Big Data Coursework – 2023

I read the data from the given .csv files into the corresponding Spark DataFrames using the spark.read.csv() method and inputting the path from the shared ECS765 data repository for taxi nyc analysis and the taxi datasets. After cleaning the yellow_tripdata_df and the green_tripdata_df, I dropped the unnecessary (_c0) column from both.

For every task, where I needed to save the output in order to plot the data, I saved the final dataframe(s) to my s3 bucket. To do this, I used the coalesce() function and passed 1 as the parameter to put all the data into 1 csv file (to make reading the data easier for visualization), write function to write the data, option() function to keep the headers as the first row, mode() to overwrite any previous files with the same name (in order to save space on my bucket) and .csv() to save the file as a csv in my personal s3 bucket.

For all my tasks, I chose to use Spark structured API functions because it has better readability and is easier to understand.

For all plots, a black edge has been added to aid readability.

I wrote the output to my personal s3 bucket so that I could save the output locally and plot the data.

SparkID to run all 10 tasks: spark-49fe5fa6d3ac405d854c37d3aa2e3f5b

Task 1

Spark IDs (I needed to get the number of rows in the yellow dataframes separately):

spark-f46359969d5545958f4edc9201e68621

spark-333a5d370a70436fa174f6409d002be6

Methodology:

1) I used an inner join to join the taxi_zone_lookup dataframe to the yellow_tripdata_df dataframe on the condition that the yellow_tripdata_df's PULocationID equals the taxi_zone_lookup's LocationID. Then I renamed the 'Borough' column to 'Pickup_Borough,' the 'Zone' column to 'Pickup_Zone' and the 'service_zone' column to 'Pickup_service_zone'. I did this using the withColumnRenamed() column Then I dropped the extra 'LocationID' using drop().
2) Then, I used an inner join to join the taxi_zone_lookup dataframe to the yellow_tripdata_df dataframe on the condition that the yellow_tripdata_df's DOLocationID equals the taxi_zone_lookup's LocationID. Then I renamed the 'Borough' column to 'Dropoff_Borough,' the 'Zone' column to ' Dropoff_Zone' and the 'service_zone' column to ' Dropoff_service_zone'. I did this using the withColumnRenamed() column in order to match the schema given in the question. Then I dropped the extra 'LocationID' using drop().
3) I used the same steps for the green_tripdata_df.

Reasoning:

- I renamed the columns and dropped the LocationID in order to match the schema given in the question.
- For the joining tasks, I used an inner join as we had to join the entire taxi_zone_lookup dataframe to the taxi dataframes only when the condition is true.
- I cached the final yellow_tripdata_df and green_tripdata_df as I would use these dataframes later and it would improve the efficiency of the program if the dataframes were already pre-loaded instead of having to be remade every time the dataframe needed to be used.

Output:

The data for the third part of the task:

Yellow Taxi Rows	Yellow Taxi Columns	Green Taxi Rows	Green Taxi Columns
22197190	22	465295	23

The final schema for the yellow and green taxi DataFrames respectively:

```
root
 |-- tpep_pickup_datetime: string (nullable = true)
 |-- tpep_dropoff_datetime: string (nullable = true)
 |-- passenger_count: string (nullable = true)
 |-- trip_distance: string (nullable = true)
 |-- PULocationID: string (nullable = true)
 |-- DOLocationID: string (nullable = true)
 |-- payment_type: string (nullable = true)
 |-- fare_amount: string (nullable = true)
 |-- extra: string (nullable = true)
 |-- mta_tax: string (nullable = true)
 |-- tip_amount: string (nullable = true)
 |-- tolls_amount: string (nullable = true)
 |-- total_amount: string (nullable = true)
 |-- congestion_surcharge: string (nullable = true)
 |-- airport_fee: string (nullable = true)
 |-- taxi_type: string (nullable = true)
 |-- Pickup_Borough: string (nullable = true)
 |-- Pickup_Zone: string (nullable = true)
 |-- Pickup_service_zone: string (nullable = true)
 |-- Dropoff_Borough: string (nullable = true)
 |-- Dropoff_Zone: string (nullable = true)
 |-- Dropoff_service_zone: string (nullable = true)

root
 |-- lpep_pickup_datetime: string (nullable = true)
 |-- lpep_dropoff_datetime: string (nullable = true)
 |-- PULocationID: string (nullable = true)
 |-- DOLocationID: string (nullable = true)
 |-- passenger_count: string (nullable = true)
 |-- trip_distance: string (nullable = true)
 |-- fare_amount: string (nullable = true)
 |-- extra: string (nullable = true)
 |-- mta_tax: string (nullable = true)
 |-- tip_amount: string (nullable = true)
 |-- tolls_amount: string (nullable = true)
 |-- ehail_fee: string (nullable = true)
 |-- total_amount: string (nullable = true)
 |-- payment_type: string (nullable = true)
 |-- trip_type: string (nullable = true)
 |-- congestion_surcharge: string (nullable = true)
 |-- taxi_type: string (nullable = true)
 |-- Pickup_Borough: string (nullable = true)
 |-- Pickup_Zone: string (nullable = true)
 |-- Pickup_service_zone: string (nullable = true)
 |-- Dropoff_Borough: string (nullable = true)
 |-- Dropoff_Zone: string (nullable = true)
 |-- Dropoff_service_zone: string (nullable = true)
```

Task 2

Spark ID: spark-512f1bbd78b14e0f9bd820cdb83df157

Methodology:

1) I grouped the data in the 'Pickup_Borough' column of the yellow_tripdata_df and counted the number of rows in each group using the groupBy() and count() methods respectively.
2) I grouped the data in the 'Pickup_Borough' column of the green_tripdata_df and counted the number of rows in each group using the groupBy() and count() methods respectively.
3) I grouped the data in the 'Dropoff_Borough' column of the yellow_tripdata_df and counted the number of rows in each group using the groupBy() and count() methods respectively.
4) I grouped the data in the 'Dropoff_Borough' column of the green_tripdata_df and counted the number of rows in each group using the groupBy() and count() methods respectively.

Reasoning:

- I cached the results of the first 2 dataframes as I would be reusing the results for task 4 and it would be more efficient to have it in memory rather than remake the dataframe.

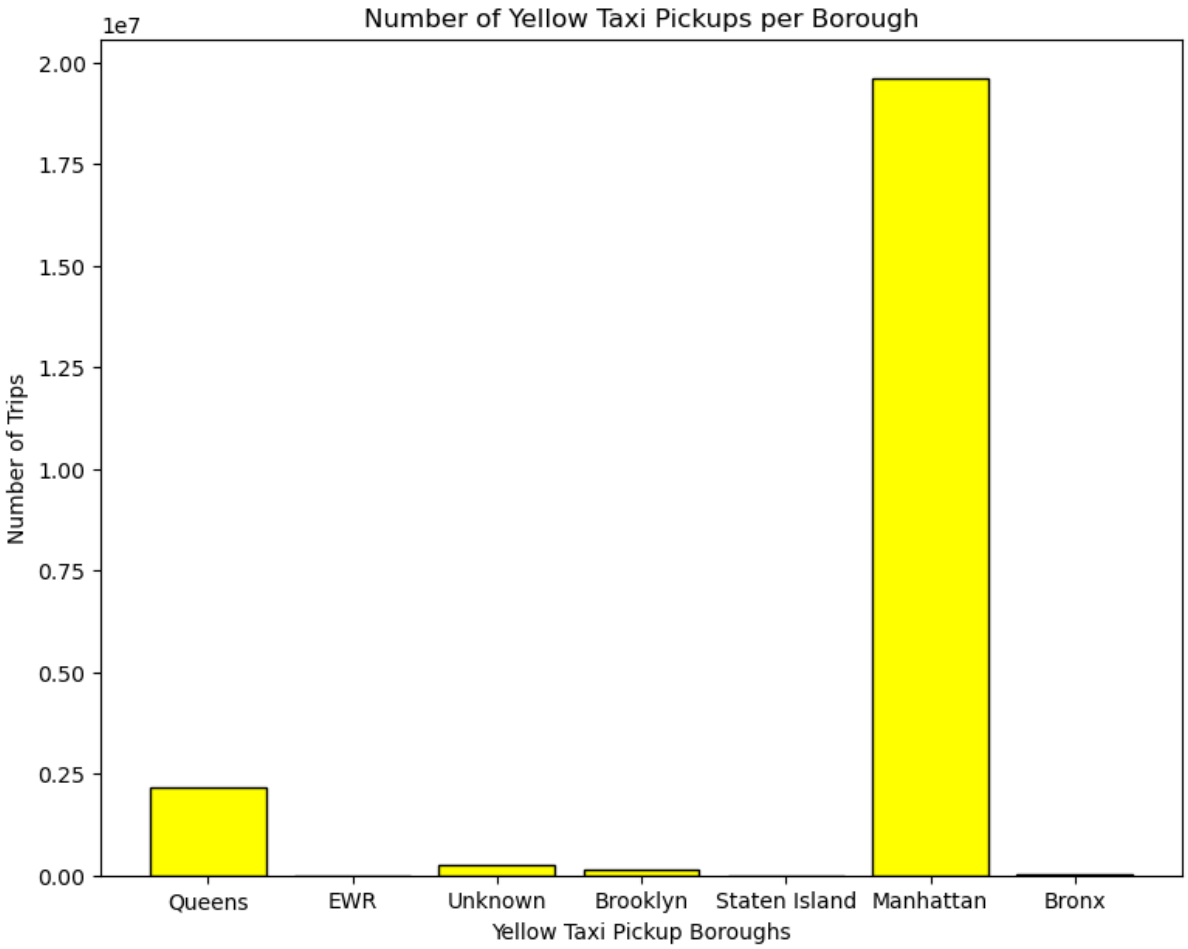
Plotting:

