nyc-tripdata.csv[1] using Spark and Scala on the Databricks platform. (A short description of how Spark and Scala are related can be found here.) You will also need to use the taxi zone lookup table using taxi_zone_lookup.csv that maps the location ID into the actual name of the region in NYC. The nyc-trip data dataset is a modified record of the NYC Green Taxi trips and includes information about the pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, fare amounts, payment types, and driver-reported passenger counts.

VERY IMPORTANT

1. Use only **Firefox, Safari or Chrome** when configuring anything related to Databricks. The setup process has been verified to work on these browsers.
2. **Carefully** follow the instructions in the Databricks Setup Guide. (You should have already downloaded the data needed for this question using the link provided before *Homework Overview*.)
    a. You **must** choose the Databricks Runtime (DBR) version as "**6.4 (includes Apache Spark 2.4.5, Scala 2.11)**". We will grade your work using this version.
    b. You **must not** choose the default DBR version of >= 7.2
    c. Note that you do not need to install Scala or Spark on your local machine. They are provided with the DBR environment.
3. **You must use only** Scala DataFrame operations for this question. Scala DataFrames are just another name for Spark DataSet of rows. You can use the DataSet API in Spark to work on these DataFrames. Here is a Spark document that will help you get started on working with DataFrames in Spark. You will lose points if you use SQL queries, Python, or R to manipulate a DataFrame.
    a. After selecting the default language as SCALA, do not use the language magic %<language> with other languages like %r, %python, %sql etc. The language magics are used to override the default language, which you **must not** do for this assignment.
    b. You **must not** use full SQL queries in lieu of the Spark DataFrame API. That is, you **must not** use functions like *sql()*, which allows you to directly write full SQL queries like spark.sql ("SELECT* FROM col1 WHERE …"). This should be df.select("*") instead.
4. The template Scala notebook *q2.dbc* (in hw3-skeleton) provides you with code that reads a data file *nyc-tripdata.csv.* The input data is loaded into a DataFrame, inferring the schema using reflection (Refer to the Databricks Setup Guide above). It also contains code that **filters the data** to only keep the rows where the pickup location is different from the drop location, and the trip distance is strictly greater than 2.0 (>2.0).
    a. All tasks listed below **must** be performed on this filtered DataFrame, or you will end up with wrong answers.
    b. Carefully read the instructions in the notebook, which provides hints for solving the problems.

5. Some tasks in this question have specified data types for the results that are of lower precision (e.g., float). For these tasks, we will accept relevant higher precision formats (e.g., double). Similarly, we will accept results stored in data types that offer "greater range" (e.g., long, bigint) than what we have specified (e.g., int).

[1] Graph derived from the NYC Taxi and Limousine Commission

6. Remove all “testing” code that renders output, or Gradescope will crash. For instance, any additional print, display, and show statements used for debugging must be removed.

Tasks

1) List the top-5 most popular locations for:
   a. **[2 pts]** dropoff based on "DOLocationID", sorted in descending order by popularity. If there is a tie, then one with a lower "DOLocationID" gets listed first.
   b. **[2 pts]** pickup based on "PULocationID", sorted in descending order by popularity. If there is a tie, then one with a lower "PULocationID" gets listed first.
2) **[4 pts]** List the top-3 locationID’s with the maximum overall activity. Here, overall activity at a LocationID is simply the sum of all pick-ups and all drop-offs at that LocationID. In case of a tie, the lower LocationID gets listed first.

   Note: If a taxi picked up 3 passengers at once, we count it as 1 pickup and not 3 pickups.

