NYC Taxi Data

Big Data Processing

Christian Juresh

210517307

Electronic Engineering and Computer Science
Queen Mary University of London
December 2023

Contents

0 Introduction

All of the tasks were run with Spark App ID: spark-fa7a3c840f174d0c803675da0b85c4bd

1 Merging Datasets

```
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)
```

1.1 Answers

- **Yellow Taxi Zone DataFrame:**

  Rows: 22,197,190

  Columns: 22

- **Green Taxi Zone DataFrame:**

  Rows: 465295

  Columns: 23

1.2 Task

This task involved several steps of data manipulation and transformation using Spark.

1.3 Approach

1. **Dataframe Creation:** Created two separate dataframes, one for yellow cabs and one for green cabs.
2. **Data Joining:** Joined the dataframes with the taxi lookup zone table. This added pickup and dropoff locations to the table. PULocationID and DOLocationID were used as the keys.
3. **Renaming Columns:** Columns were renamed for clarity, such as Pickup_Borough and Dropoff_Zone
4. **Schema Verification:** Used printSchema() and printed out the number of columns and rows to ensure the dataframes were set up correctly.

1.4 Yellow Cab Merge Code

```
join_columns = ["PULocationID", "DOLocationID"]
yellow_taxi_zone_df = yellow_tripdata_df.join(taxi_zone_lookup, yellow_tripdata_df.PULocationID == taxi_zone_lookup.LocationID,
↪  'left').withColumnRenamed("Borough", "Pickup_Borough").withColumnRenamed("Zone", "Pickup_Zone").withColumnRenamed("service_zone",
↪  "Pickup_service_zone").drop("LocationID")
yellow_taxi_zone_df = yellow_taxi_zone_df.join(taxi_zone_lookup, yellow_taxi_zone_df.DOLocationID == taxi_zone_lookup.LocationID,
↪  'left').withColumnRenamed("Borough", "Dropoff_Borough").withColumnRenamed("Zone", "Dropoff_Zone").withColumnRenamed("service_zone",
↪  "Dropoff_service_zone").drop("LocationID")

yellow_taxi_zone_df.printSchema()

print("Yellow Taxi Zone DataFrame - Rows:", yellow_taxi_zone_df.count(), "Columns:", len(yellow_taxi_zone_df.columns))
```

2 Aggregation of Data

2.1 Task

The task involved visualising data from yellow and green taxi services in New York City, broken down by boroughs for both pickups and dropoffs.

