Module: ECS765P - Big Data Processing 2023/24

Coursework: NYC Rideshare Analysis

Student name: Sahib Bhatti

Student ID: 190319889

Introduction:

This report outlines the steps taken to carry out analysis on NYC Rideshare data featuring how I tackled tasks, the results attained, showcasing visualisations, extracting meaningful insights and addressing challenges I faced and what I learnt.

Task 1:

Task explanation: This section describes the process of loading and manipulating rideshare data. The main steps included merging the rideshare data with taxi zone lookup data through pick up and drop off location fields. Following this by implementing borough, zone, and service zone fields for pick up and drop off. Then using the UNIX timestamp, and converting the date field to yyyy-MM-dd which is human understandable. Finally printing the schema and number of rows of the merged data frame.

APIs and functions used in code:

- **Loading Data:** using spark.read.option(“header”, “true”).csv, the files rideshare_data.csv and taxi_zone_lookup.csv were loaded into their separate dataframes. The command indicates that the first row of the CSV file contains the header, which specifies the column names and .csv method is used to read CSV files in Spark. Then the paths for both CSV files are constructed using the S3 bucket name.
- **Renaming Columns:** using the select function I selected the columns, and renamed them using the alias() function to differentiate between pickup and drop off columns e.g. taxi_zone_lookup_df.select(col("LocationID").alias("pickup_LocationID")…
- **Join Operations:** firstly, I joined the data through pickup_location with respect to LocationID. Secondly, I joined the dropoff_location with respect to LocationID. These were joined in accordance to the previous renamed columns.
- **UNIX timestamp data transformation:** after selecting the current date column to rename using withColumn(), I used the from_unixtime function specifying that I want the date column set to the specified format of "yyyy-MM-dd" for the timestamp.
- **Rows and Schema check:** using the count() function, I counted the number of rows of the dataframe and used the print function to verify them. I used the printSchema() function to display the schema and ensure that the schema and number of rows are correct and complete. I used the drop() function to remove an unnecessary fields that are not part of the schema.

Visualisation of results:

```
Number of rows: 69725864
root
 |-- business: string (nullable = true)
 |-- pickup_location: string (nullable = true)
 |-- dropoff_location: string (nullable = true)
 |-- trip_length: string (nullable = true)
 |-- request_to_pickup: string (nullable = true)
 |-- total_ride_time: string (nullable = true)
 |-- on_scene_to_pickup: string (nullable = true)
 |-- on_scene_to_dropoff: string (nullable = true)
 |-- time_of_day: string (nullable = true)
 |-- date: string (nullable = true)
 |-- passenger_fare: string (nullable = true)
 |-- driver_total_pay: string (nullable = true)
 |-- rideshare_profit: string (nullable = true)
 |-- hourly_rate: string (nullable = true)
 |-- dollars_per_mile: 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)
```

1.1 Schema & rows

```
+-----------+---------------+-----------+----------+
|trip_length|total_ride_time|time_of_day|      date|
+-----------+---------------+-----------+----------+
|       4.98|          761.0|    morning|2023-05-22|
|       4.35|         1423.0|    morning|2023-05-22|
|       8.82|         1527.0|    morning|2023-05-22|
|       8.72|         1761.0|    morning|2023-05-22|
|       5.05|         1762.0|    morning|2023-05-22|
|      12.64|         2504.0|    morning|2023-05-22|
|       14.3|         1871.0|    morning|2023-05-22|
|       1.05|          323.0|    morning|2023-05-22|
|       0.57|          169.0|    morning|2023-05-22|
|       2.08|          822.0|    morning|2023-05-22|
|       0.84|          615.0|    morning|2023-05-22|
|      11.79|         1779.0|    morning|2023-05-22|
|       0.91|          413.0|    morning|2023-05-22|
|       3.37|         1487.0|    morning|2023-05-22|
|       2.77|          801.0|    morning|2023-05-22|
|       1.15|          552.0|    morning|2023-05-22|
|      11.98|         2455.0|    morning|2023-05-22|
|       8.54|         2517.0|    morning|2023-05-22|
|       3.16|         1405.0|    morning|2023-05-22|
|       3.58|          924.0|    morning|2023-05-22|
+-----------+---------------+-----------+----------+
only showing top 20 rows
```

1.2 Converted data column

Challenges faced:

1) Complex join operations: Attempting the join in two steps and making them accurate was a difficult hurdle. Starting is always difficult and this join being one of the first main tasks in this coursework was a good milestone to conquer to get me comfortable with the following tasks to come. I overcame the challenge of the join through validating it, using independent research and then testing the output.

2) Unnecessary fields: Upon first printing the schema, I had pickup_LocationID and dropoff_LocationID which were not part of the schema required. I overcame this challenge by researching how to remove columns. I came across the drop() function which I had used to remove these unnecessary fields to ensure that my schema perfectly resembled the desired output.

Knowledge and insights attained:

This opening task had made me familiar with the dataset, it's respective fields and the general functions that may be required to manipulate and transform the data for the following tasks. I had learnt how to clean the data with appropriate column names, and how to transform the data in a format that is human understandable such as using UNIX time function to transform the date.