3) **[4 pts]** List all the boroughs (of NYC: Manhattan, Brooklyn, Queens, Staten Island, Bronx along with "Unknown" and "EWR") and their total number of activities, in descending order of a total number of activities. Here, the total number of activities for a borough (e.g., Queens) is the sum of the overall activities (as defined in part 2) of all the LocationIDs that fall in that borough (Queens). An example output format is shown below.

```
+-------------+----------------------+
|      Borough|total_number_activities|
+-------------+----------------------+
|        Bronx|                 17689|
|     Brooklyn|                 15785|
|      Unknown|                  1982|
|Staten Island|                  1092|
|    Manhattan|                   521|
|          EWR|                   200|
|       Queens|                    30|
+-------------+----------------------+
```

4) **[5 pts]** List the top 2 days of the week with the largest number of daily average pick-ups, along with the average number of pick-ups on each of the 2 days in descending order (no rounding off required). Here, the average pickup is calculated by taking an average of the number of pick-ups on different dates falling on the same day of the week. For example, *02/01/2021*, *02/08/2021* and *02/15/2021* are all Mondays, so the average pick-ups for these is the sum of the pickups on each date divided by 3. An example output is shown below.

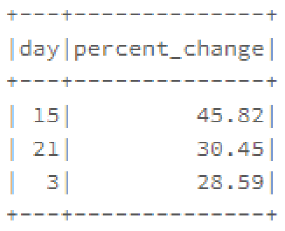
```
+-----------+---------+
|day_of_week|avg_count|
+-----------+---------+
|     Monday| 17231.82|
|   Saturday| 14901.45|
+-----------+---------+
```

   Note: The day of week is a string of the day’s full spelling, e.g., "Monday" instead of the number 1 or "Mon". Also, the pickup_datetime is in the format: yyyy-mm-dd

5) **[6 pts]** For each hour of a day (0 to 23, 0 being midnight) — in the order from 0 to 23 (inclusively), find the zone in the Brooklyn borough with the **largest** number of total pick-ups.

Note: All dates for each hour should be included.

6) **[7 pts]** Find which 3 different days in the month of January, in Manhattan, that saw the largest positive percentage increase in pick-ups compared to the previous day, in the order from largest percentage increase to smallest percentage increase. An example output is shown below.



```
+---+--------------+
|day|percent_change|
+---+--------------+
| 15|         45.82|
| 21|         30.45|
|  3|         28.59|
+---+--------------+
```

Note: All years need to be aggregated to calculate the pickups for a specific day of January. The change from Dec 31 to Jan 1 can be excluded.

List the results of the above tasks in the provided **q2_results.csv** file under the relevant sections. **These pre-formatted sections also show you the required output format from your Scala code with the necessary columns — while column names can be different, their resulting values must be correct.**

- You must **manually enter** the output generated into the corresponding sections of the *q2_results.csv* file, preferably using some spreadsheet software like MS-Excel (but make sure to keep the csv format). For generating the output in the Scala notebook, refer to `show()` and `display()` functions of Scala.
- Note that you can edit this csv file using text editor, but please be mindful about putting the results under designated columns.

Note: Do **NOT** modify anything other than filling in those required output values in this csv file. **We grade by running the Spark Scala code you write and by looking at your results listed in this file**. So, make sure that your output is actually obtained from the Spark Scala code you write.

Hint: You may find some of the following DataFrame operations helpful:
`toDF, join, select, groupBy, orderBy, filter, agg, Window(), partitionBy, orderBy, etc.`

Q3 [35 points] Analyzing Large Amount of Data with PySpark on AWS

Technology	PySpark, AWS
Allowed Libraries	NA
Max allowed runtime	NA
Deliverables	[Gradescope] • **q3.ipynb**: PySpark notebook for this question (for the **larger** dataset). • **q3_output_large.csv**: output file **(comma-separated)** for the **larger** dataset.

VERY IMPORTANT: Use Firefox, Safari or Chrome when configuring anything related to AWS.

You will try out PySpark for processing data on Amazon Web Services (AWS). Here you can learn more about PySpark and how it can be used for data analysis. You will be completing a task that may be accomplished using a commodity computer (e.g., consumer-grade laptops or desktops). However, we would like you to use this exercise as an opportunity to learn distributed computing on Amazon EC2, and to gain experience that will help you tackle more complex problems.