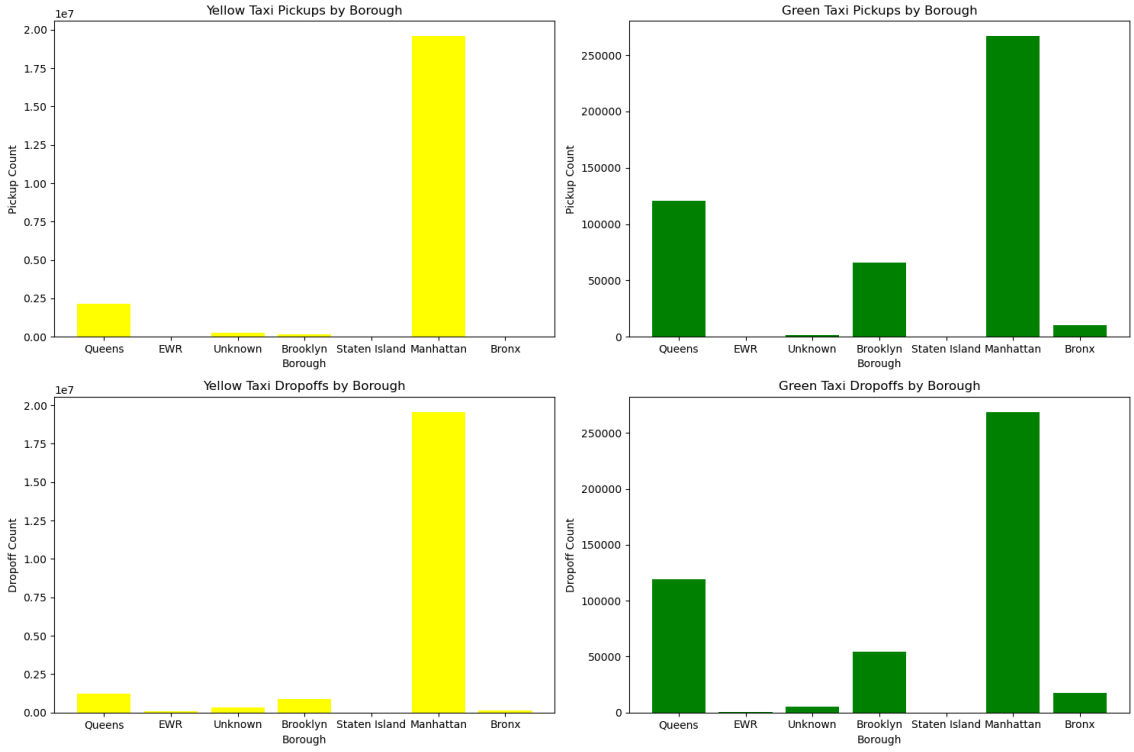
2.2 Approach

1. **Data Aggregation:** The data were grouped by boroughs for both pickups and dropoffs by using the groupBy() function.

2. **Collection of Results:** The count() function was applied to tally the number of pickups and dropoffs in each borough, and the results were collected into local memory using the collect() function

2.3 Visualisation

The data were used to create bar graphs, with each graph representing either dropoffs or pickups for yellow or green taxis. Bar graphs are an effective way to visualise and make comparisons between categorical data.



2.4 Challenges

Scale Difference: The scale of the yellow and green taxi trip counts is significantly different, which can make visual comparisons difficult. There are almost 80x more yellow taxi pickups than green taxi pickups in Manhattan. For the sake of keeping the green chart results visible, the y scale values are not kept the same between charts.

2.5 Insights

The visualisation helps to quickly identify which boroughs have the highest and lowest demand for taxi services. Manhattan has extremely high usage of taxis, and unlike yellow taxis, green taxis have a higher share of their rides outside of Manhattan, in places such as Queens, Brooklyn, and the Bronx. This sort of data would be useful for deciding resource allocation.

2.6 Aggregation Code

```
yellow_pickup_counts = yellow_taxi_zone_df.groupBy('Pickup_Borough').count().collect()
green_pickup_counts = green_taxi_zone_df.groupBy('Pickup_Borough').count().collect()
yellow_dropoff_counts = yellow_taxi_zone_df.groupBy('Dropoff_Borough').count().collect()
green_dropoff_counts = green_taxi_zone_df.groupBy('Dropoff_Borough').count().collect()
```