Task 2:

Task explanation: This section focuses on aggregating the data through calculating and visualising the number of trips, platform profits and driver earnings. These metrics were evaluated against the performance of rideshare businesses across many months. Histogram visualisations were plotted to extract insights for stakeholders to make decisions for strategies on growth and allocation of resources from assessing the market.

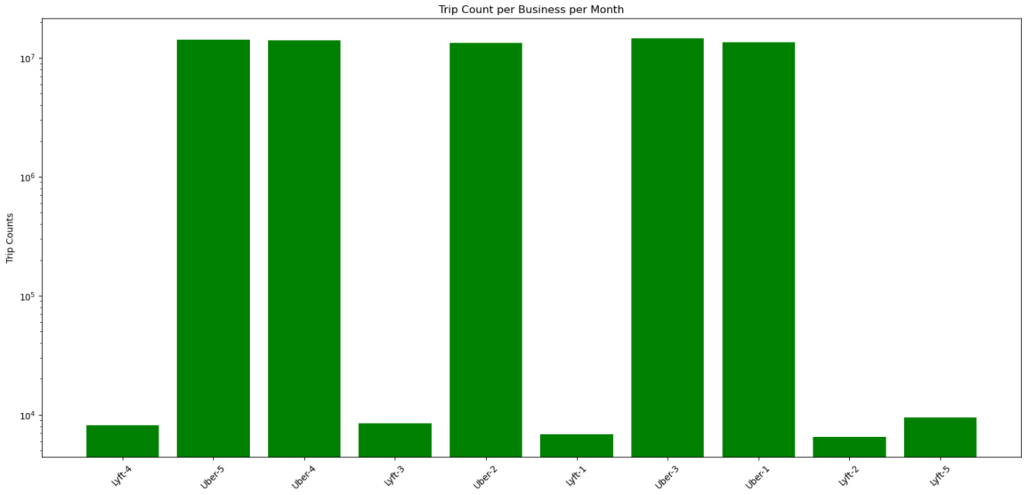
APIs and functions used in code:

- **Data aggregations (groupby, orderby count, sum) & data preparation functions:**
    - **[2.1]** For the **number of trips** for each business in each month **groupby()** was used to group the dataframe by the “business” column and month which was extracted from the “date” column using the **month()** function. Using **alias()** the date column was renamed to month. Then finally **count()** was used to count the number of occurrences for each combination of the grouped column where it computes the number of trips per business per month. Code: df.groupBy("business", month("date").alias("month")).count()
    - **[2.2]** For **calculating platform profit** for each business in each month the **groupby()** function was used again to combine the dataframe by the “business” and “month” fields again in the same way. Now the data has been grouped, the profit is aggregated through the **sum() function**; sum("rideshare_profit") calculates the sum of the "rideshare_profit" values for each group. Then using **alias()** the resulting column from the aggregating is named to total profit; alias("total_profit").
    - **[2.3]** For **calculating the driver earnings** for each business in each month, firstly the **col()** function was used to specify the “driver_total_pay” field where **cast(DoubleType())** was used to convert the column to DoubleType. Then using **withColumn()** and the **month()** function, the month was extracted into another column. So **groupby()** was used to group by the "business" and "month" columns, then **sum()** computes the sum of the "driver_total_pay" column within each group.

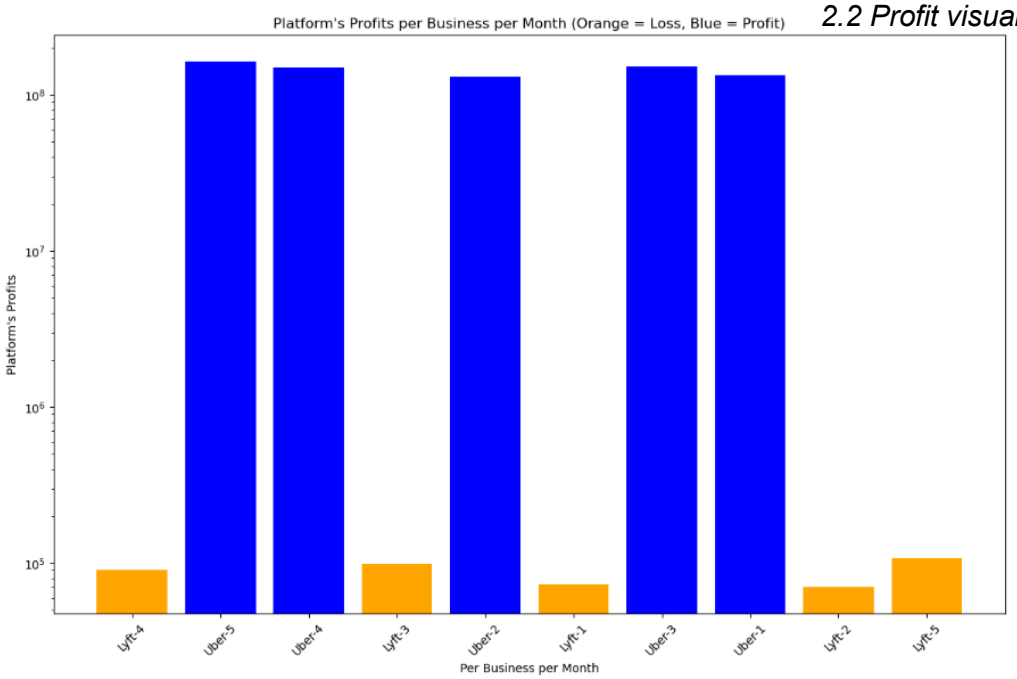
- **Histogram visualisation:** employing Python’s Matplotlib library to plot histograms. Each subtask had it’s own respective dataframe i.e. df_trip, df_profit and df_earnings. Utilizing the **coalesce()** function to “1” single partition was used in writing the DataFrames to their own individual single output csv files in the specified s3_bucket location. The write() function is set to overwrite to replace any existing data in the specified location, and “header” is set to true to include column headers in the csv file. Then utilising pandas, pd.read_csv() was used to read the csv files containing the dataframe for each subtask which was then plotted in histograms accordingly.