The services you will primarily be using are Amazon S3 storage, Amazon Elastic Cloud Computing (EC2) virtual servers, and Amazon Elastic MapReduce (EMR) managed Hadoop framework. You will be creating an S3 bucket, running code through EMR, and then storing the output into that S3 bucket.

For this question, you will only use up a very small fraction of your AWS credit.

If you have any issues with the AWS Academy account, please fill out this form.

Setting Up AWS Environment

Go through all the steps in the AWS Setup Guide (You should have already completed Step 1 to create your account) to set up your AWS environment, e.g., setting up billing alert, creating S3 storage bucket, uploading skeleton file, and, **EXTREMELY IMPORTANTLY learning how to terminate all AWS clusters properly, or you will run out of AWS credits and may not be able to complete this question**.

Datasets

In this question, you will use a dataset of trip records provided by the New York City Taxi and Limousine Commission (TLC). You will be accessing the dataset directly through AWS via the code outlined in the homework skeleton. Specifically, you will be working with two samples of this dataset, one small, and one much larger. Further details about this dataset are available here and here, and you may explore the structure of the data via [1] [2].

EXTREMELY IMPORTANT: Both the datasets are in the **US East (N. Virginia)** region. Using machines in other regions for computation will incur data transfer charges. Hence, set your region to **US East (N. Virginia)** in the beginning (not Oregon, which is the default). **This is extremely important, otherwise your code may not work, and you may be charged extra.**

Goal

You work at NYC TLC, and since the company bought a few new taxis, your boss has asked you to locate potential places where taxi drivers can pick up more passengers. Of course, the more profitable the locations are, the better. Your boss also tells you not to worry about short trips for **any** of your analysis, so only analyze trips which are **2.0 miles or longer**.

First, find the **20** most popular drop off locations in the Manhattan borough by finding which of these destinations had the greatest **passenger count**.

Now, analyze all pick-up locations.

- For each pick-up location determine
  - the **average total amount** per trip,
  - the total **count** of all trips that start at that location, and
  - the **count** of all trips that start at that location and end at one of most popular drop-off locations.
- Using the above values,
  - determine the **proportion** of trips that end in one of the popular drop-off locations (# trips that end in drop off location divided by total # of trips) and
  - multiply that proportion by the **average total amount** to get a **weighted profit value** based on the probability of passengers going to one of the popular destinations.

Bear in mind, your boss is not as savvy with the data as you are and is not interested in location IDs. To make it easy for your boss, provide the **Borough** and **Zone** for each of the top 20 pick-up locations you determined.

Tasks

You are provided with a python notebook (q3.ipynb) file which you will complete and load into EMR. You are provided with the load_data() function, which loads two PySpark DataFrames. The first is **trips** which contain a DataFrame of trip data, where each record refers to one (1) trip. The second is **lookup** which maps a LocationID to its information. It can be linked to either the PULocationID or DOLocationID fields in the trips DataFrame.

The following functions must be completed for full credit.

VERY IMPORTANT

- Ensure that the parameters for each function remain as defined and the output order and names of the fields in the PySpark DataFrames are maintained.
- Do not import any functions which were not already imported within the skeleton.
- **You must NOT round any numeric values**. Rounding numbers can introduce inaccuracies. Our grader will be checking the first 8 decimal places of each value in the DataFrame.