3 Filtering Data

3.1 Task

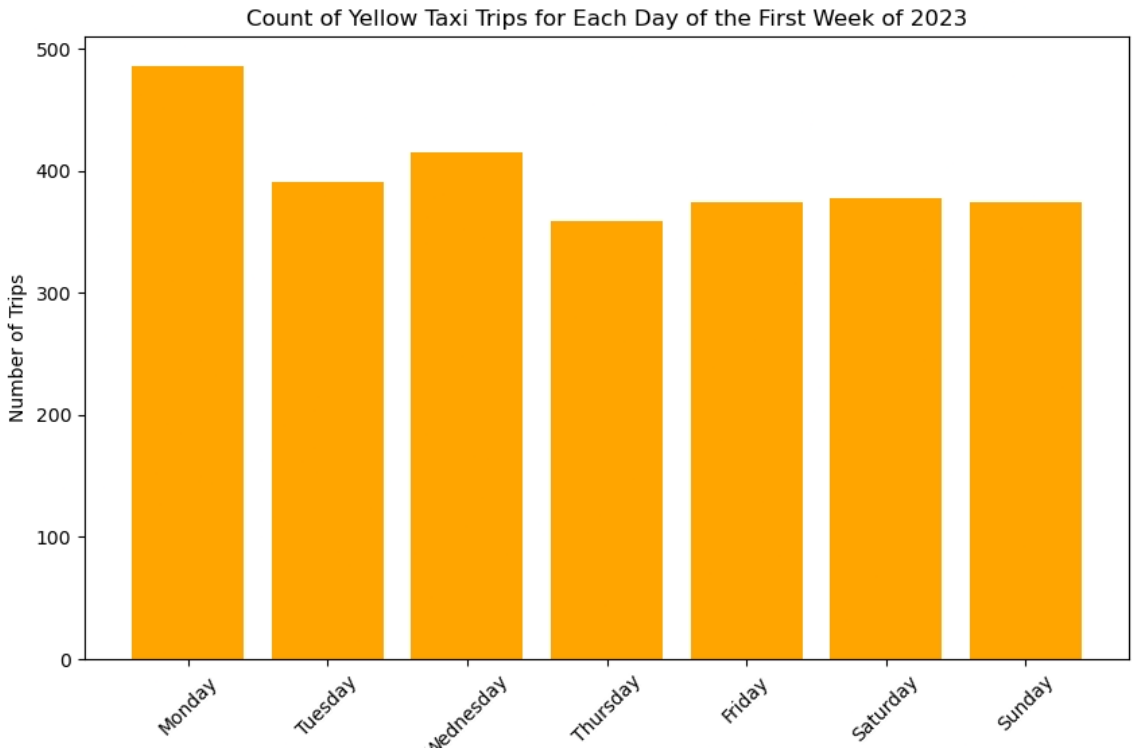
This task involved filtering the data based on certain conditions and visualising the results to understand the distribution of trips across different days of the week.

3.2 Approach

1. **Data Filtering:** A User-defined Function (UDF) filter_condition was created to filter out trips based on three conditions.
2. **Data Transformation:** The filter_udf function was applied to the dataset to retain only the rows that met specific criteria. Additional columns for date of pickup and day of the week were then added using withColumn() to help group the data by data later in the code.
3. **Aggregation:** The filtered data was then grouped by the day of the week column, and the count of trips for each day was calculated.

3.3 Visualisation

The aggregated data was used to create another bar chart like in Section 2. with days of the week on the x axis and number of trips on the y axis.



3.4 Insights

There is a peak in activity on Mondays with these filtered conditions. This data is likely filtered in this way to allow authorities to determine which days are optimal for investigating suspicious trips.

3.5 Filtering Code

```
def filter_condition(fare, distance, pickup_datetime):
    fare = float(fare)
    distance = float(distance)
    pickup_date = datetime.strptime(pickup_datetime, "%Y-%m-%d %H:%M:%S").date()
    return fare > 50 and distance < 1 and pickup_date >= datetime(2023, 1, 1).date() and pickup_date <= datetime(2023, 1, 7).date()

filter_udf = udf(filter_condition, BooleanType())

yellow_filtered_df = yellow_tripdata_df.filter(filter_udf(yellow_tripdata_df['fare_amount'], yellow_tripdata_df['trip_distance'],
↪  yellow_tripdata_df['tpep_pickup_datetime']))

yellow_filtered_df = yellow_filtered_df.withColumn("pickup_date", to_date("tpep_pickup_datetime", "yyyy-MM-dd"))
yellow_filtered_df = yellow_filtered_df.withColumn("day_of_week", dayofweek("pickup_date"))

daywise_trip_counts = yellow_filtered_df.groupBy("day_of_week").count().collect()
```