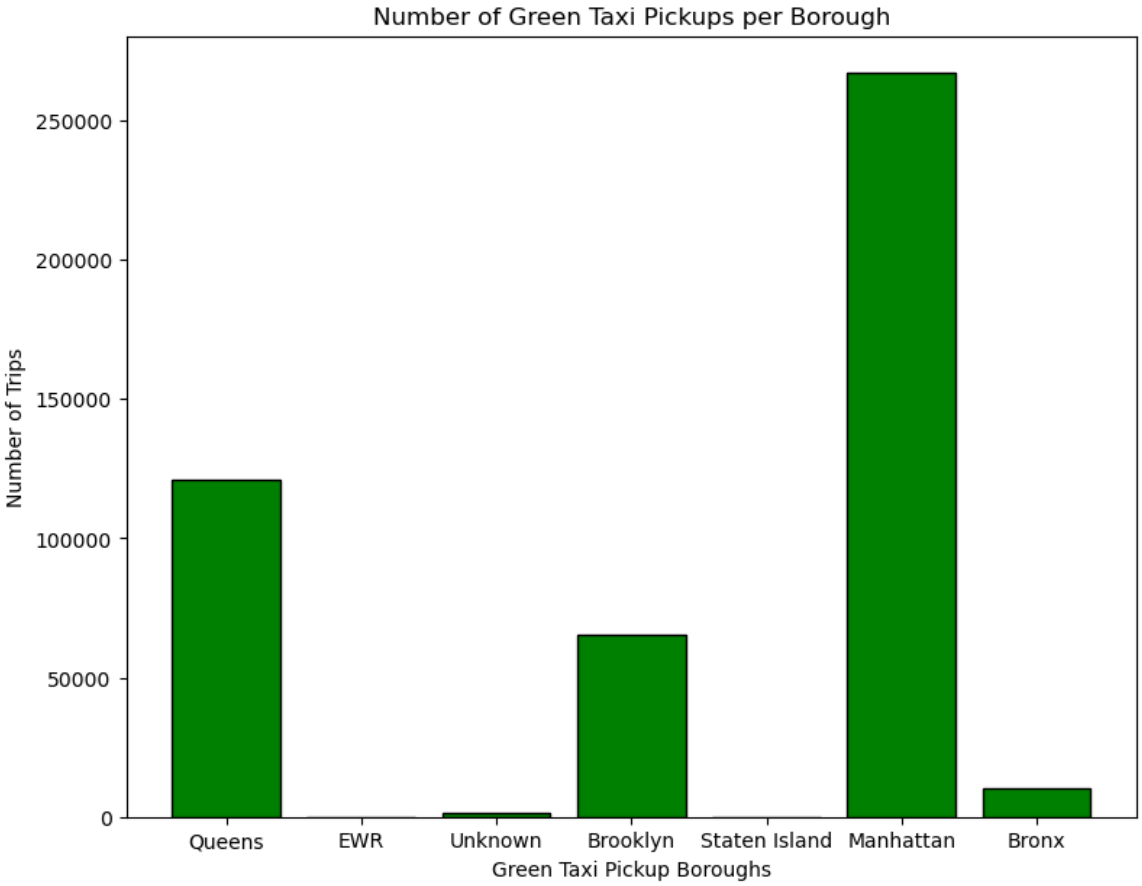
For all 4 plots, I read the csv file using pandas.read_csv() into a pandas dataframe. I then used the borough columns (as a list) for the x-axis and the count column for the y-axis. Then I used matplotlib.pyplot library to plot it as a bar graph using the plt.bar() method. I then added labels for the x-axis, y-axis and title respectively using:

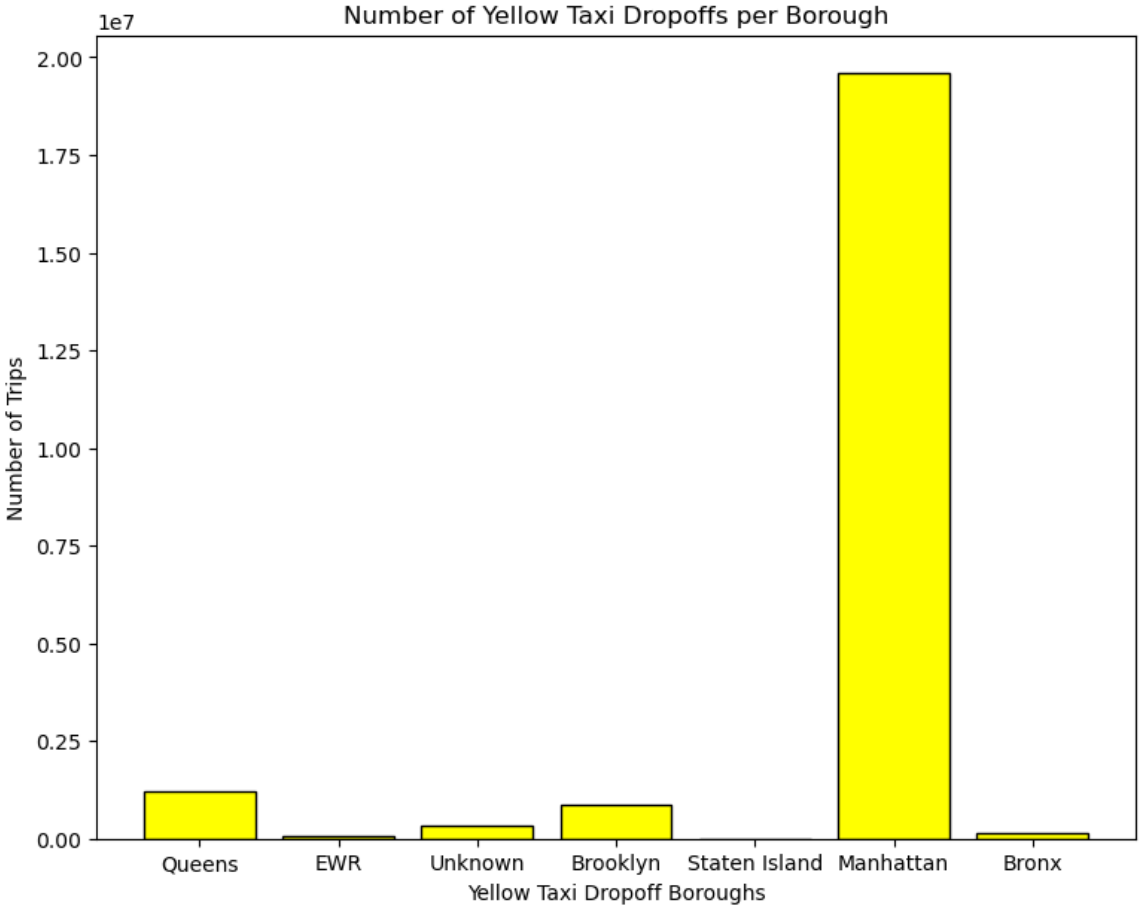
1) plt.xlabel('Yellow Taxi Pickup Boroughs'), plt.ylabel('Number of Trips') and plt.title('Number of yellow taxi pickups per borough')
2) plt.xlabel 'Green Taxi Pickup Boroughs'), plt.ylabel('Number of Trips') and plt.title('Number of Green taxi pickups per borough')
3) plt.xlabel('Yellow Taxi Dropoff Boroughs'), plt.ylabel('Number of Trips') and plt.title('Number of yellow taxi dropoffs per borough')
4) plt.xlabel('Number of Green Taxi Dropoffs per Borough'), plt.ylabel('Number of Trips') and plt.title('Number of Green Taxi Dropoffs per Borough')