a) **[1 pts] user()**
   i. Returns your GT Username as a string (e.g., gburdell3)

b) **[2 pts] long_trips(trips)**
   i. This function filters trips to keep only trips 2 miles or longer (e.g., >= 2).
   ii. Returns PySpark DataFrame with the same schema as **trips**
   iii. **Note: Parts c, d and e will use the result of this function**

c) **[6 pts] manhattan_trips(trips, lookup)**
   i. This function determines the top 20 locations with a *DOLocationID* in Manhattan by sum of passenger count.
   ii. Returns a PySpark DataFrame (mtrips) with the schema (DOLocationID, pcount)

d) **[6 pts] weighted_profit(trips, mtrips)**
   i. This function determines
      i. the average *total_amount*,
      ii. the *total count of trips*, and
      iii. the *total count of trips ending in the top 20 destinations* and return the *weighted_profit* as discussed earlier in the homework document.
      iv. Returns a PySpark DataFrame with the schema (PULocationID, weighted_profit) for the *weighted_profit* as discussed earlier in this homework document.

e) **[5 pts] final_output(wp, lookup)**
   i. This function
      i. takes the results of *weighted_profit*,
      ii. links it to the *borough* and *zone* through the **lookup** data frame, and
      iii. returns the top 20 locations with the highest *weighted_profit*.
   ii. Returns a PySpark DataFrame with the schema (Zone, Borough, weighted_profit)

Once you have implemented all these functions, run the main() function, which is already implemented, and update the line of code to include the name of your output s3 bucket and a location. **This function will fail** if the output directory already exists, so make sure to **change it each time** you run the function.

Example: `final.write.csv('s3://cse6242-gburdell3/output-large3')`

Your output file will appear in a folder in your s3 bucket as a csv file with a name which is similar to *part-0000-4d992f7a-0ad3-48f8-8c72-0022984e4b50-c000.csv*. Download this file and rename it to **q3_output_large.csv** for submission. Do **NOT** make any other changes to the file.

Hints:

1. Refer to DataFrame commands such as filter, join, groupBy, agg, limit, sort, withColumnRenamed and withColumn. Documentation for the DataFrame APIs is located here.
2. Testing on a single, small dataset (i.e., a "test case") is helpful, and is **insufficient** in discovering all potential issues, especially if such issues only become apparent when the code is run on larger datasets. Thus, it is important for you to develop more ways to review and verify your code logic.
3. Precision in data analytics is very important. Keep in mind that **precision reduction in an earlier step can accumulate and be magnified**, subsequently significantly affecting the final output's precision (e.g., for a dataset with 1,000,000 data points, a 0.0001 difference for each data point can lead to a total difference of 100 over the whole dataset).
4. Check if you're reducing the precision (or "scale") too aggressively. Can you relax the restriction during intermediate steps?
5. Make sure you return a DataFrame. If you get NoneType errors, you are most likely not returning what you think you are.
6. Some columns may need to be cast to the right data type. Keep that in mind!

IMPORTANT: Strictly follow the guidelines below, or your answer may not be graded.

1. Double check that you are submitting the correct files — we only want the script and output from the larger dataset. Also, double check that you are writing the right dataset's output to the right file.
2. You are welcome to store your script's output in any bucket you choose, as long as you can download and submit the correct files.
3. Do not make any manual changes to the output files.
4. Regular Pyspark Dataframe Operations and PySpark SQL operations can be used.
   4.1. To use PySpark SQL operations, you must use the SQL Context on the Spark Dataframe. Example: df.sql_ctx.sql("SELECT * FROM some_data")
5. Do not import any additional packages, INCLUDING pyspark.sql.functions, as this may cause the autograder to work incorrectly. Everything you need should be imported for you.
6. Remove all "testing" code that renders output, or Gradescope will crash. For instance, any additional print, display, and show statements used for debugging must be removed.

Q4 [10 points] Analyzing a Large Dataset using Spark on GCP

Technology	Spark, Google Cloud Platform (GCP)
Allowed Libraries	NA
Max allowed runtime	NA
Deliverables	[Gradescope] **q4.ipynb:** the PySpark notebook for this question.

VERY IMPORTANT: Use Firefox, Safari or Chrome when configuring anything related to GCP.

GCP Guidelines

Instructions to set up GCP Credits, GCP Storage and Dataproc Cluster are provided as video tutorials (part 1, part 2, and part 3) and as written instructions.