Visualisation of results:

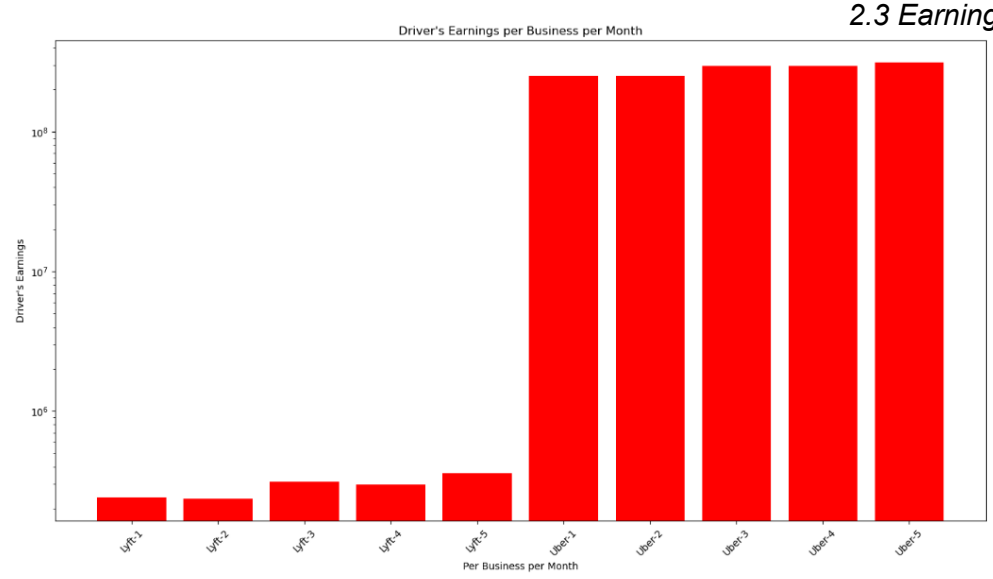
2.1 Trip visualisation



2.2 Profit visualisation



2.3 Earnings visualisation



Result insights:

- **Trip Counts Histogram:** This graph shows a consistent difference between the amount of trips between businesses across months where Uber showed higher trip counts than Lyft. Uber dominates the market in this instance and from analysing the graph and as a stakeholder, I'd advise them to continue their current strategies. Whereas if I was a stakeholder for Lyft, I'd advise them to reduce their price of trips so that riders are inclined to book more often with Lyft increasing their trip counts and competitiveness against Uber.
- **Platform Profits Histogram:** This graph captures Uber's success again over other businesses such as Lyft since Uber has leading platform profits per month. Lyft is making losses indicated by the orange bars. Therefore, if I was a stakeholder for Uber I'd advise them to continue their current strategies since they have been successful. Whereas for Lyft, the histogram helps me understand that they require profit-growth strategies so I'd advise them to cut down on costs or invoke new strategies to attract more customers to gain more trips and thus profit.
- **Driver Earnings Histogram:** This graph displays distinctive higher earnings across months for Uber drivers in comparison to Lyft drivers. These findings across all three graphs correlate with Uber having higher trip counts, thus higher profits and thus higher earnings for drivers whereas for Lyft it is almost the complete opposite. From viewing the graph and as a stakeholder, I'd recommend Uber to continue their current strategies, but I'd advise Lyft to focus on strategies to increase their turnover which will steadily increase profits and thus driver earnings.

Challenges faced:

In this task, I faced issues with visualising my results. More precisely I faced difficulties implementing the **coalesce()** function and writing the DataFrames into their individual csv files. At times the server would crash when executing this second task or I would face errors even though the code was correct. I overcame this challenge by creating new terminals and running the code where I finally managed to download the csv files from the folder locations I specified in the code. From there I plotted the histograms using Python's Matplotlib library.

Knowledge and Insights Attained

The histograms were useful tools to visually analyse the data and spot trends. The graphs indicated that Uber is a larger established company with greater market share, which translates to higher trip volume, platform profits and thus driver earnings. Whereas in comparison to Lyft, action must be drastically taken to increase turnover, trip counts, profit and driver earnings. In turn this would improve their market competitiveness and enhance market share and profit growth.