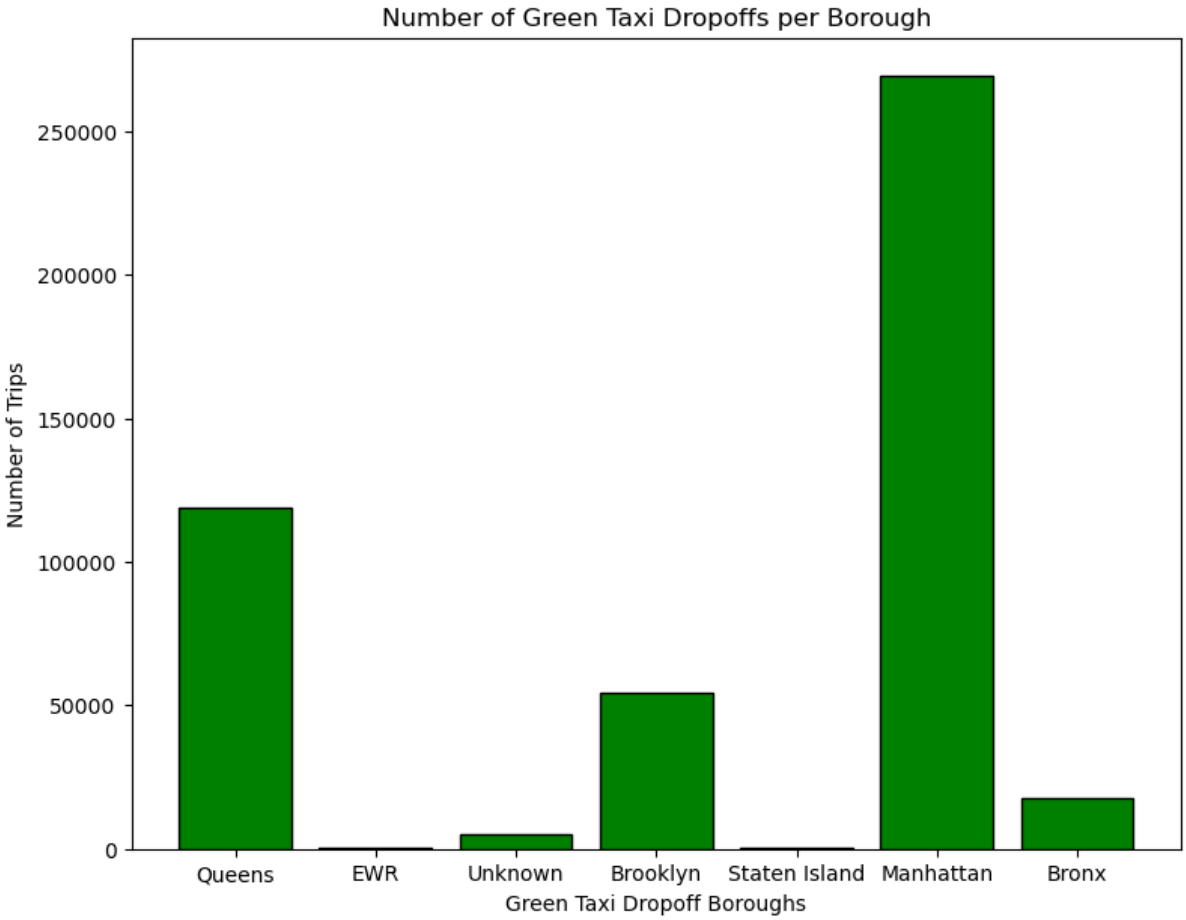
methods to improve the readability of the graph. I finally used plt.show() to show the plot.

Output:


Number of Yellow Taxi Pickups per Borough
1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Number of Trips
Queens
EWR
Unknown
Brooklyn
Staten Island
Manhattan
Bronx
Yellow Taxi Pickup Boroughs


Number of Green Taxi Pickups per Borough
Number of Trips
250000
200000
150000
100000
50000
0
Queens
EWR
Unknown
Brooklyn
Staten Island
Manhattan
Bronx
Green Taxi Pickup Boroughs


Number of Yellow Taxi Dropoffs per Borough
1e7
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Number of Trips
Queens
EWR
Unknown
Brooklyn
Staten Island
Manhattan
Bronx
Yellow Taxi Dropoff Boroughs


Number of Green Taxi Dropoffs per Borough
Number of Trips
250000
200000
150000
100000
50000
0
Queens
EWR
Unknown
Brooklyn
Staten Island
Manhattan
Bronx
Green Taxi Dropoff Boroughs

Task 3

Spark ID: spark-e1c9f02ef1a64bc2be886909454a468f

Methodology:

1) I created my own user defined function called filter which takes in a spark dataframe and returns a spark dataframe. In the function, I created a new dataframe consisting of the ‘tpep_pickup_datetime’, ‘trip_distance’ and ‘fare_amount’ columns using select() as those were the only columns I needed for this task and it would be more efficient with no unnecessary data. I then turned the string data in the ‘tpep_pickup_datetime’ column into date objects using the to_date() function within the withColumn() function. I then used the filter() function to keep dates between 2023-01-01 and 2023-01-07 (inclusive) and does not have trip_distance < 1 and does not have "fare_amount" > 50.
2) I passed the yellow_tripdata_df dataframe into the filter method and stored the filtered dataframe with the plot_filter_yellow_trip variable.
3) With the new dataframe, I grouped the data in the ‘tpep_pickup_datetime’ column of the plot_filter_yellow_trip dataframe and counted the number of rows in each group using the groupBy() and count() methods respectively.

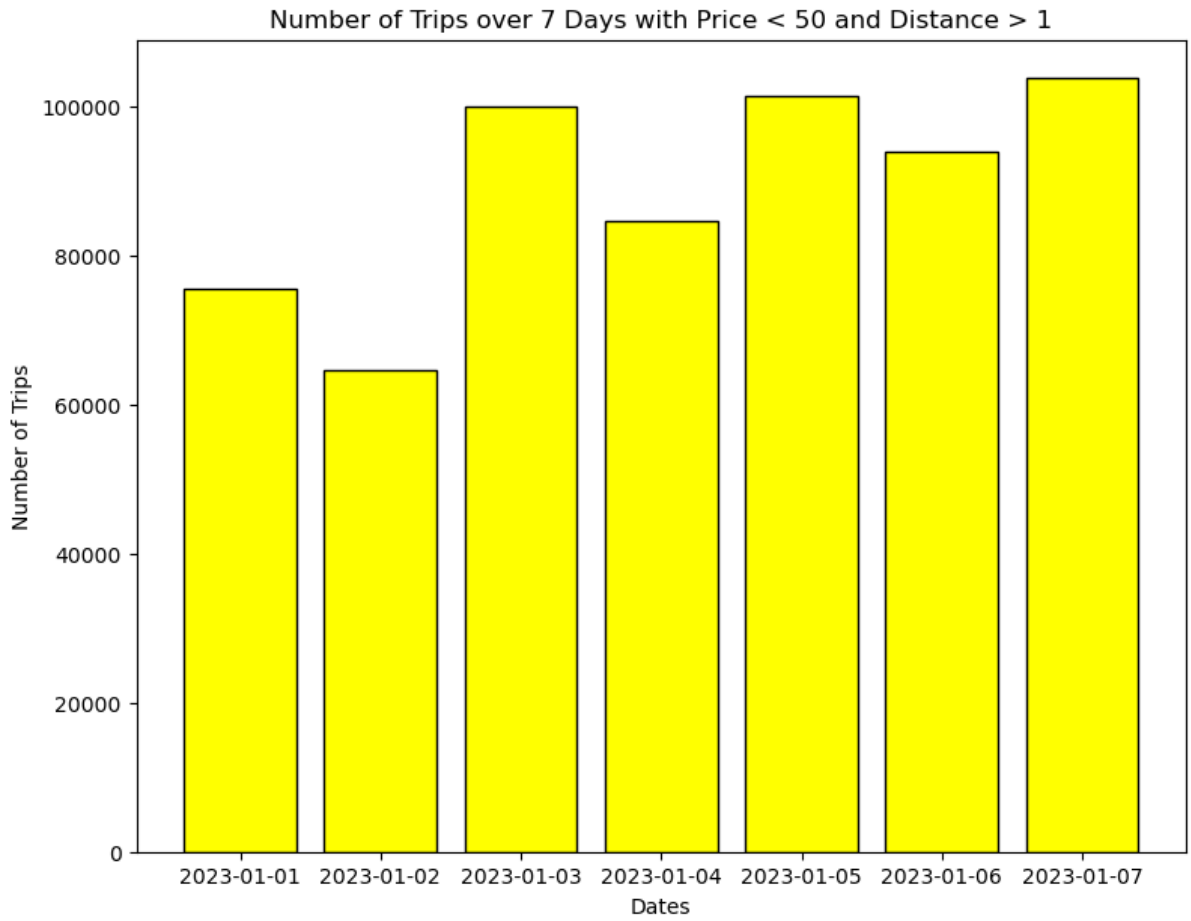
Reasoning:

- I turned the string data in the ‘tpep_pickup_datetime’ column into date objects because the question says that we need to plot dates along the x-axis and because we need to filter based on dates.
- I used the ‘tpep_pickup_datetime’ column due to personal preference as it is the first timestamp column in the schema, but ‘tpep_dropoff_datetime’ column should work as well.

Plotting:

I read the csv file using pandas.read_csv() into a pandas dataframe. I then used the tpep_pickup_datetime columns (as a list) for the x-axis and the count column for the y-axis. Then I used matplotlib.pyplot library to plot it as a bar graph using the plt.bar() method. I then added labels for the x-axis, y-axis and title respectively using plt.xlabel('Dates'), plt.ylabel('Number of Trips') and plt.title('Number of Trips over 7 Days with Price < 50 and Distance > 1') methods to improve the readability of the graph. I finally used plt.show() to show the plot.