Helpful tips/FAQs for special scenarios:

a) If GCP service is disabled for your google account, try the steps in this google support link
b) If you have any issues with GCP free credits, please fill out this form

Goal

The goal of this question is to familiarize you with creating storage buckets/clusters and running Spark programs on Google Cloud Platform. This question asks you to create a new Google Storage Bucket and load the NYC Taxi & Limousine Commission Dataset. You are also provided with a Jupyter Notebook q4.ipynb file, which you will load and complete in a Google Dataproc Cluster. Inside the notebook, you are provided with the skeleton for the load_data() function, which you will complete to load a PySpark DataFrame from the Google Storage Bucket you created as part of this question. Using this PySpark DataFrame, you will complete the following tasks using Spark DataFrame functions.

You will use the data file yellow_tripdata09-08-2021.csv; the preceding link allows you to download the dataset you are required to work with for this question from the course DropBox. Each line represents a single taxi trip consisting of the comma-separated columns bulleted below. All columns are of string data type. You must convert the highlighted columns below into decimal data type **(do NOT use float datatype)** inside their respective functions when completing this question. **Do not convert any datatypes within the load_data function.** While casting to a decimal datatype, use a precision of 38 and a scale of 10.

- vendorid
- tpep_pickup_datetime
- tpep_dropoff_datetime
- passenger_count
- **trip_distance (decimal data type)**
- ratecodeid
- store_and_fwd_flag
- pulocationid
- dolocationid
- payment_type
- **fare_amount (decimal data type)**
- extra
- mta_tax
- **tip_amount (decimal data type)**
- **tolls_amount (decimal data type)**
- improvement_surcharge
- total_amount

Tasks

VERY IMPORTANT: you must first perform the task *a* BEFORE performing task *b, c, d, e and f*. No points are allocated to task a, but it is essential that you correctly implement the load_data() function as the remaining graded tasks depend upon this task and its correct implementation.

a) **[0 pts — required]** Function *load_data()* to load data from a Google Storage Bucket into a Spark DataFrame
b) **[2 pts]** Function *exclude_no_pickuplocations()* to exclude trips with no pick-up locations (i.e., pick-up location id column is null or is zero. In other words, assume zero is not a valid pickup location id.) in the original data from a.
c) **[2 pts**] Function *exclude_no_tripdistance()* to exclude trips with no distance (i.e., trip distance column is null or zero) in the dataframe output by *exclude_no_pickuplocations()*. . Note: Cast the trip_distance column to decimal datatype before filtering.
d) **[2 pts]** Function *include_fare_range()* to include trips with fare from $20 (inclusively) to $60 (inclusively) in the dataframe output by *exclude_no_tripdistance()*. Note: Cast the fare_amount column to decimal datatype before filtering.
e) **[2 pts]** Function *get_highest_tip()* to identify the highest tip (rounded to 2 decimal places) in the dataframe output by *include_fare_range()*. Note: Cast the tip_amount column to decimal datatype before filtering.
f) **[2 pts]** Function *get_total_toll()* to calculate the total toll amount (rounded to 2 decimal places) in the dataframe output by *include_fare_range()*. Note: Cast the tolls_amount column to decimal datatype before filtering.

IMPORTANT: Strictly follow the guidelines below, or your answer may not be graded.
1. Regular PySpark Dataframe Operations and PySpark SQL operations can be used.
2. Make sure to download the notebook from your GCP cluster **before** deleting the GCP cluster (otherwise, you will lose your work).
3. Do not add new cells to the notebook, as this may break the auto-grader.
4. Remove all “testing” code that renders output, or Gradescope will crash. For instance, any additional print, display, and show statements used for debugging must be removed.

Q5 [10 points] Regression: Automobile price prediction, using Microsoft Machine Learning Studio

Note: Create and use a free workspace instance on Microsoft Machine Learning Studio. Use your Georgia Tech username (e.g., jdoe3) to login.

