CSE 6242 / CX 4242: Data and Visual Analytics | Georgia Tech | Fall 2025
HW 3: Spark, Docker, DataBricks, AWS and GCP

Download the HW3 Skeleton, Q1 Data, Q2 Data, and Q4 Data before you begin. Also, create an AWS Academy account as outlined in Step 1 of the AWS Setup Guide

Homework Overview

Modern-day datasets are large. For example, the NASA Terra and Aqua satellites each produce over 300GB of satellite imagery daily. These datasets are too large for typical computer hard drives and require advanced technologies for processing. In this assignment, you will work with a dataset of over 1 billion taxi trips from the New York City Taxi & Limousine Commission (TLC). Further details on this dataset are available here.

This assignment aims to familiarize you with various tools that will be valuable for future projects, research, or career opportunities. By including AWS and GCP, we want to provide the opportunity to explore and compare these rapidly evolving platforms. This experience will help you make informed decisions when selecting a cloud platform in the future, allowing you to get started quickly and confidently. Many of the computational tasks in this assignment are straightforward, though quite a bit of "setup" will be needed before reaching the actual "programming" stage. Setting up work environments, launching clusters, monitoring computation usage, and running large-scale experiments on cloud platforms are important skills. This assignment familiarizes you with using machine clusters and understanding the pay-per-use model of most cloud services, offering a valuable first experience with cloud computing for many students.

The maximum possible score for this homework is **100 points**

Important Notes

A. Submit your work by the due date on the course schedule.
   a. Every assignment has a generous 48-hour grace period, allowing students to address unexpected minor issues without facing penalties. You may use it without asking.
   b. Before the grace period expires, you may resubmit as many times as you need.
   c. TA assistance is not guaranteed during the grace period.
   d. Submissions during the grace period will display as "late" but **will not** incur a penalty.
   e. **We will not accept any submissions executed after the grace period ends**.
B. Always use the **most up-to-date assignment** (version number at the bottom right of this document). The latest version will be listed in Ed Discussion.
C. You may discuss ideas with other students at the "whiteboard" level (e.g., how cross-validation works, use HashMap instead of an array) and review any relevant materials online. However, **each student must write up and submit the student's own answers**.
D. All incidents of suspected dishonesty, plagiarism, or violations of the Georgia Tech Honor Code will be subject to the institute's Academic Integrity procedures, directly handled by the Office of Student Integrity (OSI). Consequences can be severe, e.g., academic probation or dismissal, a 0 grade for assignments concerned, and prohibition from withdrawing from the class.

Submission Notes

A. All questions are graded on the Gradescope platform, accessible through Canvas.
B. We will not accept submissions anywhere else outside of Gradescope.
C. Submit all required files as specified in each question. Make sure they are named correctly.
D. You may upload your code periodically to Gradescope to obtain feedback on your code. **There are no hidden test cases.** The score you see on Gradescope is what you will receive.
E. You must **not** use Gradescope as the primary way to test your code. It provides only a few test cases and error messages may not be as informative as local debuggers. Iteratively develop and test your code locally, write more test cases, and follow good coding practices. Use Gradescope mainly as a "final" check.
F. **Gradescope cannot run code that contains syntax errors**. If you get the "The autograder failed to execute correctly" error, verify:
   a. The code is free of syntax errors (by running locally)
   b. All methods have been implemented
   c. The correct file was submitted with the correct name
   d. No extra packages or files were imported
G. When many students use Gradescope simultaneously, it may slow down or fail. It can become even slower as the deadline approaches. You are responsible for submitting your work on time.
H. Each submission and its score will be recorded and saved by Gradescope. **By default, your last submission is used for grading.** To use a different submission, **you MUST "activate" it** (click the "Submission History" button at the bottom toolbar, then "Activate").

Do I need to use the specific version of the software listed?

Under each question, you will see a set of technologies with specific versions - this is what is installed on the autograder and what it will run your code with. Thus, installing those specific versions on your computer to complete the question is highly recommended. You may be able to complete the question with different versions installed locally, but you are responsible for determining the compatibility of your code. **We will not award points for code that works locally but not on the autograder.**

Q1 [25 points] Analyzing trips data with PySpark

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.

You will use the yellow_tripdata_2019-01_short.csv dataset, a modified record of the NYC Green Taxi trips that 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, unless specifically instructed to.**

Technology	PySpark, Docker
Deliverables	[Gradescope] **q1.ipynb:** your solution as a Jupyter Notebook file

IMPORTANT NOTES:
- Only regular PySpark Dataframe Operations can be used.
- While you are allowed to use pyspark.sql.functions like `sum()` and `avg()`, do NOT reference `sqlContext` within the functions you are defining for the assignment.
- Do NOT use the function `sqlContext.sql('select * ... ')`. Students frequently run into difficult-to-resolve issues with this approach. Additionally, since you already practiced SQL extensively in HW1, solving this task in SQL would offer limited educational value.
- If you re-run cells, remember to **restart the kernel** to clear the Spark context, otherwise an existing Spark context may cause errors.
- 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!
- Do not add any cells or additional library imports to the notebook.
- Remove all your additional debugging code that renders output, as it will crash Gradescope. For instance, any additional print, display and show statements used for debugging must be removed.
- If you encounter a "connection refused" error on Gradescope, it likely means that the system timed out while running your code. Review your code to ensure you aren't performing unnecessary processing, such as using overly complex nested loops.

Tasks and point breakdown

1. **[4 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

2. **[6 pts]** You will be modifying the function `common_pair`. Return the top 10 pickup-dropoff location pairs that have the highest sum of `passenger_count` who have traveled between them. Sort the location pairs by total passengers between pairs. For each location pair, also compute the average amount per passenger over all trips (name this `per_person_rate`), utilizing `total_amount`.

   For pairs with the same total passengers, sort them in descending order of `per_person_rate`. Filter out any trips that have the same pick-up and drop-off location. Rename the column for total passengers to `total_passenger_count`.

   Sample Output Format — The values below are for demonstration purposes:

PULocationID	DOLocationID	total_passenger_count	per_person_rate
1	2	23	5.242345
3	4	5	6.61345634

3. **[6 pts]** You will be modifying the function `distance_with_most_tip` . Filter the data for trips having fares (`fare_amount`) greater than $2.00 and a trip distance (`trip_distance`) greater than 0. Calculate the tip percent (`tip_amount * 100 / fare_amount`) for each trip. Round all trip distances **up** to the closest mile and find the average `tip_percent` for each `trip_distance`. Sort the result in descending order of `tip_percent` to obtain the top 15 trip distances which tip the most generously. Rename the column for rounded trip distances to `trip_distance`, and the column for average tip percents `tip_percent`.

   Sample Output Format — The values below are for demonstration purposes:

trip_distance	tip_percent
2	6.2632344561
1	4.42342882

4. **[9 pts]** You will be modifying the function `time_with_most_traffic` to determine which hour of the day has the most traffic. Calculate the traffic for a particular hour using the average speed of all taxi trips which began during that hour. Calculate the average speed as the average `trip_distance` divided by the average trip duration, as distance per hour. Make sure to determine the average durations and average trip distances before calculating the speed. It will likely be helpful to cast the dates to the long data type when determining the interval. A day with low average speed indicates high levels of traffic. The average speed may be 0, indicating very high levels of traffic.

Additionally, you must separate the hours into AM and PM, with hours 0:00-11:59 being AM, and hours 12:00-23:59 being PM. Convert these times to the 12-hour time, so you can match the output below. For example, the row with 1 as time of day, should show the average speed between 1 am and 2 am in the `am_avg_speed` column, and between 1 pm and 2pm in the `pm_avg_speed` column.

Use `date_format` along with the appropriate pattern letters to format the time of day so that it matches the example output below. Your final table should contain values sorted from 0-11 for `time_of_day`. There may be data missing for a time of day, and it may be null for `am_avg_speed` or `pm_avg_speed`. If an hour has no data for am or pm, there may be missing rows. You will not have rows for all possible times of day, and do not need to add them to the data if they are missing.

Sample Output Format — The values below are for demonstration purposes:

time_of_day	am_avg_speed	pm_avg_speed
1	0.953452345	9.23345272
2	5.2424622	null
4	null	2.55421905

Q2 [30 pts] Analyzing dataset with Spark on Databricks

For this question, we will be analyzing a dataset in Databricks using Spark DataFrames in Python. 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 on the Databricks platform. You will 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-tripdata 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.

Technology	Spark Databricks
Deliverables	[Gradescope] • **q2.dbc:** Your solution as dbc archive file (.dbc) exported from Databricks (see Databricks Setup Guide below) • **q2.ipynb:** Your solution as a Jupyter Notebook file (.ipynb) exported from Databricks • **q2_results.csv:** The output results from your code in the Databricks q2 notebook. You must carefully copy the outputs of the *display()* or *show()* function into a file titled *q2_results.csv* under the relevant sections. Double-check and compare your actual output with the results you copied.

IMPORTANT NOTES:
- Use only **Firefox, Safari or Chrome** when configuring anything related to Databricks. The setup process has been verified to work on these browsers.
- **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*.)
  - Note that you do not need to install Spark on your local machine because it is provided in the Databricks environment.

- **You must use only** Spark DataFrame operations for this question. 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.
  - Do not use the language magic %<language> with other languages like %r, %sql etc. Language magics are used to override the default language, which you **must not** do for this assignment.
  - 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.
- The template 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 (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).

---

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

- o All tasks listed below **must** be performed on this filtered DataFrame, or you will end up with wrong answers.
  - o Carefully read the instructions in the notebook, which provide hints for solving the problems.

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

Tasks and point breakdown

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 the 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 Databricks, refer to `show()` and `display()` functions in Spark.
- Note that you can edit this csv file using a text editor, but ensure that results are placed under the designated columns. If you encounter a "Can only compare identically-labeled Series objects" error, check that your results meet the column and row requirements for the task.

If you encounter a "UnicodeDecodeError", save the file as ".csv UTF-8" to resolve.

Note: Do **NOT** modify anything other than filling in those required output values in this csv file. **We grade by reviewing the Spark code you write and by looking at your results listed in this file**. So, make sure your output is obtained from the Spark code you write. **Failure to include the dbc and ipynb files will result in a deduction from your overall score.**

Q3 [35 points] Analyzing Large Amount of Data with PySpark on 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 AWS, and to gain experience that will help you tackle more complex problems.

The services you will primarily be using are Amazon S3 storage, Amazon Athena. You will be creating an S3 bucket, running code using Athena and its serverless PySpark engine, and then storing the output into that S3 bucket. Amazon Athena is serverless, meaning that you pay only for what you use. There are no servers to maintain that will accrue costs whether it's being used or not.

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 post in the dedicated AWS Setup Ed Discussion thread.

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. Optionally, if you would like to learn more about the dataset, check out here and here; also optionally, you may explore the structure of the data by referring to [1] [2].

You are provided with a python notebook (`q3.ipynb`) file which you will complete and load into Athena. You may need to allow third party cookies in your browser if the notebook doesn't load in Athena. You are provided with the load_data() function, which loads two PySpark DataFrames. The first DataFrame, **trips,** contains trip data where each record refers to one (1) trip. The second DataFrame, **zones,** maps a LocationID to its trip information. It can be linked to either the PULocationID or DOLocationID fields in the trips DataFrame.

Technology	PySpark, AWS
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.

IMPORTANT NOTES
- Use Firefox or Chrome when configuring anything related to AWS.
- **EXTREMELY IMPORTANT**: Both 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.**
- **Strictly follow the guidelines below, or your answer may not be graded.**

   a. 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.
   b. Do not import any functions which were not already imported within the skeleton.
   c. You will not have access to the Spark object directly in the autograder. If you use it in your functions, the autograder will fail!

d. Double check that you are submitting the correct files, and the filenames follow the correct naming standard — 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.

e. You are welcome to store your script's output in any bucket you choose if you can download and submit the correct files.

f. Do not make any manual changes to the output files.

g. Do not remove #export from the HW skeleton.

h. Do not import any additional packages, as this may cause the autograder to malfunction. Everything you need has already been imported for you.

i. Using .rdd() can cause issues in the GradeScope environment. You can accomplish this assignment without it. In general, since the RDD API is outdated (though not deprecated), you should be wary of using this API.

j. Remove all your additional debugging code that renders output, as it will crash Gradescope. For instance, any additional print, display and show statements used for debugging must be removed.

k. Refer to DataFrame commands such as filter, join, groupBy, agg, limit, sort, withColumnRenamed and withColumn. Documentation for the DataFrame APIs is located [here](#).

l. Testing on a single, small dataset (i.e. a "test case") is helpful, but is not sufficient for discovering all potential issues, especially if such issues only become apparent when the code is run on larger datasets. It is important for you to develop more ways to review and verify your code logic.

m. Overwriting the DataFrames from the function parameters can cause unintended side effects when it comes to rounding. Be sure to preserve the DataFrames in each function.

n. Make sure you return a DataFrame. If you get NoneType errors, you are most likely not returning what you think you are.

o. Some columns may need to be cast to the right data type. Keep that in mind!

Tasks and point breakdown

0. **[0 pts]** Setting up the AWS environment.
   a. 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., creating S3 storage bucket, and uploading skeleton file.

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

2. **[3 pts]** `trip_statistics(trips)`
   a. Compute basic statistics (count, mean, stddev, min, max) for trip_distance.
   b. Returns PySpark DataFrame with the schema (summary, trip_distance)

3. **[5 pts]** `busiest_hour(trips)`
   a. Determine the hour (0-23) of the day with the highest number of trips.
   b. Returns a PySpark DataFrame with a single row showing the hour with the highest trip count and the corresponding number of trips. Schema (hour, trip_count)

4. **[5 pts]** `most_freq_pickup_locations(trips)`
   a. Identify the top 10 pickup locations (by PULocationID) with the highest number of trips.
   b. Returns a PySpark DataFrame showing the top 10 PULocationIDs with the highest trip counts. Schema (PULocationID, trip_count)

5. **[6 pts]** `avg_trip_distance_and_duration(trips)`
   a. Calculate the average trip distance and average trip duration in minutes (i.e., divided by 60)

for each hour of the day (0-23). **A valid trip must have a non-null timestamp and a trip distance greater than zero**.

    b. Returns a PySpark DataFrame with 24 rows showing each hour (0-23) along with the average trip distance and average trip duration for that hour. Schema (hour, avg_trip_distance, avg_trip_duration)

Note: You can use unix_timestamp to help with calculating the duration. If there are null or invalid timestamps, you will want to handle those accordingly.

6. **[10 pts]** `most_freq_peak_hour_fares(trips, zones)`

    a. Identify the top 10 most frequent routes (combinations of PULocationID and DOLocationID) with pickups during peak hours (e.g., 7 AM – 8:59 AM and 4 PM – 6:59 PM). Peak hours can be defined as `7 <= hour < 9 and 16 <= hour < 19`. Display the pickup and drop-off location ID pairs along with their trip counts and average total fare rounded to two decimal places. Use total_amount for calculating average total fare. **To be a valid trip, a route must have a drop-off location different from the pickup location.**

    b. Returns a PySpark DataFrame showing the top 10 routes during peak hours with their trip counts, following the schema (PULocationID, PUZone, DOLocationID, DOZone, trip_count, avg_total_fare)

7. **[5 pts]** `q3_output_large.csv`

    a. The csv output of most_freq_peak_hour_fares using the large dataset, submitted to Gradescope as q3_output_large.csv

Note: There are expected DataFrame output examples for each question in q3.ipynb.

Once you have implemented all these functions, run the main() function with the large dataset, which has already been implemented, and update the line of code to include the name of your output S3 bucket and a location. **This function will overwrite your output** if the output directory already exists, so be aware of that when you do multiple runs. It will execute all your functions at once against the large dataset.

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.

Once you have finished coding, you can export the notebook from "Notebook Explorer" by selecting your notebook and clicking "Export File" from the Actions dropdown.

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

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

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

IMPORTANT NOTES:
- Use Firefox, Safari or Chrome when configuring anything related to GCP.
- **Strictly follow the guidelines below, or your answer may not be graded.**
  - Regular PySpark Dataframe Operations can be used.
  - Do **NOT** use any functions from the RDD API or your code will break the autograder. In general, the RDD API is considered outdated, so you should use the DataFrame API for better performance and compatibility.
  - Make sure to download the notebook from your GCP cluster **before** deleting the GCP cluster (otherwise, you will lose your work).
  - Do not add new cells to the notebook, as this may break the auto-grader.

- Remove all your additional debugging code that renders output, as it will crash Gradescope. For instance, any additional print, display and show statements used for debugging must be removed.
- Do not use any .rdd function in your code. <u>Not only will this break the autograder, but you should be wary of using this function in general</u>.
- Ensure that you are only submitting a COMPLETE solution to Gradescope. Anything less will break the autograder. Write local unit tests to help test your code.

Tasks and point breakdown

1. **[0 pts]** Set up your GCP environment
   a. Instructions to set up GCP Credits, GCP Storage and Dataproc Cluster are provided here: <u>written instructions</u>.
   b. Helpful tips/FAQs for special scenarios:
      i. If GCP service is disabled for your google account, try the steps in this <u>google support link</u>
      ii. If you have any issues with the GCP free credits, please post in the dedicated GCP Setup Ed Discussion thread.
2. **[0 pts — required]** Function `load_data()` to load data from a Google Storage Bucket into a Spark DataFrame
   a. You must first perform this task (part <u>2</u>) BEFORE performing parts <u>*3, 4, 5, 6 and 7*</u>. No points are allocated to task 2, 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. Upload code to Gradescope **ONLY** after completing all tasks and removing/commenting all the testing code. Anything else will break the autograder.
3. **[2 pts]** Function `exclude_no_pickup_locations()` to exclude trips with no pick-up locations (pick-up location id column is null or is zero) in the original data from a.
4. **[2 pts]** Function `exclude_no_trip_distance()` to exclude trips with no distance (i.e., trip distance column is null or zero) in the dataframe output by `exclude_no_pickup_locations()`.
5. **[2 pts]** Function `include_fare_range()` to include trips with fare from $20 (inclusively) to $60 (inclusively) in the dataframe output by `exclude_no_trip_distance()`.
6. **[2 pts]** Function `get_highest_tip()` to identify the highest tip (rounded to 2 decimal places) in the dataframe output by `include_fare_range()`.
7. **[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()`.