Output:


Number of Trips over 7 Days with Price < 50 and Distance > 1
Number of Trips
0
20000
40000
60000
80000
100000
2023-01-01
2023-01-02
2023-01-03
2023-01-04
2023-01-05
2023-01-06
2023-01-07
Dates

Task 4

Spark ID: spark-ef1c932afdae46a7bc745cffbcd0f20b

Methodology:

1) I used the orderBy() method on the yellow_pickup_borough dataframe to order the dataframe by the number of trips. I used the desc() method to arrange the dataframe in descending order and the limit() method to get the top 5 results.
2) I reused the steps for the green_pickup_borough dataframe.

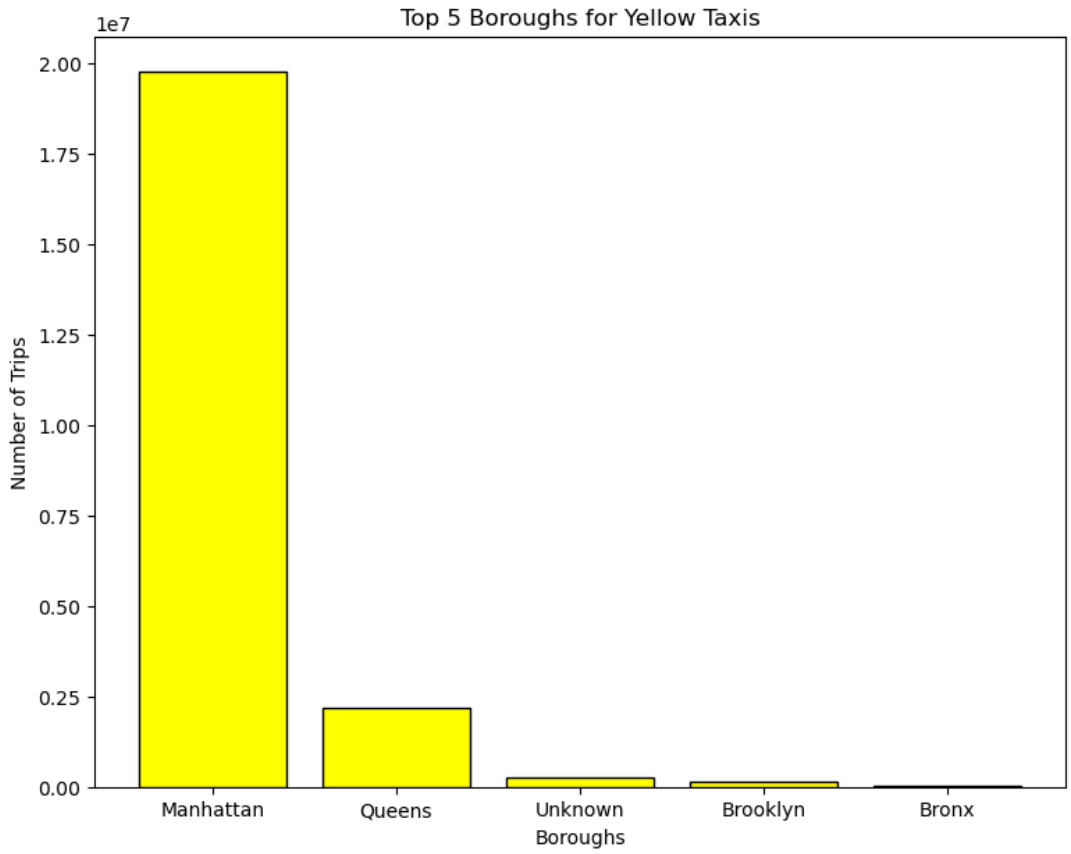
Reasoning:

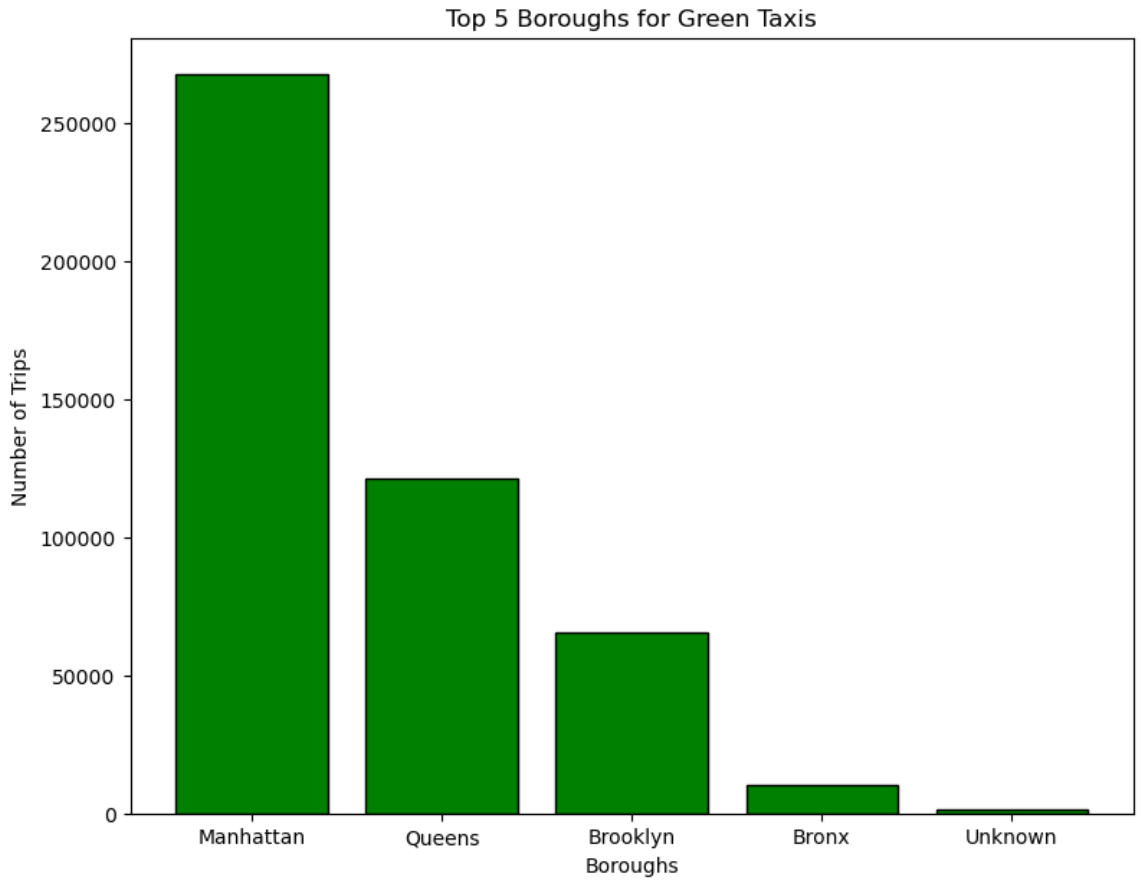
- I reused the yellow_pickup_borough and green_pickup_borough dataframes because the data needed for both the tasks are the same and the code will run faster if the data is already preloaded.

Plotting:

For both the graphs, I read the csv file using pandas.read_csv() into a pandas dataframe. I then used the Pickup_Borough columns (as a list) for the x-axis and the count column for the y-axis. Then I used matplotlib.pyplot library to plot it as a bar graph using the plt.bar() method. I then renamed the x and y-axis using the plt.xlabel('Days') and plt.ylabel('Number of Trips') methods. I finally used plt.show() to show the plot. The only difference is the colors of the graphs, the csv files and the title names.

Output:




Top 5 Boroughs for Green Taxis
Number of Trips
250000
200000
150000
100000
50000
0
Manhattan
Queens
Brooklyn
Bronx
Unknown
Boroughs

Task 5

SparkId: spark-40c5aeb51dfb41848d12dd9270ab5448

Methodology:

1) I created my own user defined function called findanomaly() which takes in a spark dataframe and an integer, and returns a spark dataframe. In the function, computed the lower bound as 50% less than the mean and the upper bound as 50% more than the mean. I then used the filter function to remove rows where the count was within 50% +- of the mean.
2) I read the data from the given .csv files into the corresponding Spark DataFrames using the spark.read.csv() method and inputting the path from the shared ECS765 data repository for taxi nyc analysis. After cleaning the yellow_June dataframe, I dropped the unnecessary (_c0) column.
3) I then turned the string data in the 'tpep_pickup_datetime' column into date objects using the to_date() function within the withColumn() function.
4) I grouped the data in the 'tpep_pickup_datetime' column of the plot_filter_yellow_trip dataframe and counted the number of rows in each group using the groupBy() and count() methods respectively.
5) To get the total number of trips, I created a dataframe after applying the sum() function to the count column and collected the values in the first row.
6) To get the mean_count I divided the total_trips by the number of entries in the yellow_June dataframe and printed it.
7) I then applied the find_anomaly function on the yellow_june dataset with the mean.