Task 3:

Task explanation: This task involves analysing the where the demand lies for the businesses using the data. The aim is to identify the top 5 popular pickup and drop off boroughs each month, followed by identifying the top 30 earnest routes where the sum of driver_total_pay is used as the profit. Since 3.1, 3.2 and 3.3 required similar API functions, I have summarised them together.

APIs and functions used in code:

- **Partitioning using Window functions and ordering:** using the function Window.partitionBy("month") the window is partitioned by the "month" column. The purpose of this is to rank pickup or drop off boroughs based on the month column thus we combine this with the orderby() function against the column trip_count in descending order since you want the top boroughs per month at the top.
- **Manipulating columns to rank:** using withColumn() this function is used create a new column named "rank" where the rank() function is applied to assign a rank to each row within its partition previously defined where the over() function is used to apply the window specification to the DataFrame, thus ranking the top boroughs for each month.
- **Filtering:** applying the filter() function against the rank column where the rank is less than or equal to 5 is used to select the top 5 boroughs for both pickup and drop off for each month
- **Selecting and ordering columns:** using the select() function, we specify which columns we want included in the dataframe, for the top 5 pickup boroughs each month these were Pickup_borough/Dropoff_borough, month, and trip_count. Using orderby() we order the month by ascending and the trip count by descending for each month.
- **Concatenating to create columns**: using the concat_ws() function, concatenates the values of the ‘Pickup_Borough’ and ‘Dropoff_Borough’ columns with the separator " to ", creating a new column named "Route". Each row in the ‘Route’ column represents the route from the pickup borough to the dropoff borough.
- **Grouping:** using the groupBy() function to group the route ‘Route’ column which creates groups of rows where each group represents a unique route.
- **Aggregation and ordering:** using the sum() function to compute the sum of the driver_total_pay column per each route to calculate the driver earnings per route. The aggregation of this is renamed to ‘total_profit’ using the withColumnRenamed() function. Using the orderby() function, this total_profit column is ordered by descending to the get the top profitable routes.
- **Displaying results:** Finally the show() function is used to display the dataframes with their requested amount of records.

Visualisation of results:

Pickup_Borough	month	trip_count
Manhattan	1	5854818
Brooklyn	1	3360373
Queens	1	2589034
Bronx	1	1607789
Staten Island	1	173354
Manhattan	2	5808244
Brooklyn	2	3283003
Queens	2	2447213
Bronx	2	1581889
Staten Island	2	166328
Manhattan	3	6194298
Brooklyn	3	3632776
Queens	3	2757895
Bronx	3	1785166
Staten Island	3	191935
Manhattan	4	6002714
Brooklyn	4	3481220
Queens	4	2666671
Bronx	4	1677435
Staten Island	4	175356
Manhattan	5	5965594
Brooklyn	5	3586009
Queens	5	2826599
Bronx	5	1717137
Staten Island	5	189924

3.1 Top 5 Pickups per month

Dropoff_Borough	month	trip_count
Manhattan	1	5444345
Brooklyn	1	3337415
Queens	1	2480080
Bronx	1	1525137
Unknown	1	535610
Manhattan	2	5381696
Brooklyn	2	3251795
Queens	2	2390783
Bronx	2	1511014
Unknown	2	497525
Manhattan	3	5671301
Brooklyn	3	3608960
Queens	3	2713748
Bronx	3	1706802
Unknown	3	566798
Manhattan	4	5530417
Brooklyn	4	3448225
Queens	4	2605086
Bronx	4	1596505
Unknown	4	551857
Manhattan	5	5428986
Brooklyn	5	3560322
Queens	5	2780011
Bronx	5	1639180
Unknown	5	578549

3.2 Top 5 Dropoffs per month

Route	total_profit
Manhattan to Manhattan	3.33857725550022 84E8
Brooklyn to Brooklyn	1.7394472147999206E8
Queens to Queens	1.1470684719998911E8
Manhattan to Queens	1.0173842820999996E8
Queens to Manhattan	8.603540026000004E7
Manhattan to Unknown	8.010710241999996E7
Bronx to Bronx	7.414622575999324E7
Manhattan to Brooklyn	6.799047558999999E7
Brooklyn to Manhattan	6.3176161049999975E7
Brooklyn to Queens	5.045416243000009E7
Queens to Brooklyn	4.729286536000016E7
Queens to Unknown	4.629299990000003E7
Bronx to Manhattan	3.24863251700001E7
Manhattan to Bronx	3.1978763450000063E7
Manhattan to EWR	2.3750888619999986E7
Brooklyn to Unknown	1.0848827569999995E7
Bronx to Unknown	1.0464800209999993E7
Bronx to Queens	1.0292266499999996E7
Queens to Bronx	1.0182898729999997E7
Staten Island to Staten Island	9686862.45000001
Brooklyn to Bronx	5848822.560000003
Bronx to Brooklyn	5629874.41
Brooklyn to EWR	3292761.710000002
Brooklyn to Staten Island	2417853.82
Staten Island to Brooklyn	2265856.4599999995
Manhattan to Staten Island	2223727.37
Staten Island to Manhattan	1612227.7199999997
Queens to EWR	1192758.6599999997
Staten Island to Unknown	891285.8100000002
Queens to Staten Island	865603.3799999999

only showing top 30 rows

3.3 Top 30 Earnest routes

Results insights:

It is clear that the most popular pickup and dropoff boroughs correlate with the most profitable routes such as Manhattan. It is also common for the most popular pickup boroughs to also be the most popular drop off boroughs. Thus, we can see which boroughs have the most rider/taxi activity. We can assume the reason of this being to due some areas having greater populations than others, or some areas being larger in general. We can also see that the boroughs with high trip counts clearly have the most total profit gained from those areas. Thus as a stakeholder, these results portray the where the demand lies which can be important for helping in making strategies such as allocating resources and for example deciding on the cost of a trip etc. Through knowing the demand, prices can be adjusted accordingly to best stimulate profits.

Challenges faced:

1) Window and rank: Prior to this task, I had not known how to partition the data by month and how to rank it. Through research, I came across the window and rank functions which enabled me to overcome this challenge as they helped me rank the top boroughs per month.

2) Creating the route column: At first glance, it was unknown to me on how to create the route column from pickup to dropoff borough fields. Through researching into the concat_ws() function I overcame this challenge.

Knowledge and Insights Attained:

In this task, I learnt how to partition and rank data for the purpose of generating insights. By ranking data, you can focus on where the demand in the data lies. This is useful, especially as a CEO as you want to focus resources in areas required, such as lower profitable routes require more attention. From this task I understood that there are still many API functions for me to discover.

Task 4:

Task explanation: Task 4 aims to discover which time of day maximises average driver earnings and average trip lengths. These metrics are then combined to calculate the average earnings per mile for each time of day. Attaining this knowledge can reveal insights for which time of day drivers can optimise their earnings and productivity.

APIs and functions used in code:

- **Data aggregations (groupBy, avg, sortBy) & data preparation functions:**
    - Using groupBy() to group the dataframe by ‘time_of_day’ column creating unique groups for each time of day i.e. morning, afternoon, evening, night.
    - Applying the aggregate avg() to the ‘driver_total_pay’ and to ‘trip_length’ columns and computing the average total pay and average trip length across each unique time of day and then applying the alias() function to rename the aggregated columns to ‘average_total_pay’ for task 4.1 and ‘average_trip_length’ for task 4.2.
    - Sorting the new ‘average_total_pay’ column through the orderBy() function in descending order which therefore sorts the time of day categories based on the average pay earned by drivers. For example, the highest average pay is sorted to the top in accordance with the time of day. Sorting the new ‘average_trip_length’ in the same descending order using the orderBy() function where the greatest average trip length is at the top with accordance to the time of day.
- **Join operation and substeps:** Joining the recent metrics from task 4.1 and 4.2 of average_total_pay to average_trip_length on the ‘time_of_day’ column creating a dataframe with these columns.
    - Using the withColumn() function to add a new column ‘average_earning_per_mile’ to the dataframe which is calculated through using the col() function to retrieve values from ‘average_total_pay’ and dividing by ‘average_trip_length’ using the ‘/’ operator performing record-wise division.
    - Using the select() function to relevantly select only the ‘time_of_day’ and ‘average_earning_per_mile’ columns from the DataFrame.
    - Sorting the ‘average_earning_per_mile' column through descending order where the greatest average earning per mile value is listed at the top against the time of day.
- **Displaying results:** Finally the show() function is used to display the dataframes with their requested amount of records.

Visualisation of results:

time_of_day	average_driver_total_pay
afternoon	21.212428756593535
night	20.08743800359271
evening	19.77742770239839
morning	19.633332793944835

4.1 Average driver total pay

time_of_day	average_trip_length
night	5.32398480196174
morning	4.927371866442785
afternoon	4.861410525661209
evening	4.484750367447519

4.2 Average trip length

time_of_day	average_earning_per_mile
evening	4.409928330894958
afternoon	4.363430869420024
morning	3.984544565766397
night	3.7730081416068377

4.3 Average earning per mile

Results insights:

The results underline how the time of the day impacts the average driver pay, average trip length and the average earning per mile which is both of these combined. While the afternoon has the highest average driver total pay, and the nigh has the highest average trip length, as a driver it would be most worthwhile to work during the evening since it has the highest average earning per mile. This being due to the evening have a moderate average driver total pay but having the lowest average trip length. The second safest option would be work during the afternoon since this has the highest average driver total pay and second lowest average trip length resulting in the second highest average earning per mile. Thus as a stakeholder, these insights are extremely useful in determining what time of day is most productive, efficient and profitable for drivers to work to optimise their earnings.

Challenges faced:

1) Join operation: Figuring out how to join two data frames and then combining them to calculate the average earning per mile was tricky. I overcame this by realising I have to use the shared ‘time_of_day’ column and using the col() function to extract each dataframes respective metric of ‘average_driver_total_pay’ and ‘average_trip_length’ to divide them to attain values of average_earning_per_mile.

2) Using a combination of functions: Utilising a combination of functions to achieve the correct result seem difficult at first glance. However I overcame this by realising I have to write the code logically step by step to retrieve the answer required.

Knowledge and Insights Attained:

The output analysis uncovered beneficial insights for how driver earnings and trip lengths are impacted by time of day. The average_earnings_per_mile column is especially useful as it can enable drivers to schedule their work for times where it is more profitable, to optimise their income even though there may be fewer or shorter trips. The CEO could use this data to apply incentives to less profitable times for drivers to work to balance out the average earnings per mile across each part of the day to enable constant earnings for drivers but also constant throughput for the CEO. What I had learn from this task is how to use the avg() function which I also have not done before in spark.

Task 5

Task explanation: This section focuses on examining the average waiting time for rideshare services in the month of January. It involves plotting a histogram visualisation ordered by the days in January which can be used to depict outliers of average waiting times.

APIs and functions used in code:

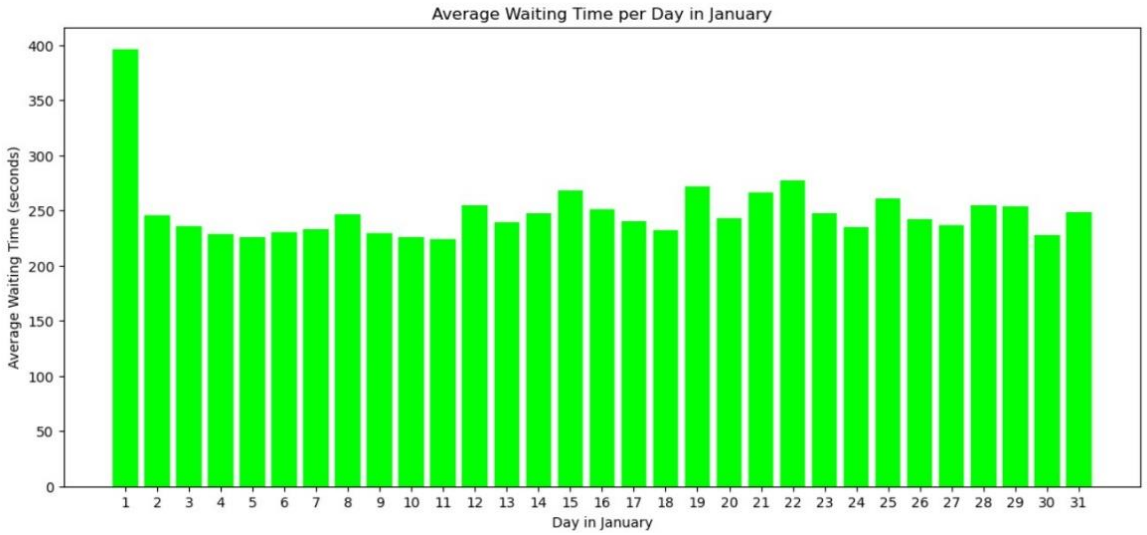
- **Data preparation functions:**
    - Using the filter() function in task 5.1 to filter in combination with the month() function to extract the month from the ‘date’ column and to filter it to the condition of equalling 1 which is corresponds to January i.e. df.filter(month("date") == 1).
    - Applying the withColumn() function to add a new column named "day" to the DataFrame. The second argument being the dayofmonth("date") function that calculates and extracts the day of the month for each date in the "date" column.
    - Then using the groupBy() function to group the January date by the ‘day’ column.
- **Aggregation function:** using the aggregate avg() function to compute the average values of ‘request_to_pickup’ column for each group of days and using the alias() function rename the resulting column as ‘average_waiting_time’.
- **Sorting function:** using orderby() function to order the ‘average_waiting_time’ by day e.g. 1st - 31st **.**
- **Histogram visualisation:** employing Python’s Matplotlib library to plot histogram for the average waiting time (y-axis) by day (x-axis). Utilizing the coalesce() function to “1” single partition was used in writing the DataFrames to it’s single output csv file in the specified s3_bucket location. The write() function is set to overwrite to replace any existing data in the specified location, and “header” is set to true to include column headers in the csv file. Then utilising pandas, pd.read_csv() was used to read the csv file containing the dataframe which was then plotted into the histogram accordingly.
- **Filtering:** In task 5.2, the filter() function is used to filters the recent DataFrame to retain only the rows where the "average_waiting_time" column value is greater than 300 seconds. Then, using the select() function it selects the "day" and "average_waiting_time" columns from the filtered DataFrame, which represent the days and their corresponding average waiting times where the threshold of 300 seconds is exceeded.
- **Displaying results:** Finally the show() function is used to display the dataframes with their requested amount of records.

Visualisation of results:

5.1: Average waiting time dataframe and histogram:

day	average_waiting_time
1	396.5318744409635
2	246.05148716456986
3	235.68026834234155
4	228.85434668408274
5	226.08877381422872
6	230.35306927438575
7	233.25699185710533
8	246.41358687741243
9	229.265944341545
10	225.65276195086662
11	224.40468798627612
12	255.17599322195403
13	239.22308233638282
14	247.49345781069232
15	268.5346481777792
16	251.55102299494047
17	240.5772885527869
18	231.90770494488552
19	272.02203820618143
20	243.43761253646377

only showing top 20 rows



5.2: Average waiting time > 300s:

day	average_waiting_time
1	396.5318744409635

5.3: Why was the average waiting time longer on these days compared to other days?

The average waiting time was longer on day 1. The reason can be due to there being increased demand on the first of January. This can be due to the day being New Years Day and many people going out and celebrating, thus rideshare companies and other companies in general may have experienced a large spike in demand for the day. Thus the demand for rideshare vehicles may have exceeded supply and resulted in greater waiting times for riders to find a driver. Another reason for increased waiting times could be due to rideshare companies facing operational issues that day, so were slower in responding, resulting in greater wait times. This analysis uncovers valuable insights as strategic decisions could be made for drivers to plan their schedules effectively to find ways to reduce the average wait time by picking up riders in a closer proximity only. Or rideshare companies can adjust allocation of drivers or prices on days with busy demand such as reducing prices on these days to maintain customer satisfaction.

Challenges faced:

1) Extracting the data: The main challenge I faced in this task was wondering how to extract the data for solely .After experimenting with different functions, I overcame this challenge by revisiting the filter() combined with month() function from task 2 where using the month() function I could simply filter the ‘date’ column to ‘1’.

2) Sorting the data: Another challenge was partitioning and ordering the data by ‘day’. I overcame this challenge by using withColumn()to add a new column named "day" to the DataFrame, and came across the dayofmonth() function applied to “date” which extracted the day of the month for each date in the "date" column.

Knowledge and Insights Attained:

From this task, I learnt the importance of filtering and ordering data to extract and preprocess the data. These were necessary functions required to solve this task since the aggregation function avg() could only be applied once the data was in a format ready to be processed. Furthermore, by plotting the histogram and analysing the data, this can benefit rideshare companies as well as drivers as it can enable them to allocate resources more efficiently to combat the high waiting times. Companies can implement strategies such as boosting pay for the day so that more drivers are willing to work and thus reducing the wait time.

Task 6

Task explanation: Task 6 involves analysing the number of trips across different locations during various times of the day. Firstly, this task aims to identify trip counts greater than 0 and less than 1000 for different pickup borough at different times of day. Secondly, calculating the number of trips for each borough during the evening. Lastly, determining the number of trips from Brooklyn to Staten Island and showing 10 samples. This task uses the taxi zone lookup which we joined to the rideshare dataset in task 1 which contains all the borough, pickup and drop off information.

APIs and functions used in code:

- **Data aggregations (groupby, orderby count) & data preparation functions:**
    - **[6.1]** The first task begins by using the groupby() function to group the DataFrame by 2 columns, ‘Pickup_Borough’ and ‘time_of_day’. Then it aggregates the grouped data by using the count() function to count the number of trips for each group, followed by the alias() function which names the resulting column as ‘trip_count’. Then filter() function combined with the col() function is used to extract ‘trip_count’ values only between 0 and 1000 where the ‘&’ is used to combine these conditions. Finally, using the orderBy() function, the filtered data is ordered by ‘Pickup_Borough’ and ‘time_of_day’
    - **[6.2]** The next task uses the filter() function combined with the col() function to filter "time_of_day" to "evening" for each row of the DataFrame. This is followed by using the groupby() function to group both ‘Pickup_Borough’ and ‘time_of_day’ again, where the agg() function is used with the count() function to count the number of trips for each ‘Pickup_Borough’, followed by the alias() function which names the resulting column as ‘trip_count’. Finally, using the orderBy() function, the filtered data is ordered by ‘Pickup_Borough’ in ascending order.
    - **[6.3]** This final task begins by using the filter() function to specify the ‘Pickup_borough’ column using col() as ‘Brooklyn’ and the ‘Dropoff_borough’ column as ‘Staten Island’. Then using the select() function we specified that we only want to include the columns ‘Pickup_borough’, ‘Dropoff_borough’, and ‘Pickup_zone’ in the DataFrame.

- **Displaying results:** Finally the show() function is used to display the dataframes with their requested amount of records. In task 6.3, the code was used .show(10, truncate=False) to show 10 samples and to set truncate to false.

Visualisation of results:

Pickup_Borough	time_of_day	trip_count
EWR	afternoon	2
EWR	morning	5
EWR	night	3
Unknown	afternoon	908
Unknown	evening	488
Unknown	morning	892
Unknown	night	792

6.1 trip count between 1-1000

Pickup_Borough	time_of_day	trip_count
Bronx	evening	1380355
Brooklyn	evening	3075616
Manhattan	evening	5724796
Queens	evening	2223003
Staten Island	evening	151276
Unknown	evening	488

6.2 trip count evening time

Total trips from Brooklyn to Staten Island: 69437

Pickup_Borough	Dropoff_Borough	Pickup_Zone
Brooklyn	Staten Island	DUMBO/Vinegar Hill
Brooklyn	Staten Island	Dyker Heights
Brooklyn	Staten Island	Bensonhurst East
Brooklyn	Staten Island	Williamsburg (South Side)
Brooklyn	Staten Island	Bay Ridge
Brooklyn	Staten Island	Bay Ridge
Brooklyn	Staten Island	Flatbush/Ditmas Park
Brooklyn	Staten Island	Bay Ridge
Brooklyn	Staten Island	Bath Beach
Brooklyn	Staten Island	Bay Ridge

only showing top 10 rows

6.3 Brooklyn to Staten & trip count

Challenges faced:

Filtering: Accurately filtering the data to attain the desired criteria felt tedious. Especially in task 6.3 where I had to calculate the number of trips that started in Brooklyn and ended in Staten Island. This task seemed difficult at first glance as I was confused to which function to use. I overcame this challenge by realised a simple filter function can be used to assign the pickup borough to Brooklyn and using the '&' operator I could also filter the drop off borough to Staten Island.

Knowledge and insights attained:

The insights gained from this task are valuable since it demonstrates how trip counts vary by the location and the time of day. Stakeholders can view this data and pick out which borough at which time is high in activity, which can enable rideshare companies to optimise where to allocate resources and be flexible in doing so. The knowledge I attained from this task is how to use the filter function more confidently, whether it is filtering with a single or between many column variables.

Task 7

Task explanation: This task aims to identify the top 10 most popular routes in terms of trip count for different routes between Uber and Lyft, comparing how the two rideshare firms compete across different routes. The main steps include creating a route column by concatenating pickup and drop off zones, aggregating the trip count for each unique route for Uber and Lyft and finally listing the top 10 routes by total trip count with separate counts for Uber and Lyft.

APIs and functions used in code:

- **Concatenating values:** Using the withColumn() function to create a new column named ‘route’, I generated the values within this column using the concat_ws() function which concatenated the values of the ‘Pickup_Zone’ column and the ‘Dropoff_Zone’ column, whilst separated by the string ‘to’.
- **Grouping & counting columns:** Applying the groupBy() function to group columns ‘Route’ and ‘business’.
- **Count aggregation:** Aggregating the grouped ‘Route’ and ‘business’ columns to a count() function to count the number of trips.
- **Separating counts:** using the groupBy() function to group the ‘Route’ column, the pivot() function is applied to the grouped dataframe and is applied to the ‘business’ column which takes unique values from the "business" column which are either ‘Uber’ of ‘Lyft’ and creates separate columns for each of these values.
- **Aggregation:** using the sum() function which is applied to ‘count’, it calculates the total count of Uber and the total count of Lyft.
- **Handling missing values:** The fillna(0) method is used to fill records with 0 if there are any missing combinations such as missing values for routes that don’t have any values for a business.
- **Renaming columns:** renaming the separated Uber and Lyft columns to ‘uber_count’ and ‘lyft_count’ using the withColumnRenamed() function for clarity.
- **Adding columns:** using the withColumn() function to create a new column named ‘total_count’ which is the sum total result of the ‘uber_count’ and ‘lyft_count’ columns for each row specified by the col() function.
- **Sorting & ranking routes:** using the orderBy() function based on the ‘total_count’ column to order it by descending order so that the highest count displays first.
- **Top 10 routes only:** applying the limit(10) function to giving the top 10 routes with the highest total counts only.
- **Displaying results:** Finally the show() function is used to display the dataframe with the top 10 routes, setting truncating to false the output to ensure that the entire contents of each cell, especially the routes column to ensure all columns are fully visible.

Visualisation of results:

Route	lyft_count	uber_count	total_count
JFK Airport to NA	46	253211	253257
East New York to East New York	184	202719	202903
Borough Park to Borough Park	78	155803	155881
LaGuardia Airport to NA	41	151521	151562
Canarsie to Canarsie	26	126253	126279
South Ozone Park to JFK Airport	1770	107392	109162
Crown Heights North to Crown Heights North	100	98591	98691
Bay Ridge to Bay Ridge	300	98274	98574
Astoria to Astoria	75	90692	90767
Jackson Heights to Jackson Heights	19	89652	89671

7.1 Top 10 routes per business

Challenges faced:

Separating trip counts by business: When tackling this task, I delved into it without knowing how to separate the trip counts for Uber and Lyft into separate columns for each route. After doing some research online, I overcame this challenge by implemented the pivot() function after grouping the data by route, to execute accurate separation of Uber and Lyft counts.

Knowledge and insights attained:

The top 10 popular routes results offer useful insights into where the top amount of rideshare users travel and for separate rideshare businesses. It enables rideshare companies to understand in which routes the bulk of their trips reside which can help rideshare companies to make better informed decisions on where to allocate resources efficiently such as focusing strategies to influence more drivers to work in the pickup areas of the route to meet demand as well as understanding some behaviour of rideshare users. Additionally, rideshare companies can implement pricing strategies such as increasing price for the most popular and demanding routes to increase profitability.

Concluding this report:

This project on NYC Rideshare analysis has upskilled me in spark and on implementing the correct API functions to tackle tasks. In collaboration with the terminal, I have become more comfortable with handling big data processes. Furthermore, the analysis conducted on the results and understanding the results has sharpened my ability in obtaining valuable insights.