4 Top-K Processing

4.1 Task

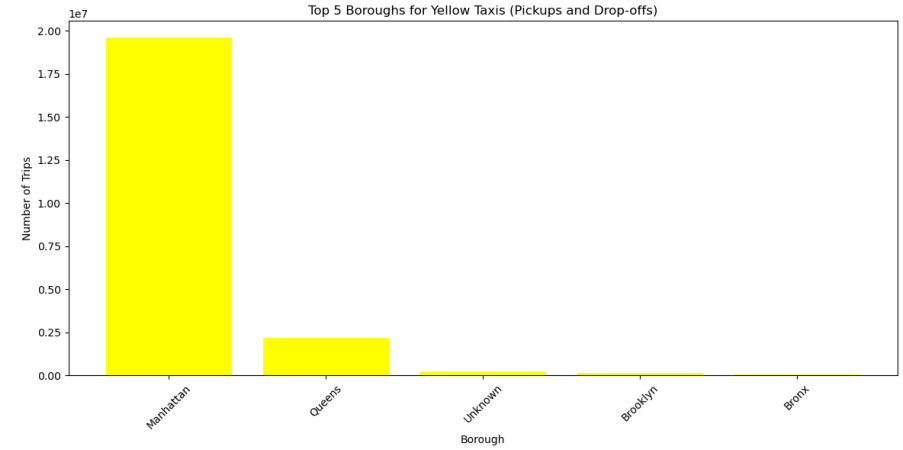
This task involved identifying and visualising the top 5 boroughs for taxi trips.

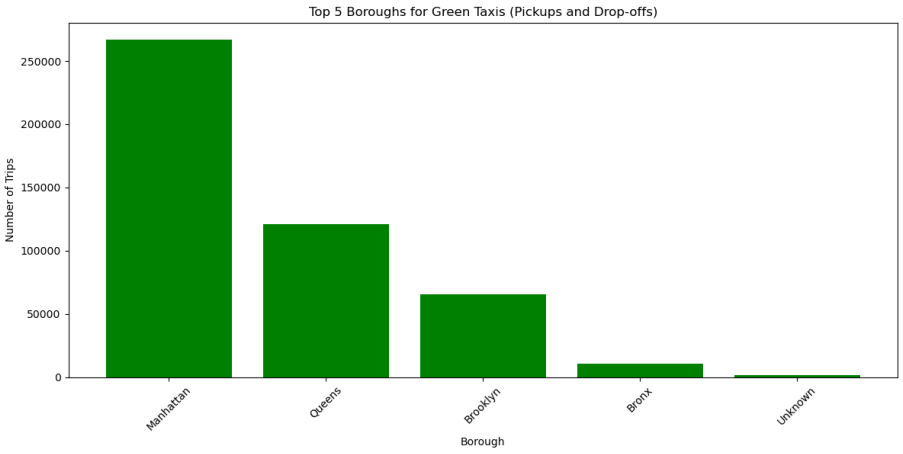
4.2 Approach

1. **Data Aggregation:** Used the groupBy() function to aggregate data.
2. **Sorting and Limiting Results:** The counts were ordered in descending order using the orderBy() function. limit() was then used to restrict the results to the top 5 boroughs.

4.3 Visualisation

Two bar plots were created to visualise the data, one for each taxi color. This approach is similar to the methods described in Sections 2 and 3.





4.4 Challenges

See Section 2.4

4.5 Insights

The visualisation is very similar to the one in Section 2. However, an ordered, filtered chart allows focusing in on a more specific set of results.

4.6 Top-K Processing Code

```
yellow_pickup_top5 = yellow_taxi_zone_df.groupBy('Pickup_Borough').count().orderBy(desc('count')).limit(5)
yellow_dropoff_top5 = yellow_taxi_zone_df.groupBy('Dropoff_Borough').count().orderBy(desc('count')).limit(5)

green_pickup_top5 = green_taxi_zone_df.groupBy('Pickup_Borough').count().orderBy(desc('count')).limit(5)
green_dropoff_top5 = green_taxi_zone_df.groupBy('Dropoff_Borough').count().orderBy(desc('count')).limit(5)
```

5 Finding Anomalies

5.1 Task

This task involved identifying any anomalies in daily trip counts from a filtered dataset.

5.2 Approach

1. **Data Filtering:** The dataset was first filtered to include only the trips from June 2023 using filter().
2. **Daily Aggregation:** The groupBy() function was used to aggregate the data day by day, counting the number of trips for each day in June.
3. **Statistical Analysis:** The first and third quartiles of the daily trip counts was calculated using the approxQuantile() function. The interquartile range (IQR) was then determined from these values.
4. **Anomaly Detection:** A lower and upper bound for anomalies was calculated using 1.5*IQR below Q1 and above Q3. Each days trip count was compared to these bounds, and a label was given accordingly, with "High" for counts above the upper bound, and "Low" for counts below the lower bound. All other values would be assigned to "Normal"

5.3 Visualisation

The data was collected and used to create a scatter plot. This type of plot is very effective in displaying the relationship between two continuous variables with individual data points. It is great for showing outliers, trends, and clusters within data.

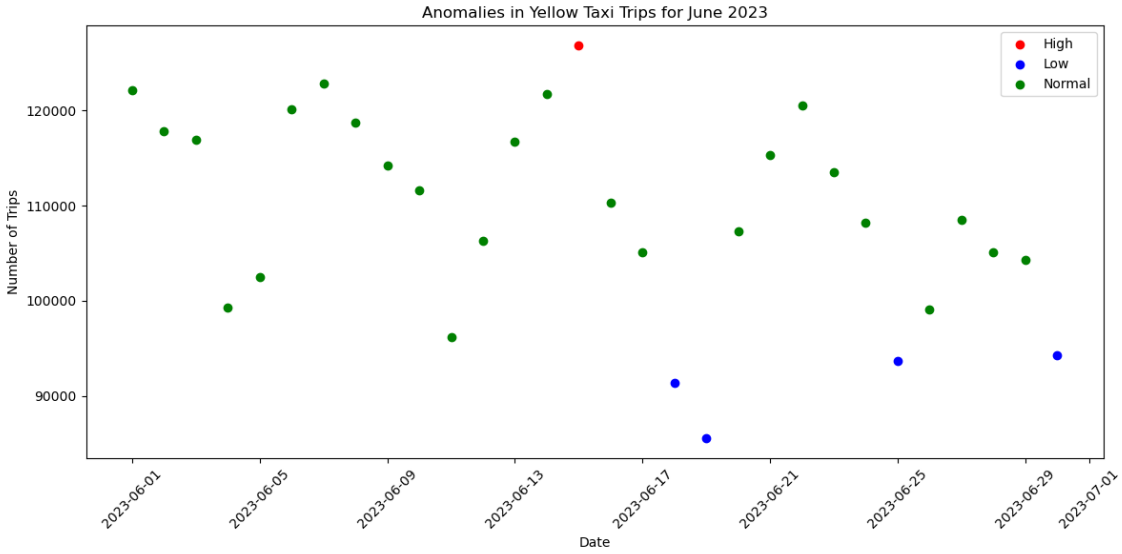
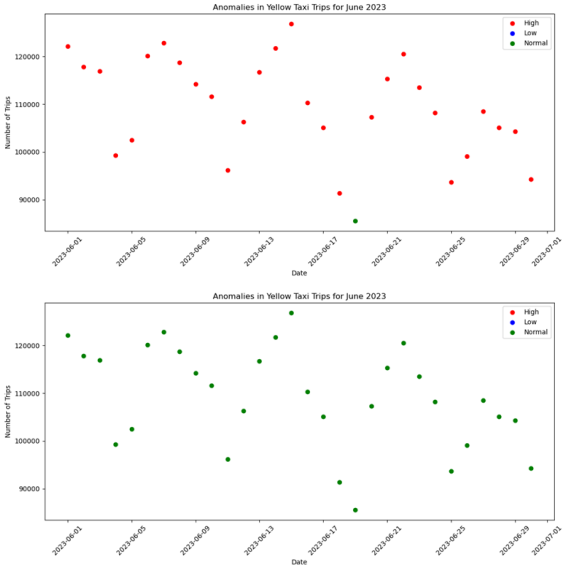


Figure 1: Low <40th High >60th

5.4 Challenges

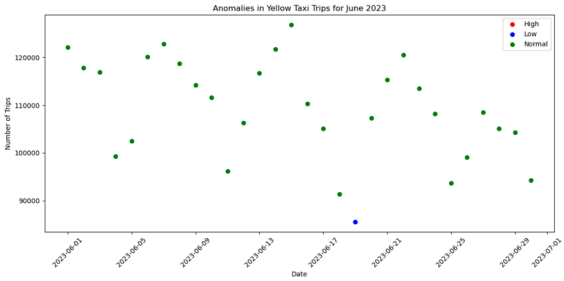


The method suggested in the Task instructions and the Student Forum is not suitable for this dataset if taken literally. Almost all of the data points exceed 80% of the mean, making them fall into the "High" anomaly category. Since the variability is low, the thresholds are too close to the center of the data, leading to a misclassification of regular data points as anomalies.

(mean x 0.2) = Low Threshold (mean x 0.8) = High Threshold

If we use 20 and 80 as our percentiles for continuity from the initial suggested range, all values appear as normal. This is a reasonable conclusion to stop at as the range of values is not that large.

Low <20th High >80th



We only begin to see a single low outlier at around the 30th and 70th percentile. This brings us in line with the original mean calculation, except with the values set correctly this time.

Low <30th High >70th

5.5 Insights

In the visualisation in Figure 1, we can see a few low outliers, and one high outlier. However this chart uses ranges that somewhat stretch the definition of an outlier. This is discussed further in Section 5.4.

Taxi companies can use these insights to adjust their pricing strategies on certain days with expected high or low demand, such as special events or public holidays.

5.6 Anomalies Mean Code Snippet

```
mean_trips_june = daily_trips_june.agg(avg("count").alias("mean_trips")).collect()[0]["mean_trips"]

high_threshold_june = mean_trips_june * 0.8
low_threshold_june = mean_trips_june * 0.2
```

5.7 Anomalies IQR Code

```
yellow_june_2023_df = yellow_taxi_zone_df.filter(
    (year(col("tpep_pickup_datetime")) == 2023) &
    (month(col("tpep_pickup_datetime")) == 6)
)

daily_trips_june = yellow_june_2023_df.groupBy(date_format(col("tpep_pickup_datetime"), "yyyy-MM-dd").alias("date")).count()

stats = daily_trips_june.approxQuantile("count", [0.4, 0.6], 0)
Q1, Q3 = stats
IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

daily_trips_june = daily_trips_june.withColumn(
    "anomaly",
    when(col("count") > upper_bound, "High")
    .when(col("count") < lower_bound, "Low")
    .otherwise("Normal")
)

daily_trips_plot_data_june = daily_trips_june.select("date", "count", "anomaly").orderBy("date").collect()
```

6 Statistical Processing

6.1 Task

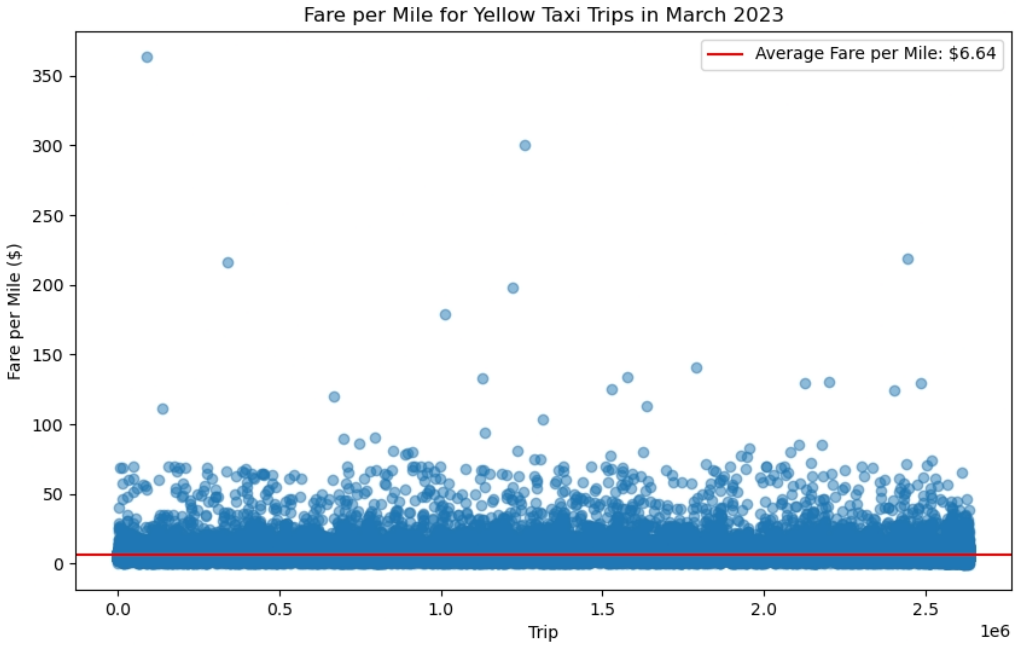
This task involved calculating an average fare per mile from a filtered dataset, along with checking for any outliers.

6.2 Approach

1. **Data Filtering:** The yellow taxi dataset was first filtered to only include trips from March 2023.
2. **Data Cleaning:** Trips with a distance of zero miles or less were removed. This ensured that trips that never occurred would not skew the average calculation.
3. **Calculation:** A new column, fare_per_mile, was created by dividing the fare amount by the trip distance. This allows comparison.
4. **Average Fare Calculation:** The average fare per mile was calculated across all trips using the avg() function.

6.3 Visualisation

A scatter plot was created with individual trips on the x axis and fare per mile on the y axis. The reason for this is mentioned in Section 5.3. A horizontal line was drawn to represent the average fare per mile, providing a reference point against which to identify outliers.



6.4 Insights

Outliers begin above the $80 per mile point, where the frequency of points begins to decrease.

These values can be used to find data entry errors, unusually short trips with high fares, or other anomalies. The average far per mile provides a baseline to calculate the cost-effectiveness of the taxi service.

6.5 Statistical Processing Code

```
yellow_march_2023_df = yellow_taxi_zone_df.filter(
    (year(col("tpep_pickup_datetime")) == 2023) &
    (month(col("tpep_pickup_datetime")) == 3)
)

yellow_march_2023_df = yellow_march_2023_df.filter(col("trip_distance") > 0)

yellow_march_2023_df = yellow_march_2023_df.withColumn("fare_per_mile", col("fare_amount") / col("trip_distance"))

average_fare_per_mile = yellow_march_2023_df.agg(avg("fare_per_mile").alias("avg_fare_per_mile")).collect()[0]["avg_fare_per_mile"]

fare_per_mile_data = yellow_march_2023_df.select("fare_per_mile").collect()
```

7 Data Distribution

7.1 Answer

- Percentage of solo trips in Yellow Taxi data: 73.12%
- Percentage of solo trips in Green Taxi data: 79.07%

7.2 Task

This task involved analysing the distribution of solo passenger trips in the datasets.

7.3 Approach

1. **Data Counting:** The total number of trips for both datasets was determined using the count() function
2. **Filtering Solo Trips:** Within each dataset, trips where the passenger_count was equal to 1 were isolated using the filter() function.
3. **Calculating Percentages:** The percentage of solo trips was calculated by dividing the number of solo trips by the total number of trips, then multiplying by 100.

7.4 Visualisation

These results did not need to be visualised. Since they are simply two numerical values that are close together, a plot doesn't make much sense. Even a bar chart would not demonstrate a great difference between the two values.

7.5 Insights

Green taxi trips have solo passengers more often. Yellow taxis are likely used more often for recreational purposes in groups, while green taxis are predominantly used by solo commuters

These data can be valuable in planning which taxi sizes should be deployed. By comparing green and yellow taxis, it is possible to look for differences between the two services in passenger numbers.

7.6 Distribution Code

```
total_yellow_trips = yellow_tripdata_df.count()
solo_yellow_trips = yellow_tripdata_df.filter(col("passenger_count") == 1).count()
percentage_solo_yellow = (solo_yellow_trips / total_yellow_trips) * 100

total_green_trips = green_tripdata_df.count()
solo_green_trips = green_tripdata_df.filter(col("passenger_count") == 1).count()
percentage_solo_green = (solo_green_trips / total_green_trips) * 100

print(delimiter)
print(f"Percentage of solo trips in Yellow Taxi data: {percentage_solo_yellow:.2f}%")
print(f"Percentage of solo trips in Green Taxi data: {percentage_solo_green:.2f}%")
print(delimiter)
```

10 Time-window based Analysis

10.1 Task

This task involved identifying the months with the highest number of taxi trips for yellow and green taxis.

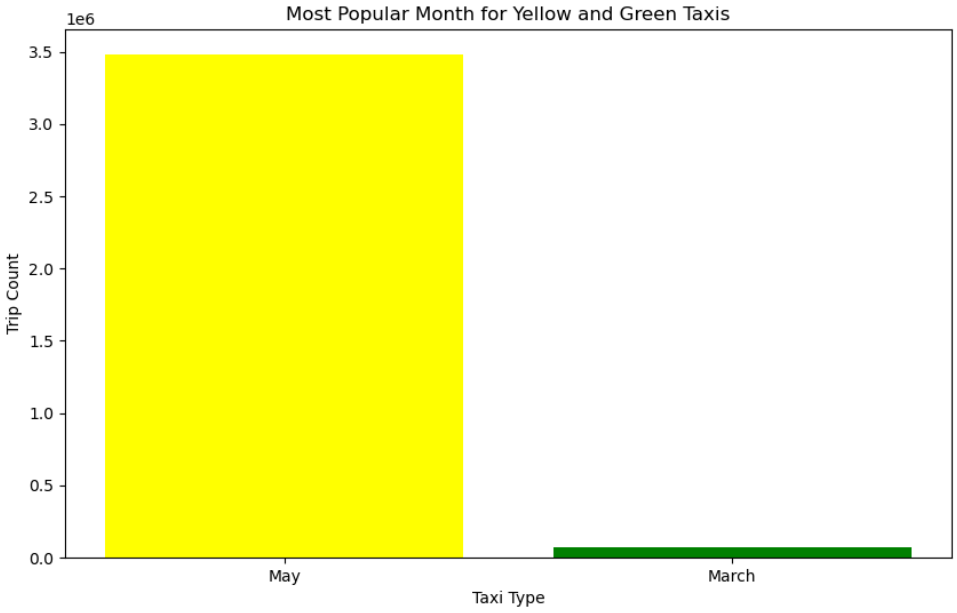
10.2 Approach

1. **Data Aggregation:** For each dataset, the data were grouped by month using the groupBy() function applied to the pickup datetime column. This organised the trips by the month they were taken

2. **Identifying the Peak Month:** The orderBy() function sorted the grouped data in descending order based on the trip count, and the first() function filtered the entry with the highest count.

10.3 Visualisation

The data was plotted as a bar plot for the same reasons as in Section 2.3. However, this time the y scale is the same for yellow and green taxis. This is because there are only two values and we are trying to have a direct comparison between them.



10.4 Insights

May was the most popular month for yellow taxis, while March was for green taxis. Yellow taxis have a significantly higher number of trips compared to green taxis. This was confirmed in Section 2. however it is the first time the difference has been visualised.

This data may be useful for taxi companies to understand resource allocation needs.

10.5 Time-window Code

```
def most_popular_month(dataframe, pickup_datetime_column, taxi_type):
        monthly_trips = dataframe.groupBy(month(col(pickup_datetime_column)).alias("month")).count()
        most_trips_month = monthly_trips.orderBy(desc("count")).first()
        return {"taxi_type": taxi_type, "month": most_trips_month["month"], "trips": most_trips_month["count"]}

    most_popular_month_yellow = most_popular_month(yellow_tripdata_df, "tpep_pickup_datetime", "yellow_taxi")
    most_popular_month_green = most_popular_month(green_tripdata_df, "lpep_pickup_datetime", "green_taxi")
```