Reasoning:

- The reason I chose 50% more than the mean for the upper bound and 50% less than the mean for the lower bound was because the mean is the average value of all the data. That means that most values in the dataset should fall around this number. So I took 50% of the mean as the cut-off point and if a values is more than 50% above the mean and less than 50% less than the mean than it is an anomaly.
- I read a new .csv file as the question had specified that data from the month of June, 2023 had to be used.
- I only the took first row of the dataframe from applying sum() to the count column() as there is only 1 row and 1 column in the dataframe.
- I cached the results of the yellow_June dataframe as I would be using the value later on to send as a parameter and to write.
- I printed the mean for data analysis purposes.

Plotting:

I read the csv file using pandas.read_csv() into a pandas dataframe. I then used the ''tpep_pickup_datetime' column (as a list) for the x-axis and the 'count' column for the y-axis. Then I used matplotlib.pyplot library to plot it as a bar graph using the plt.scatter() method. I added labels for the x-axis, y-axis and title respectively using plt.xlabel('Dates'),

plt.ylabel('Number of Trips') and plt.title ('Anomalies from Spark Processing') methods to improve the readability of the graph. I finally used plt.show() to show the plot.

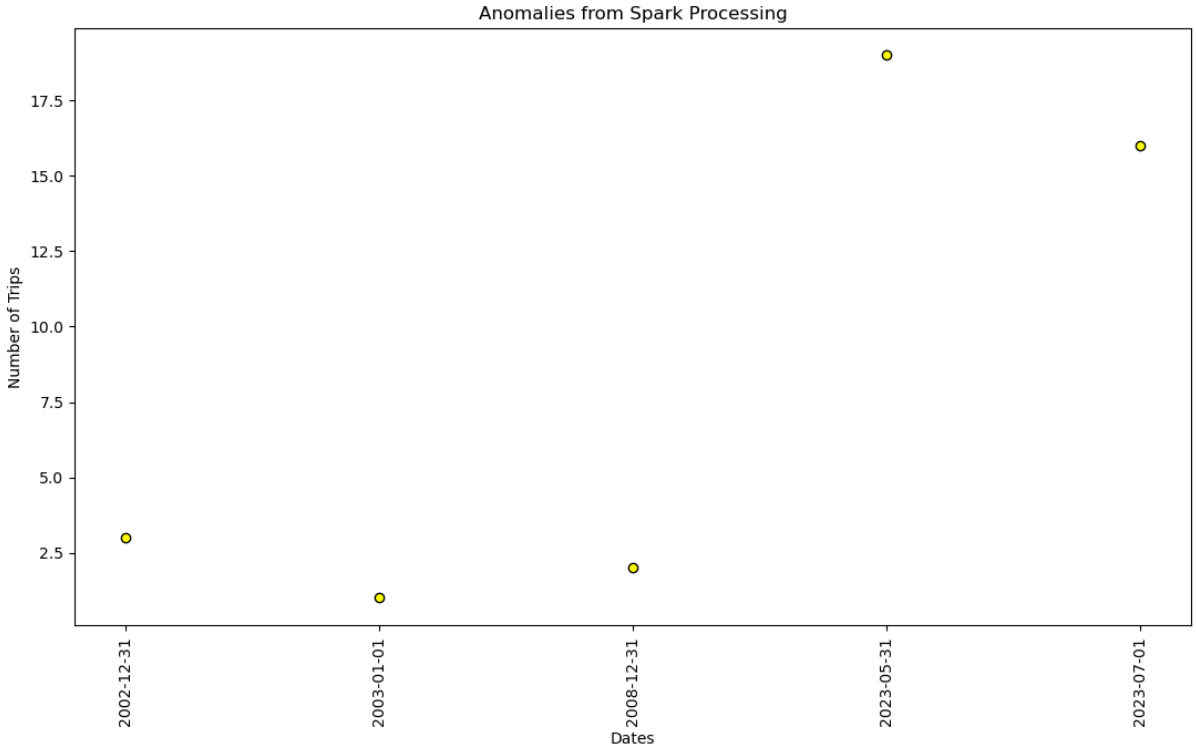
To show that the values gotten from the Spark Job were correct, I also plotted a graph with just the anomalies. I read the file with all of the data and plotted the mean, upper_bound and the lower_bound using the plt.axhline() method.

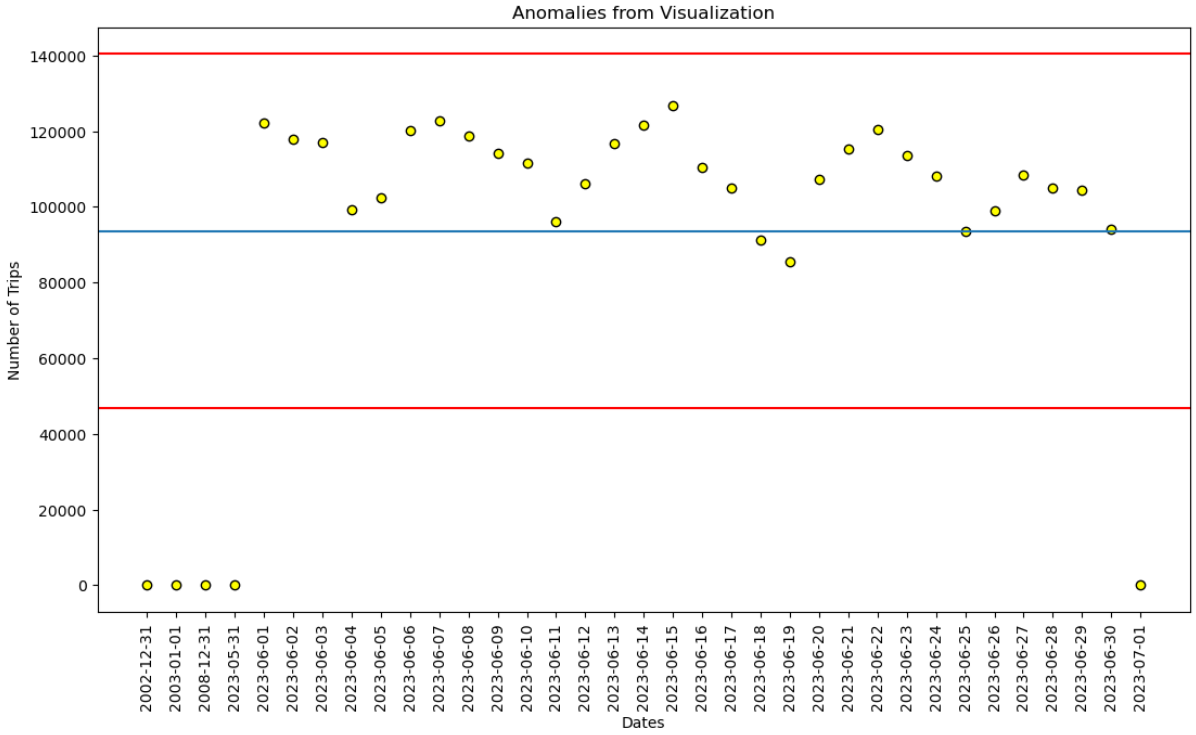
Analysis:

The values given in the instruction for the upper and lower bound returned the full dataset, so I chose my own upper and lower bound which I explained my reasoning for in the Reasoning section. From both graphs, you can clearly see that there are 5 distinct anomalies in the dataset. There are also 4 datapoints that shouldn't belong in the dataset as they do not belong to the month of June by these datapoints can chalked up to human error.

Output:

```
The mean is: 93599.0
```




Anomalies from Visualization
Number of Trips
Dates
140000
120000
100000
80000
60000
40000
20000
0
2002-12-31
2003-01-01
2008-12-31
2023-05-31
2023-06-01
2023-06-02
2023-06-03
2023-06-04
2023-06-05
2023-06-06
2023-06-07
2023-06-08
2023-06-09
2023-06-10
2023-06-11
2023-06-12
2023-06-13
2023-06-14
2023-06-15
2023-06-16
2023-06-17
2023-06-18
2023-06-19
2023-06-20
2023-06-21
2023-06-22
2023-06-23
2023-06-24
2023-06-25
2023-06-26
2023-06-27
2023-06-28
2023-06-29
2023-06-30
2023-07-01

Task 6

SparkID – spark-d8383bdd0f36452dae3b04a8d0e8c607

Methodology:

1) I read the data from the given .csv files into the corresponding Spark DataFrames using the spark.read.csv() method and inputting the path from the shared ECS765 data repository for taxi nyc analysis. After cleaning the yellow_March dataframe, I dropped the unnecessary (_c0) column.
2) I created a new dataframe using the select() method on the yellow_March dataframe. I chose the 'trip_distance' column and the 'fare_amount' column as those were the only columns I needed for this task and it would be more efficient with no unnecessary data.
3) I created a new column called 'fare_per_mile' which consisted of the data in the 'fare_amount' column (for that row) divided by the data in the 'trip_distance' column (for that row).
4) Finally, I dropped the 'trip_distance' column and the 'fare_amount' column as I only needed the 'fare_per_mile' column data to visualize the data.
5) I calculated the mean using the avg function and printed it.