Goal

The primary purpose of this question is to introduce you to Microsoft Machine Learning Studio, familiarize you to its basic functionalities and typical machine learning workflows. Go through the “Automobile price prediction” tutorial and create/run ML experiments to complete the following tasks. You will not incur any cost if you save your experiments on Azure till submission. Once you are sure about the results and have reported them, feel free to delete your experiments.

Technology	Microsoft Machine Learning Studio
Allowed Libraries	NA
Max allowed runtime	NA
Deliverables	[Gradescope] **q5_results.csv**: a csv file containing results for all parts

Tasks

You will manually modify the given file **q5_results.csv** by adding to it the results from the following tasks (e.g., using a plain text editor). Your solution will be autograded. Hence,

- DO NOT change the order of the questions.
- Report the exact numerical values that you get in your output, and **DO NOT round any of them.**
- When manually entering a value into the csv file, append it immediately after a comma, so there will be NO space between the comma and your value, and no trailing spaces or commas after your value.
- Follow the tutorial and do not change values for L2 regularization. For parts b and c, please select the columns given in the tutorial.

a) Update your GT username in the q5_results.csv file to replace gburdell3.

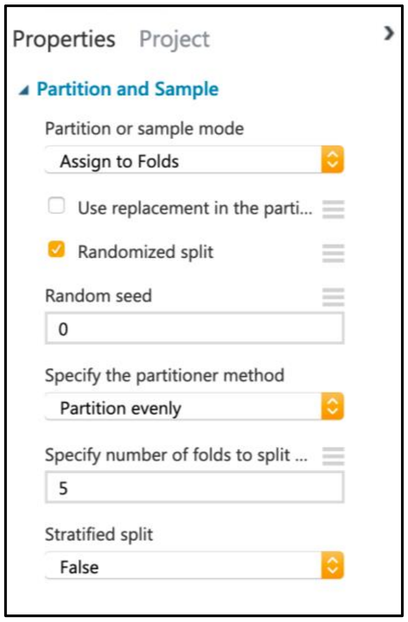
b) **[3 pts]** Repeat the experiment described in the tutorial and report values of all metrics as mentioned in the '*Evaluate Model'* section of the tutorial.

c) **[3 pts]** Repeat the experiment mentioned in part b with a different value of '*Fraction of rows in the first output'* in the split module. Change the value to 0.8 from the originally set value, 0.75. Report corresponding values of the metrics.

d) **[4 pts]** Run a new experiment — evaluate the model using 5-fold cross-validation (CV). Select parameters in the module *'Partition and sample'* (Partition and Sample) in accordance with the figure below. Set the column name as "price" for CV. Also, use 0 as a random seed. Report the values of Root Mean Squared Error (RMSE) and Coefficient of Determination for each of the five folds (1st fold corresponds to fold number 0 and so on). Do NOT round the results. Report exact values.

   To summarize, for part d, you **MUST** exactly follow each step below to run the experiment:

   A. Import the entire dataset (Automobile Price Data (Raw))
   B. Clean the missing data by dropping rows with missing values (select all columns in the dataset and do not "exclude the normalized losses" from the original tutorial). Leave the maximum missing value ratio to 1.
   C. Partition and sample the data. (**Note**: do not use "Split Data")
   D. Create a new model: Linear Regression (add the default Linear regression, i.e., do not change any values here)
   E. Finally, perform cross-validation on the dataset. (**Hint**: use the price column here)
   F. Visualize/report the values.



Properties Project

Partition and Sample

Partition or sample mode: Assign to Folds

☐ Use replacement in the parti...

☑ Randomized split

Random seed: 0

Specify the partitioner method: Partition evenly

Specify number of folds to split ...: 5

Stratified split: False

Figure: Property Tab of Partition and Sample Module

Hint: For part 4, follow each of the outline steps carefully. This should result in 5 blocks in your final workflow (including the Automobile price data (Raw) block).