Reasoning:

- I read a new .csv file as the question had specified that data from the month of March, 2023 had to be used.
- I printed the mean for data analysis purposes.

Plotting:

I read the csv file using pandas.read_csv() into a pandas dataframe. I then used the 'fare_per_mile' column (as a list) for the x-axis and as a list staring from 0 to the length of the 'fare_per_mile' column for the y-axis. Then I used matplotlib.pyplot library to plot it as a bar graph using the plt.scatter() method. I added labels for the y-axis and title respectively using plt.ylabel('Fare Per Mile') and plt.title('Statistical Processing') methods to improve the readability of the graph. Then I plotted the mean using the plt.axhline() method. I finally used plt.show() to show the plot.

However, I noticed that there were a lot of values that were concentrated around 0 so to get a better idea of the data, I created another graph with the log of the y-axis.

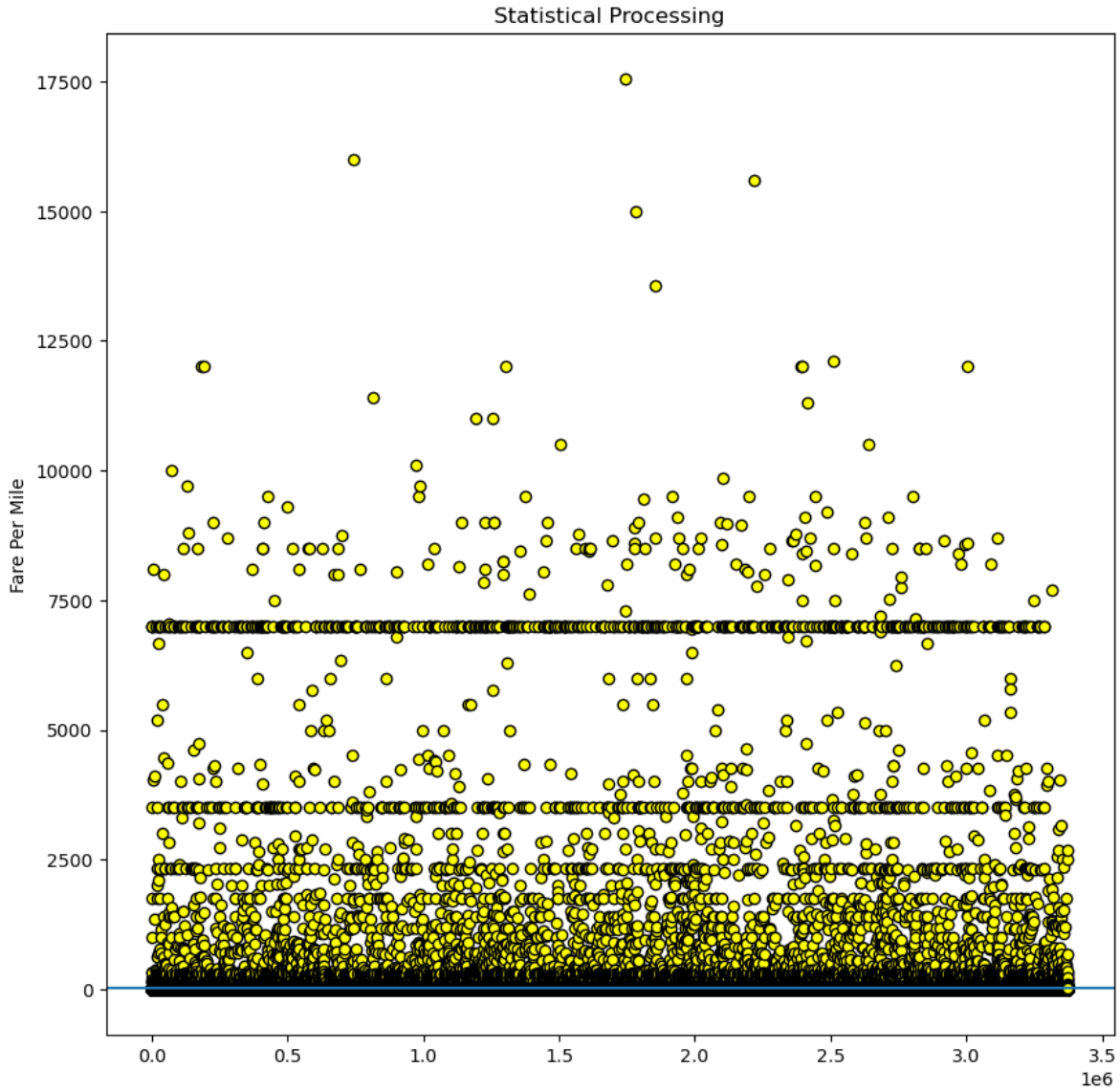
Analysis:

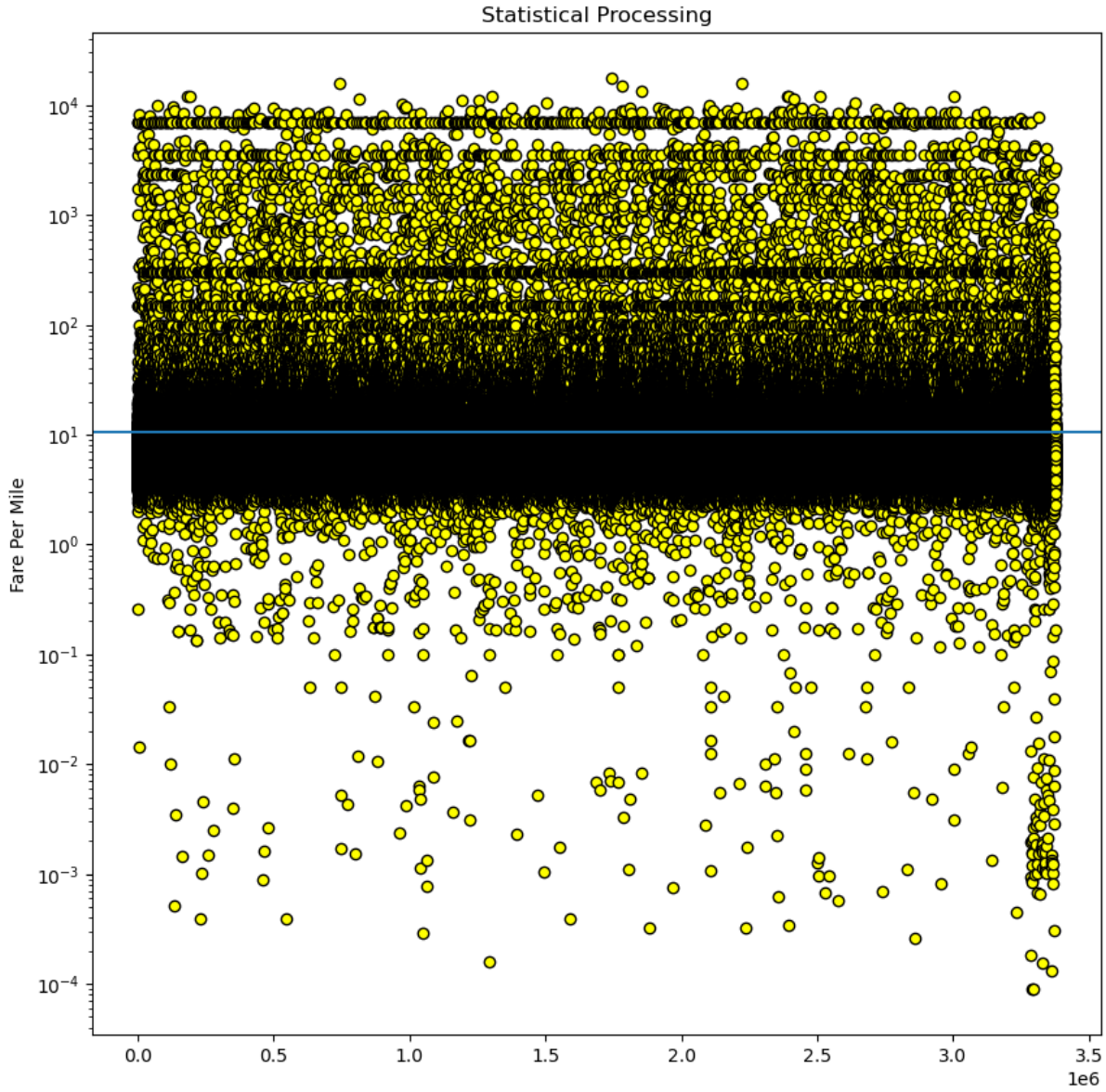
The y-axis is also stored in dollars.

The negative fare per miles could indicate the possibility of human errors present in that dataset. Since the mean is 12, that indicates a higher occurrence of data focusing on the lower values. According to the graph there are many few values that could be considered 'abnormally high,' the most prominent of which are the 5 largest values. These 5 plot points have a value between 12,500, 17700 inclusive.

Output:

```
The mean is: 10.478826871543824
```




Statistical Processing
Fare Per Mile
10^{4}
10^{3}
10^{2}
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e6

Task 7

Spark ID: spark-db73746a704b4a39b82dd36777570146

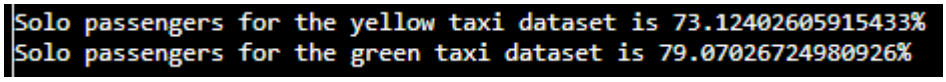
Methodology:

1) I created a new dataframe using the filter() method on the yellow_tripdata_df. I filtered out any rows where the data in the 'passenger_count' column is not equal to 1.0. I then counted the number of rows of the resulting dataframe using count(). I saved the result into the solo_yellow variable.
2) I repeated the same steps using the green_tripdata_df and stored the results into the solo_green variable.
3) I used the count() method on the yellow_tripdata_df and the green_tripdata_df dataframes to get the total number of trips for yellow taxis and green taxis and stored them into total_trip_count_yellow and total_trip_count_green variables respectively.
4) To get the output I printed the number of solo passengers/total trip count and multiplied the result by 100 to turn it into a percentage.

Reasoning:

- I filtered out rows where the number of passengers I more than 1 to get the dataframe consisting of just solo passengers.

Output:



Task 8

SparkID: spark-ce7b53eede3d4722bdd9f7b3510c9bde

Methodology:

1) I read the data from the given .csv files into the corresponding Spark DataFrames using the spark.read.csv() method and inputting the path from the shared ECS765 data repository for taxi nyc analysis. After cleaning the yellow_Jan dataframe, I dropped the unnecessary (_c0) column.
2) I changed the format of the data within the 'tpep_pickup_time' column from a string representing a timestamp to a string representing the Unix timestamp seconds. I used the unix_timestamp() method within the withColumn() method to achieve this.
3) I reused the step above for the data within the 'tpep_dropoff_time' column.
4) I then used the withColumn() method to created a new column 'trip_duration' which has the hours taken for the trip journey. This is calculated by subtracting the start time from the end time and then dividing the result by 360 and finally rounding the result.
5) I created a new dataframe using the select() method on the yellow_Jan dataframe's 'trip_duration' and ' trip_distance' columns since I only needed the those particular columns for my analysis.

Reasoning:

- I read a new .csv file as the question had specified that data from the month of January, 2023 had to be used.
- I turned the string timestamp into its unix timestamp consisting of seconds because that was the easiest method for me to turn it into an hour. It was also the most accurate way of getting the hour and just taking hour from the timestamp and subtracting the integers could have resulted in an inaccurate response.

Plotting:

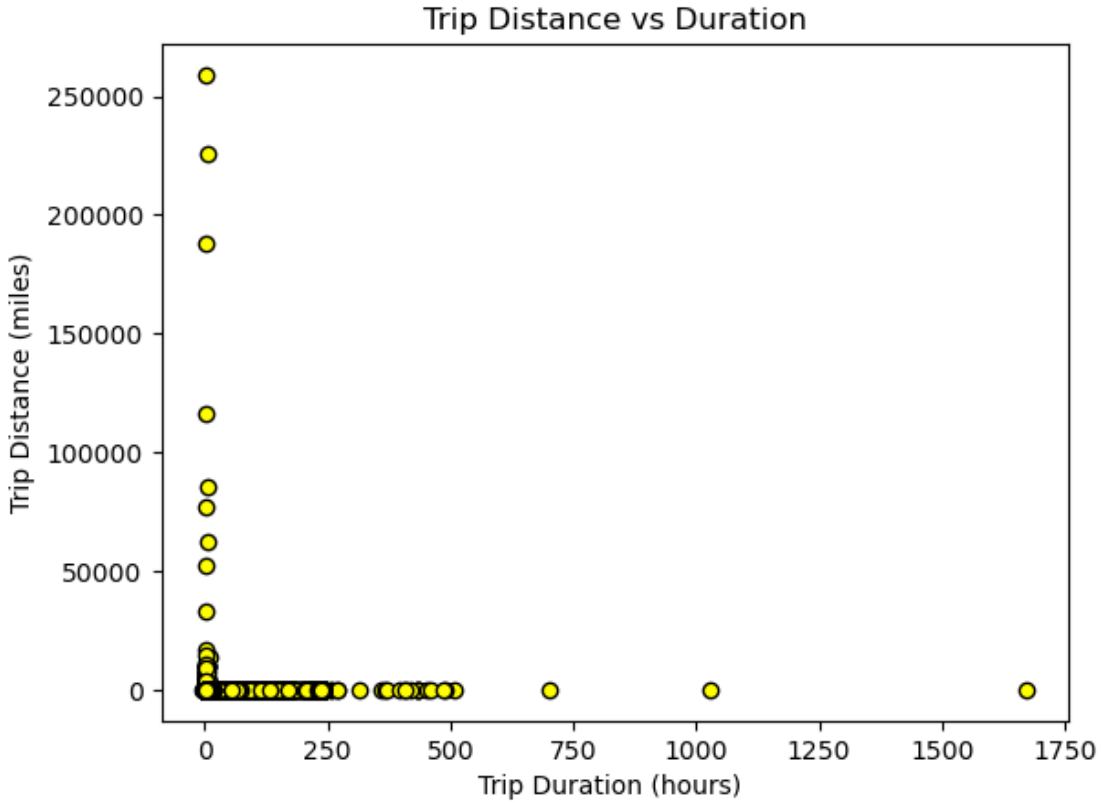
I read the csv file using pandas.read_csv() into a pandas dataframe. I then used the 'trip_duration' column (as a list) for the x-axis and the 'trip_distance' column for the y-axis. Then I used matplotlib.pyplot library to plot it as a bar graph using the plt.scatter() method. I added labels for the x-axis, y-axis and title respectively using plt.xlabel('Trip Duration (hours)'), plt.ylabel('Trip Distance (miles)') and plt.title('Trip Distance vs Duration') methods to improve the readability of the graph. I finally used plt.show() to show the plot.

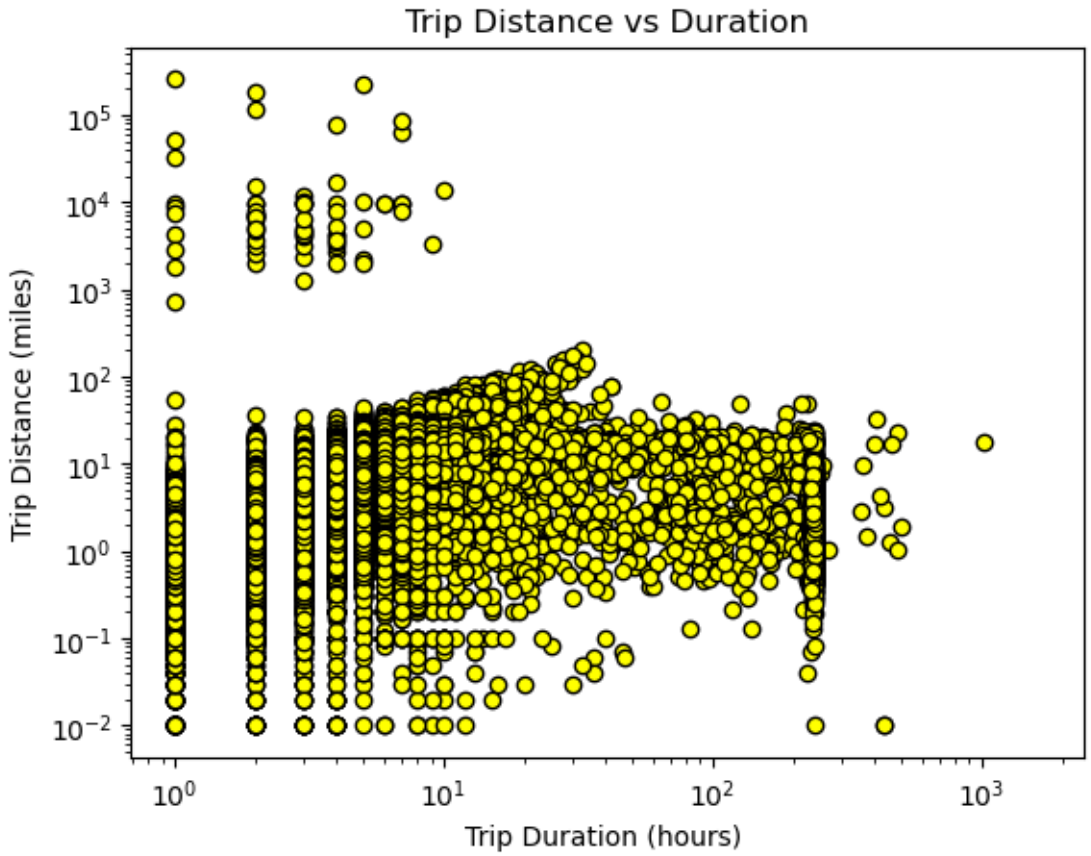
However, I noticed that there were a lot of values that were concentrated around 0 so to get a better idea of the data, I created another graph with the log of the y-axis.

Analysis:

According to the first graph, there seems to be an inverse relationship between trip distance and duration. This could be due to the addition of anomalies and human error as the log graph shows a linear relationship between trip distance and duration.

Output:


Trip Distance vs Duration
Trip Distance (miles)
250000
200000
150000
100000
50000
0
0
250
500
750
1000
1250
1500
1750
Trip Duration (hours)


Trip Distance vs Duration
Trip Distance (miles)
10^5
10^4
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^0
10^1
10^2
10^3
Trip Duration (hours)

Task 9

SparkID: spark-26ca6444833542d2ac3751c6d793fc00

Methodology:

1) I created a new dataframe using the select() method on the yellow_tripdata_df dataframe's 'Pickup_Borough' and 'taxi_type' columns since I only needed the those particular dataframes for my analysis.
2) I then created a new column 'Borough' using the withColumn() method which concatenated the data in the 'Pickup_Borough' column, the literal '_' and the data in the 'taxi_type' column. I did this order to differentiate between boroughs from the green taxi dataset and the yellow taxi dataset according to the requirements.
3) I then used the count method on the 'Borough' column to get a new dataframe with only the 'Borough' column and the count of occurrences of each borough. I then ordered the dataframe by the 'count' column using the orderBy() method. I used the desc() method to arrange the dataframe in descending order and the limit() method to get the top 5 results.
4) Finally I created a new column 'colour' using the withColumn() method which only has the literal 'yellow'.
5) I did the same steps for the green_tripdata_df dataframe and the final 'colour' column had the literal 'green.'

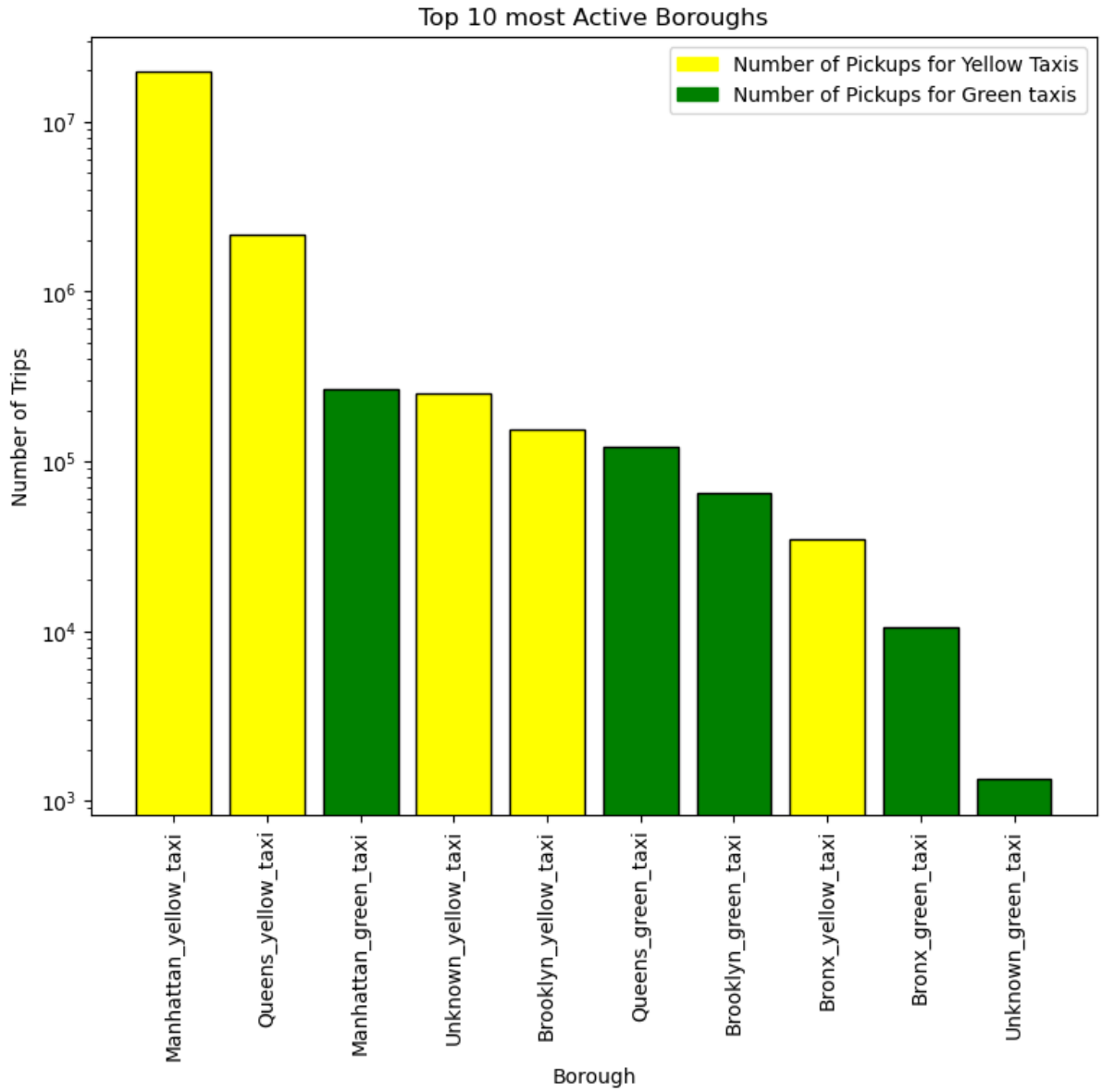
Reasoning:

- I added the 'colour' columns so that I could colour the bars separately when I had to visualise the data.

Plotting:

I read the csv file using pandas.read_csv() into a pandas dataframe. I then used the 'Borough' column (as a list) for the x-axis and the 'count' column (as a list) for the y-axis. Then I used matplotlib.pyplot library to plot it as a bar graph using the plt.bar() method. In order to avoid the overlapping of the labels on the y-axis, I set the rotation of the labels to 90 degrees. I created the legend by creating patches for the green and yellow trips and passed them to plt.legend() as parameters. I then added labels for the x-axis, y-axis and title respectively using plt.xlabel('Borough'), plt.ylabel('Number of Trips') and plt.title('Top 10 most Active Boroughs') methods to improve the readability of the graph. I finally used plt.show() to show the plot.

Output:


Top 10 most Active Boroughs
Number of Pickups for Yellow Taxis
Number of Pickups for Green taxis
Number of Trips
10^7
10^6
10^5
10^4
10^3
Manhattan_yellow_taxi
Queens_yellow_taxi
Manhattan_green_taxi
Unknown_yellow_taxi
Brooklyn_yellow_taxi
Queens_green_taxi
Brooklyn_green_taxi
Bronx_yellow_taxi
Bronx_green_taxi
Unknown_green_taxi
Borough

Task 10

Spark ID: spark-1946eb982b864f869296c214587647ed

Methodology:

8) I created a new dataframe using the select() method on the yellow_tripdata_df dataframe's 'tpep_pickup_datetime' and on the green_tripdata_df dataframe's 'lpep_pickup_datetime' (stored in task10_yellow and task10_green respectively).
9) I then extracted the months from the timestamp strings using the pyspark Date function month() and renamed the column to months using the withColumnRenamed() function for better readability.
10) I grouped the data of the 'months, column of the task10_yellow and task10_green dataframes functions and counted the number of rows in each group using the groupBy() and count() methods respectively. To get the month with the highest trip count for both the dataframes, I ordered the dataframes in descending order based on the 'count' column, and only saved the first row. With the dataframe with 1 row, I added a column colour with has the literal 'yellow' for the yellow taxi data and 'green' for the green taxi data.
11) Finally, I combined both the dataframes using the union() method.

Reasoning:

- I created new dataframes using select as I only needed those particular columns for my analysis, and it would be more efficient to only do analysis on the data I needed.
- I added the 'colour' columns so that I could colour the bars separately when I had to visualise the data.
- I only the took first row of each taxi's month dataframes.
- I used union() method since both dataframes had the same schema and I needed to plot both the data on the same graph.

Plotting:

I read the csv file using pandas.read_csv() into a pandas dataframe. For this task, I used the data_for_task10.plot() method which uses the pandas.dataframe.plot() method instead of plt.plot(), as I used previously, because I wanted only the months with the most number of trips on the x-axis instead of a range of the months. Therefore, it was easier to plot the graph I wanted using the former method rather than the latter (However, pandas.plot() uses the matplotlib library so there isn't much difference otherwise). Thus, I plotted a bar graph with 'Months' on the x-axis, 'count' on the y-axis. I set rot=0 because the label on the y-axis' default is to show it sideways. I set the color to the 'colour' column in the dataframe which allowed me to set different colours for the different rows on the same column. I created the legend by creating patches for the green and yellow trips and passed them to plt.legend() as parameters. I then added labels for the x-axis, y-axis and title respectively using plt.xlabel('Months'), plt.ylabel('Number of Trips') and plt.title('Months With the Most

Number of Trips') methods to improve the readability of the graph. I finally used plt.show() to show the plot.

Output: