Report: Optimising NYC Taxi Operations

Include your visualizations, analysis, results, insights, and outcomes. Explain your methodology and approach to the tasks. Add your conclusions to the sections

1. Data Preparation

1.1 Loading the dataset

Step 1: Import necessary libraries and set up the working directory

Step 2: Create a list of all files in the directory and initialize an empty dataframe.

Step 3: Iterate through each file, read it, and preprocess the data.

Step 6: Sample 0.7% of the data for each hour of each date.

Step 7: Concatenate all sampled data into the final dataframe.

Step 8: Save the final sampled dataset and download it.

2. Data Cleaning

2.1 Fixing Columns

2.1.1 Fix the index

2.1.2 Combined the two airport_fee columns

Combine two columns (airport_fee and Airport_fee) into a single column (Airport_Fee) and drop the original columns.

Handling Missing Values

2.1.2 Find the proportion of missing values in each column

I calculate the percentage of missing values in each column of the dataframe.

2.1.3 Handling missing values in passenger_count

Step 1: Filled missing values in the passenger_count column with the median.

```
df['passenger_count'].fillna(df['passenger_count'].median(), inplace=True)
```

Missing values in the passenger_count column are replaced with the median value of the column.

2.1.4 Handle missing values in RatecodeID

Step 1: Filled missing values in the RatecodeID column using the median or mean based on data skewness.

```
df['RatecodeID'].fillna(df['RatecodeID'].median() if df['RatecodeID'].skew() > 1 else df['RatecodeID'].mean(), inplace=True)
```

If the RatecodeID column is highly skewed (skew() > 1), missing values are filled with the median.

Otherwise, the mean is used

2.1.5 Impute NaN in congestion_surcharge

Step 1: Filled missing values in the congestion_surcharge column using the median or mean based on data skewness. Same approach is used as used in RatecodeID

If the congestion_surcharge column is highly skewed (skew() > 1), missing values are filled with the median.

Otherwise, the mean is used

2.2 Handling Outliers and Standardising Values

2.2.1 Check outliers in payment type, trip distance and tip amount columns

Step 1: Detect outliers in numeric columns (e.g., payment_type, trip_distance, tip_amount) using the Interquartile Range (IQR) method.

I created a function called detect_outliers_iqr to identify outliers in numeric columns using the Interquartile Range (IQR) method.

```
def detect_outliers_iqr(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = column[(column < lower_bound) | (column > upper_bound)]
    return outliers
```

This function calculates the first quartile (Q1), third quartile (Q3), and the IQR.

It then defines the lower and upper bounds for outliers as Q1 - 1.5 * IQR and Q3 + 1.5 * IQR, respectively.

Any values outside these bounds are flagged as outliers.

Imputing Outliers Using IQR

I created a function called impute_outliers_iqr to replace outliers in numeric columns with the median value.

```
def impute_outliers_iqr(column):
    Q1 = column.quantile(0.25)
    Q3 = column.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    median_value = column.median()

    return column.apply(lambda x: median_value if x < lower_bound or x >
    upper_bound else x)
```

This function calculates the first quartile (Q1), third quartile (Q3), and the Interquartile Range (IQR).

It defines the lower and upper bounds for outliers as Q1 - 1.5 * IQR and Q3 + 1.5 * IQR, respectively.

Any value outside these bounds is replaced with the median of the column.

3. Exploratory Data Analysis

3.1 General EDA: Finding Patterns and Trends

3.1.1 Classify variables into categorical and numerical

I separated the columns of the dataframe into numerical and categorical types for easier analysis and processing.

```
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns.tolist()

categorical_columns = df.select_dtypes(include=['object','category']).columns.tolist()
```

numerical_columns: This list contains all columns with numerical data types (int64 and float64), such as trip_distance, fare_amount, etc.

categorical_columns: This list contains all columns with categorical data types (object and category), such as payment_type, store_and_fwd_flag, etc

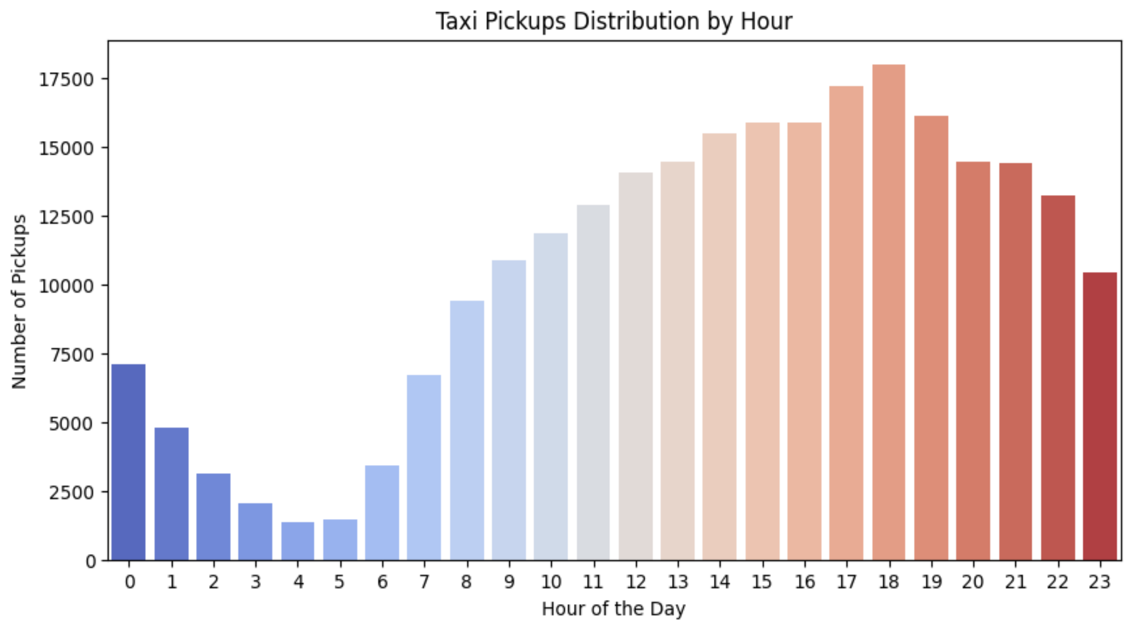
3.1.2 Analyse the distribution of taxi pickups by hours, days of the week, and months

Analyzing Taxi Pickups by Hour

Step 1: I grouped the data by the hour column to calculate the number of taxi pickups for each hour of the day.

```
hourly_trips = df.groupby('hour').size().reset_index(name='trip_count')

print(hourly_trips)
```

Step 2: I created a visualization to show the distribution of taxi pickups by hour.



Analyzing Daily Trends in Taxi Pickups

Step 1: I extracted the day of the week from the tpep_pickup_datetime column to analyze daily trends.

```
df['pickup_dayofweek'] = df['tpep_pickup_datetime'].dt.strftime('%A')
```

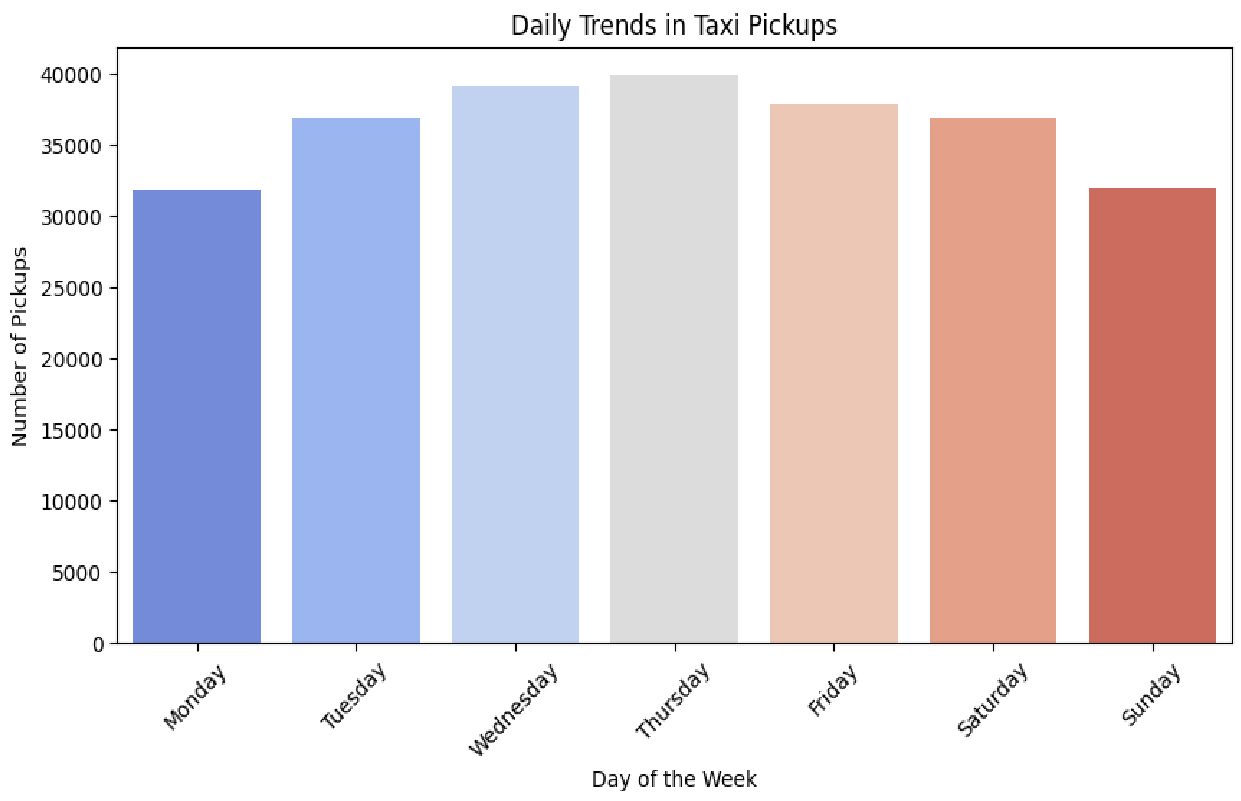
the strftime('%A') method converts the datetime values to the corresponding day of the week (e.g., Monday, Tuesday).

A new column, pickup_dayofweek, is created to store this information

Step 2: I counted the number of pickups for each day of the week and ordered them logically.

```
day_order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

daily_pickups = df['pickup_dayofweek'].value_counts().reindex(day_order)

print(daily_pickups)
```

Step 3: I created a bar plot to visualize the daily trends in taxi pickups



Analyzing Monthly Trends in Taxi Pickups

Step 1: I extracted the month from the tpep_pickup_datetime column to analyze monthly trends.

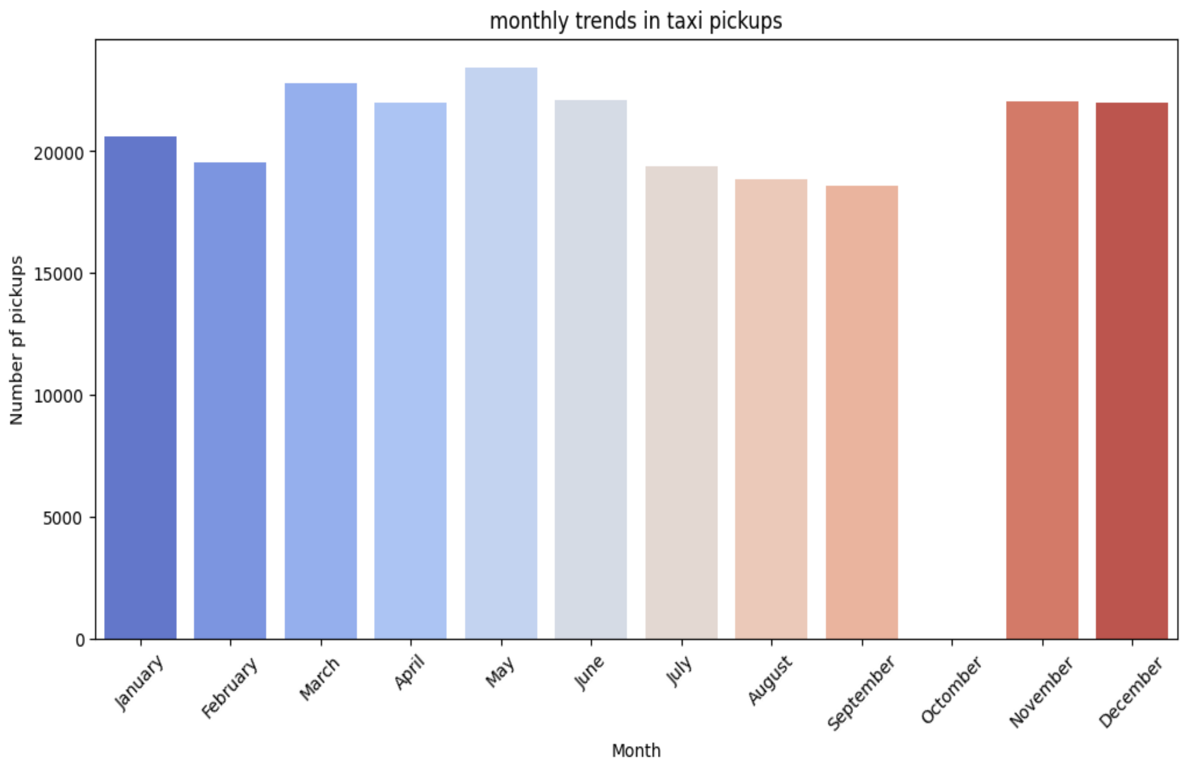
```
df['pickup_month'] = df['tpep_pickup_datetime'].dt.strftime('%B')
```

The strftime('%B') method converts the datetime values to the full month name (e.g., January, February).

A new column, pickup_month, is created to store this information.

Step 2: I counted the number of pickups for each month and ordered them chronologically.

step 3: I created a bar plot to visualize the monthly trends in taxi pickups.



3.1.3 Filter out the zero/negative values in fares, distance and tips

Step 1: I filtered out zero or negative values in the fare_amount, trip_distance, and tip_amount columns to ensure data quality.

3.1.4 Analyse the monthly revenue trends

Analyzing Monthly Revenue

Step 1: I extracted the month from the tpep_pickup_datetime column to analyze revenue trends by month.

Step 2: I grouped the data by month and calculated the total revenue for each month

```
monthly_revenue = df.groupby('pickup_month')['total_amount'].sum().reset_index()
```

The groupby('pickup_month') groups the data by the pickup_month column.

The sum() method calculates the total revenue (total_amount) for each month

Step 3: I converted the month numbers to month names for better readability.

```
import calendar

monthly_revenue['pickup_month'] =
monthly_revenue['pickup_month'].apply(lambda x: calendar.month_name[x])
```

Step 4: I reordered the monthly revenue data chronologically.

The final dataframe is printed, showing the total revenue for each month.

3.1.5 Find the proportion of each quarter's revenue in the yearly revenue

Calculating Quarterly Revenue Proportions

Step 1: I extracted the quarter from the tpep_pickup_datetime column to analyze revenue trends by quarter.

```
df['quarter'] = df['tpep_pickup_datetime'].dt.quarter
```

the dt.quarter method extracts the quarter (1 for Q1, 2 for Q2, etc.) from the tpep_pickup_datetime column.

A new column, quarter, is created to store this information.

Step 2: I grouped the data by quarter and calculated the total revenue for each quarter.

Step 3: I calculated the proportion of total revenue contributed by each quarter.

```
quarterly_revenue['proportion'] = (quarterly_revenue['total_amount'] /
quarterly_revenue['total_amount'].sum()) * 100

print(quarterly_revenue)
```

The proportion of revenue for each quarter is calculated by dividing the quarterly revenue by the total revenue and multiplying by 100.

This gives the percentage contribution of each quarter to the total revenue.

3.1.6 Analyse and visualise the relationship between distance and fare amount

Step 1: I filtered the data to include only trips with a positive trip_distance.

```
df_filtered = df[df['trip_distance'] > 0]
```

Calculating Correlation

Step 2: I calculated the correlation between trip_distance and fare_amount

```
correlation = df_filtered[['trip_distance', 'fare_amount']].corr().iloc[0, 1]

print(f"Correlation between trip_distance and fare_amount: {correlation:.2f}")
```

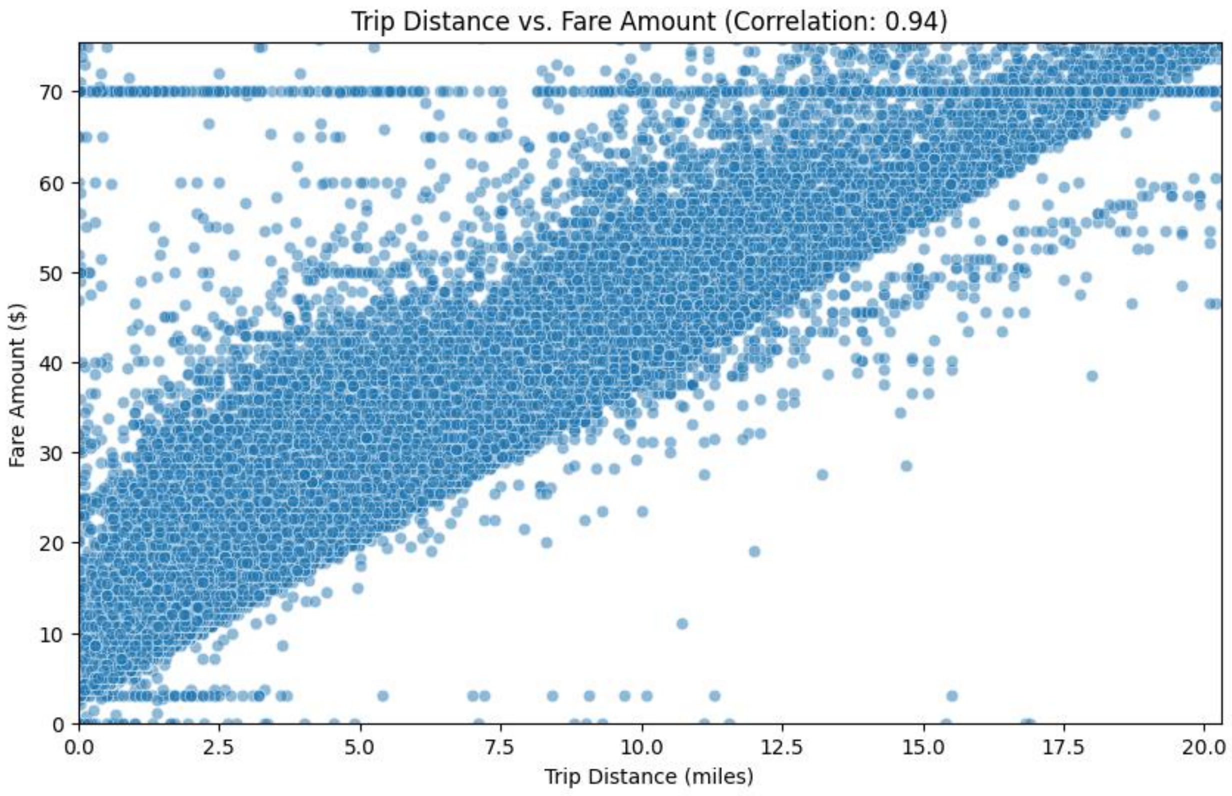
The correlation value (ranging from -1 to 1) indicates the strength and direction of the relationship:

A positive value indicates that as trip distance increases, fare amount tends to increase.

The value is rounded to two decimal places for clarity.

Visualizing the Relationship

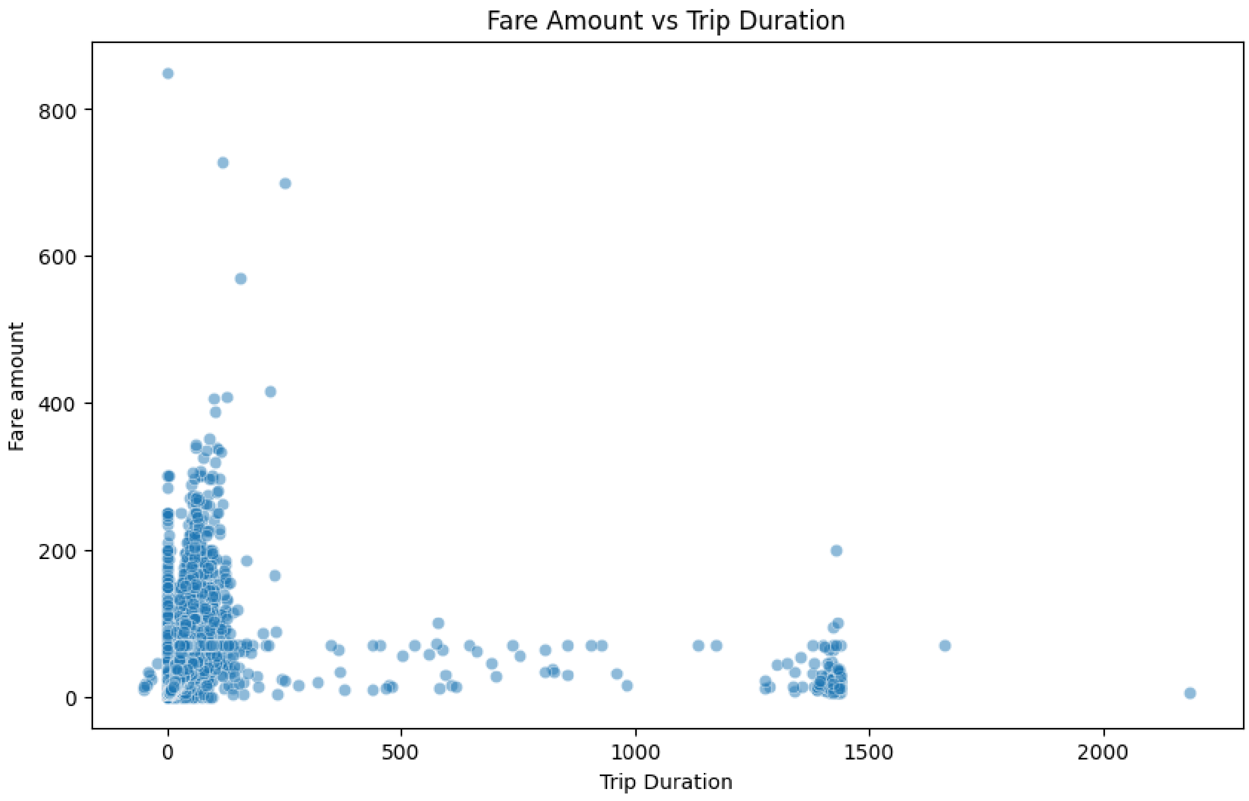
Step 3: I created a scatter plot to visualize the relationship between trip_distance and fare_amount.



3.1.7 Analyse the relationship between fare/tips and trips/passengers

Visualizing the Relationship

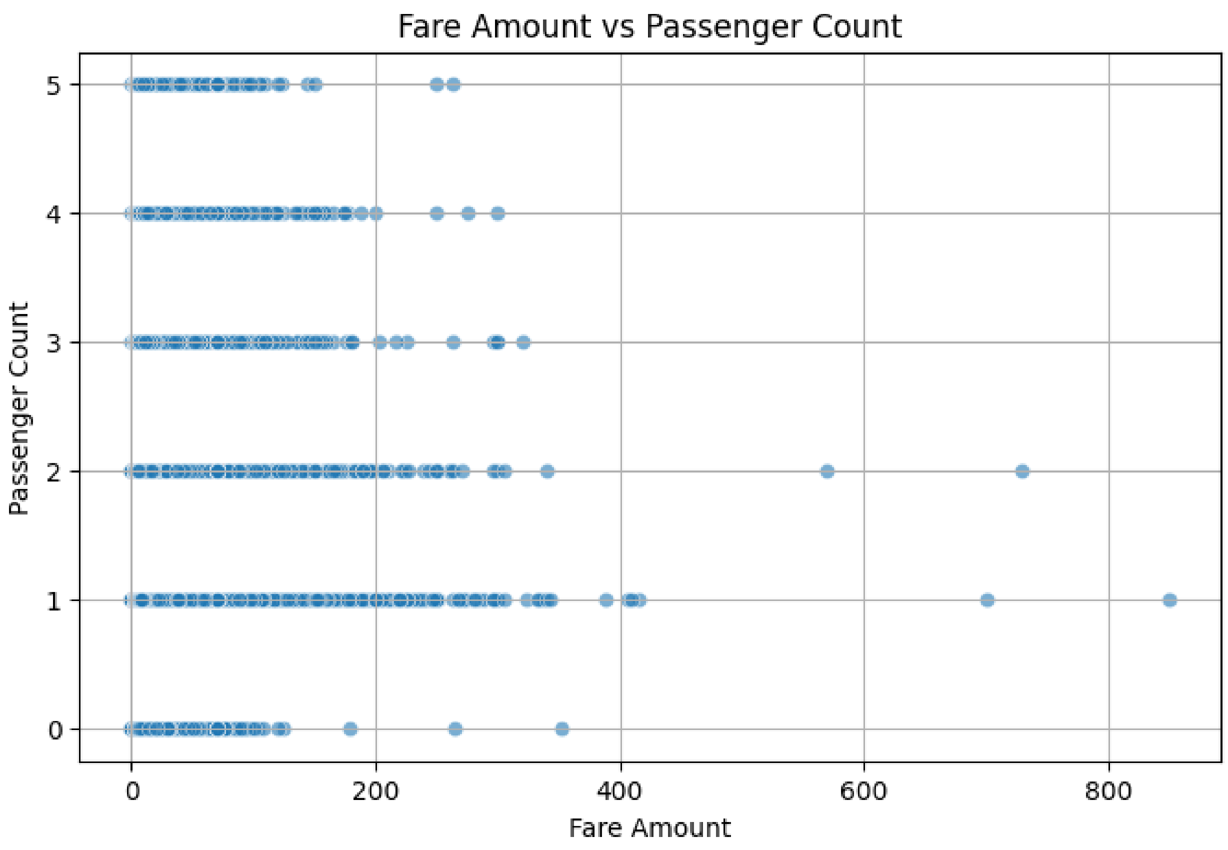
Step 1: I created a scatter plot to visualize the relationship between fare_amount and trip_duration.



This analysis provides insights into how trip duration impacts fare amounts, which can be useful for understanding pricing dynamics and customer behaviour

Visualizing the Relationship

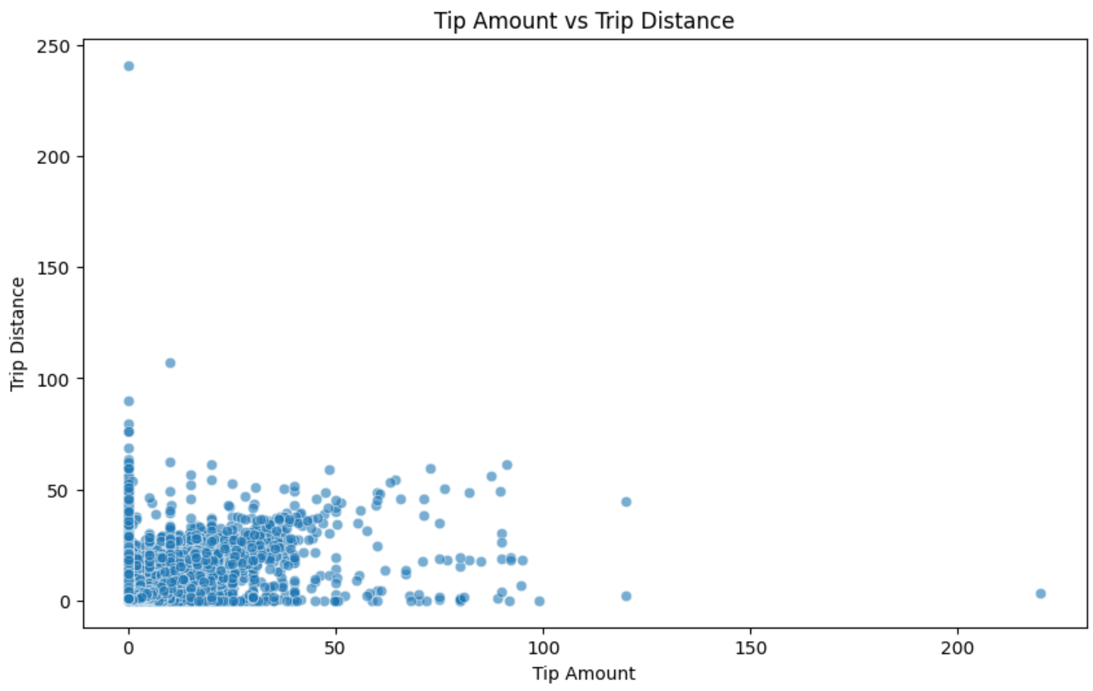
Step 1: I created a scatter plot to visualize the relationship between fare_amount and passenger_count.



This analysis provides insights into how the number of passengers impacts fare amounts, which can be useful for understanding pricing strategies and customer behaviour

Visualizing the Relationship

Step 1: I created a scatter plot to visualize the relationship between tip_amount and trip_distance.



This analysis provides insights into how trip distance impacts tipping behavior, which can be useful for understanding customer preferences and driver earnings.

3.1.8 Analyse the distribution of different payment types

Analyzing Payment Type Distribution

Mapping Payment Types to Descriptive Labels

Step 1: I mapped the numeric payment_type values to descriptive labels for better readability.

```
payment_mapping = {1: 'Credit Card', 2: 'Cash', 3: 'No Charge', 4: 'Dispute'}

df['payment_type'] = df['payment_type'].map(payment_mapping)
```

The payment_mapping dictionary converts numeric payment codes (1, 2, 3, 4) into descriptive labels (e.g., 1 → Credit Card, 2 → Cash).

This makes the data more intuitive for analysis and visualization.

Counting Payment Types

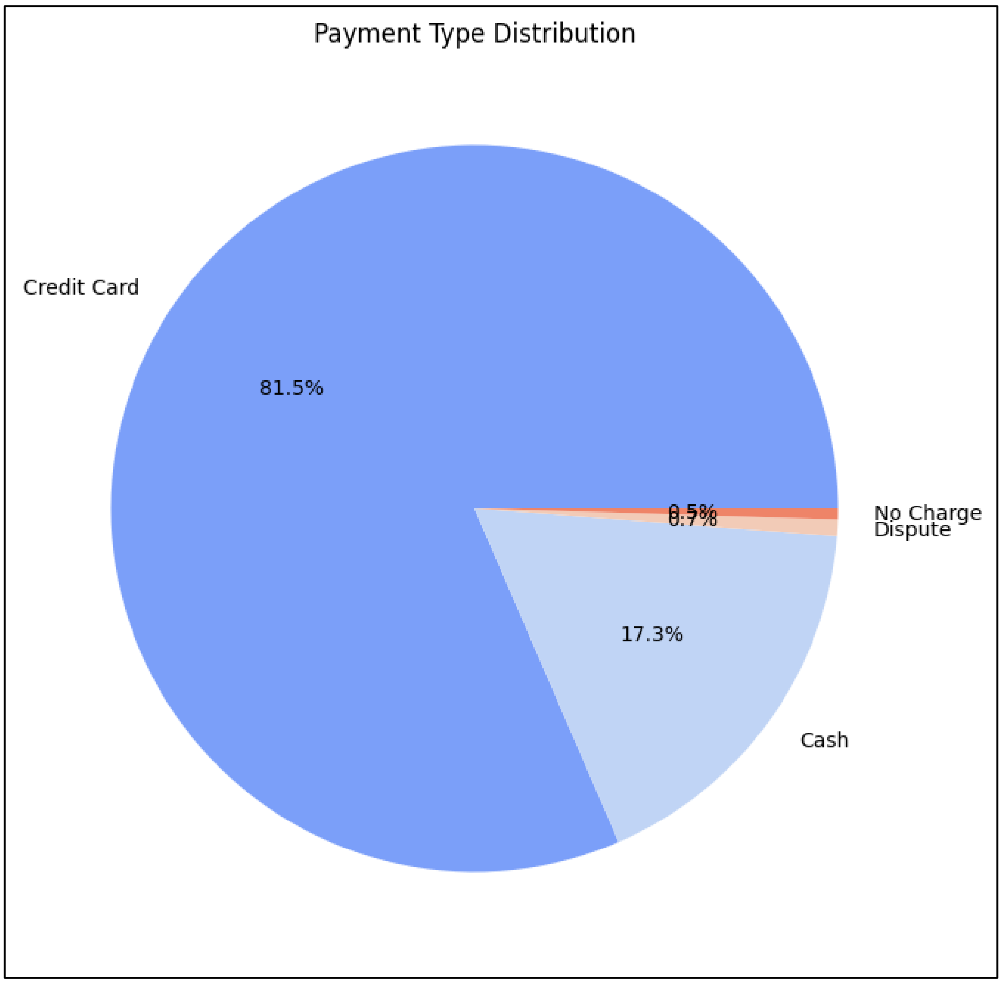
Step 2: I counted the occurrences of each payment type.

```
payments_counts = df['payment_type'].value_counts()

print(payments_counts)
```

The value_counts() method calculates the frequency of each payment type.

Visualizing Payment Type Distribution

Step 3: I created a pie chart to visualize the distribution of payment types.



3.1.9 Load the taxi zones shapefile and display it

Step 1: I mounted Google Drive to access the shapefile containing taxi zone data.

Step 2: I imported the geopandas library to work with geospatial data.

Step 3: I loaded the taxi zone shapefile into a geopandas dataframe.

```
zones =
gpd.read_file('/content/drive/MyDrive/DatasetsandDictionary/taxi_zones/taxi_zone
s.shp')zones.head()
```

3.1.10 Merge the zone data with trips data

Trip Records with Taxi Zones

Loading Trip Records and Taxi Zones

Step 1: I loaded the trip records and taxi zone data into dataframes.

```
df_trip_records = pd.read_parquet('final_sampling.parquet')
```

```
df_zones =
gpd.read_file('/content/drive/MyDrive/DatasetsandDictionary/taxi_zones/taxi_zone
s.shp')
```

df_trip_records contains the sampled trip data, including pickup location IDs (PULocationID).

df_zones contains the taxi zone data, including zone IDs (LocationID) and geographic boundaries.

Step 2: I merged the trip records with the taxi zone data using the pickup location ID (PULocationID).

```
df_merged = df_trip_records.merge(df_zones, left_on='PULocationID',
right_on='LocationID')
```

```
print(df_merged.head())
```

The merge() function combines the trip records (df_trip_records) with the taxi zone data (df_zones) based on the pickup location ID (PULocationID in trip records and LocationID in taxi zones).

3.1.11 Find the number of trips for each zone/location ID

Grouping Data by Pickup Location

Step 1: I grouped the trip records by pickup location (PULocationID) and calculated the number of trips for each location.

```
trip_counts =
df_trip_records.groupby('PULocationID').size().reset_index(name='total_trips')
```

```
trip_counts.sort_values(by='total_trips', ascending=False).head()
```

The groupby('PULocationID') groups the data by the pickup location ID.

The size() method counts the number of trips for each location.

The result is stored in a dataframe (trip_counts) with two columns: PULocationID and total_trips.

Grouping Data by Dropoff Location

Step 2: I grouped the trip records by dropoff location (DOLocationID) and calculated the number of trips for each location.

same approach which used above

This analysis helps identify the most popular pickup and dropoff locations, which can be useful for optimizing taxi operations and resource allocation.

3.1.12 Add the number of trips for each zone to the zones dataframe

Step 1: I renamed the columns in the trip_counts and drop_trip_counts dataframes to ensure consistency when merging.

```
trip_counts.rename(columns={'PULocationID': 'LocationID', 'total_trips':
'pickup_trips'}, inplace=True)
```

```
drop_trip_counts.rename(columns={'DOLocationID': 'LocationID', 'total_trips':
'dropoff_trips'}, inplace=True)
```

The rename() method is used to rename the columns:

In trip_counts, PULocationID is renamed to LocationID, and total_trips is renamed to pickup_trips.

In drop_trip_counts, DOLocationID is renamed to LocationID, and total_trips is renamed to dropoff_trips.

This ensures that the columns have consistent names for merging with the taxi zone data.

Step 2: I merged the pickup and dropoff trip counts with the taxi zone data.

```
df_zones = df_zones.merge(trip_counts, on='LocationID', how='left')

df_zones = df_zones.merge(drop_trip_counts, on='LocationID', how='left')

df_zones.fillna(0)

df_zones.head()
```

The merge() function combines the taxi zone data (df_zones) with the pickup trip counts (trip_counts) and dropoff trip counts (drop_trip_counts) using the LocationID column.

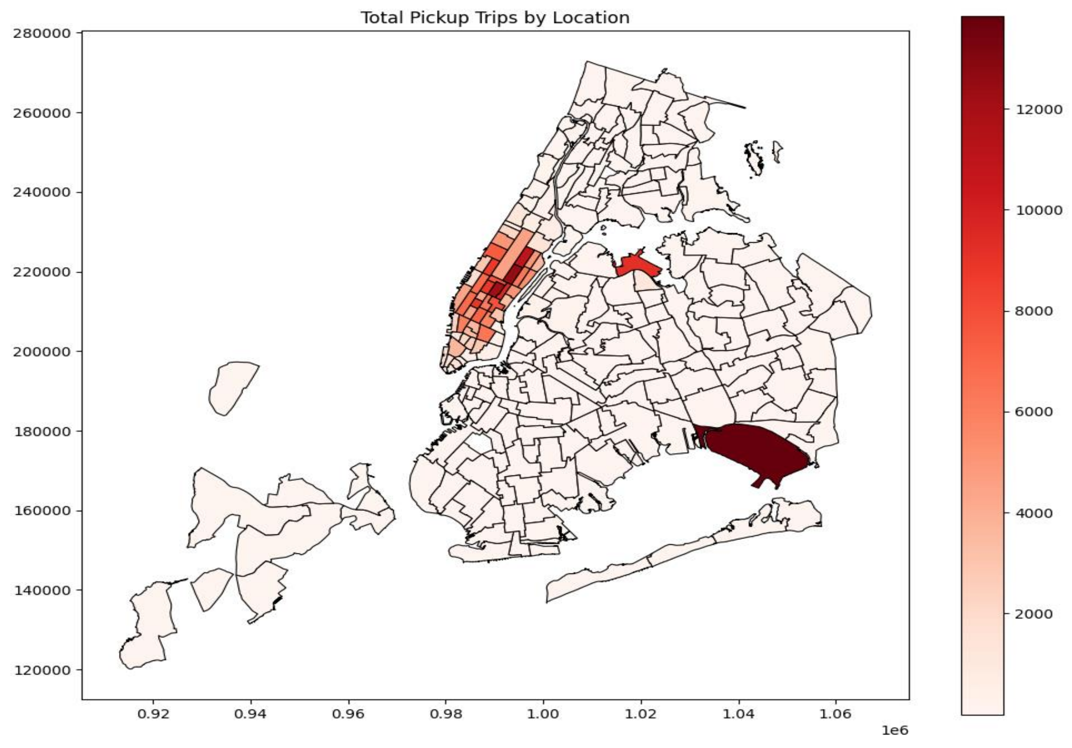
The how='left' parameter ensures that all taxi zones are retained, even if they have no trips.

Missing values (e.g., zones with no trips) are filled with 0 using fillna(0)

3.1.13 Plot a map of the zones showing number of trips

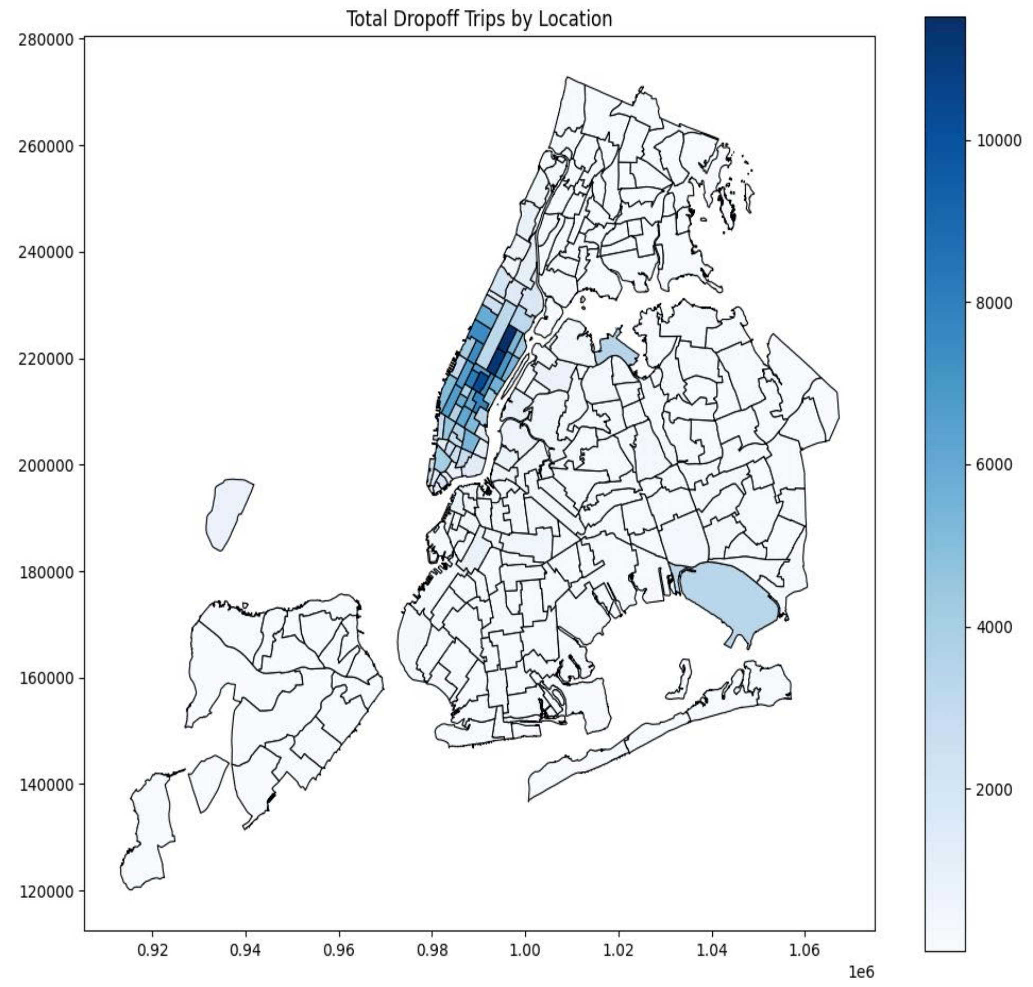
Visualizing Pickup Trips by Location

Step 1: I created a choropleth map to visualize the total pickup trips by location.



Visualizing Dropoff Trips by Location

Step 2: I created a choropleth map to visualize the total dropoff trips by location.



These visualizations help identify hotspots for taxi pickups and dropoffs, which can be useful for optimizing fleet management and resource allocation.

3.2 Detailed EDA: Insights and Strategies

3.2.1 Identify slow routes by comparing average speeds on different routes

Analyzing Slowest Routes at Different Times of the Day

Calculating Average Trip Duration by Route and Hour

Step 1: I grouped the data by pickup location (PULocationID), dropoff location (DOLocationID), and hour to calculate the average trip duration for each route and hour.

```
slow_routes = df.groupby(['PULocationID', 'DOLocationID',
'hour'])['trip_duration'].mean().reset_index()

slow_routes = slow_routes.sort_values(by='trip_duration', ascending=False)

print(slow_routes.head())
```

The groupby() method groups the data by pickup location, dropoff location, and hour.

The mean() method calculates the average trip duration for each group.

The result is sorted in descending order to identify the slowest routes.

Calculating Trip Speed

Step 2: I calculated the trip speed in miles per hour (mph) for each trip.

```
df['trip_duration_hours'] = df['trip_duration'] / 3600

df['speed_mph'] = df['trip_distance'] / df['trip_duration_hours']
```

The trip_duration (in seconds) is converted to hours by dividing by 3600.

The speed is calculated by dividing the trip_distance (in miles) by the trip_duration_hours.

categorizing Time of Day

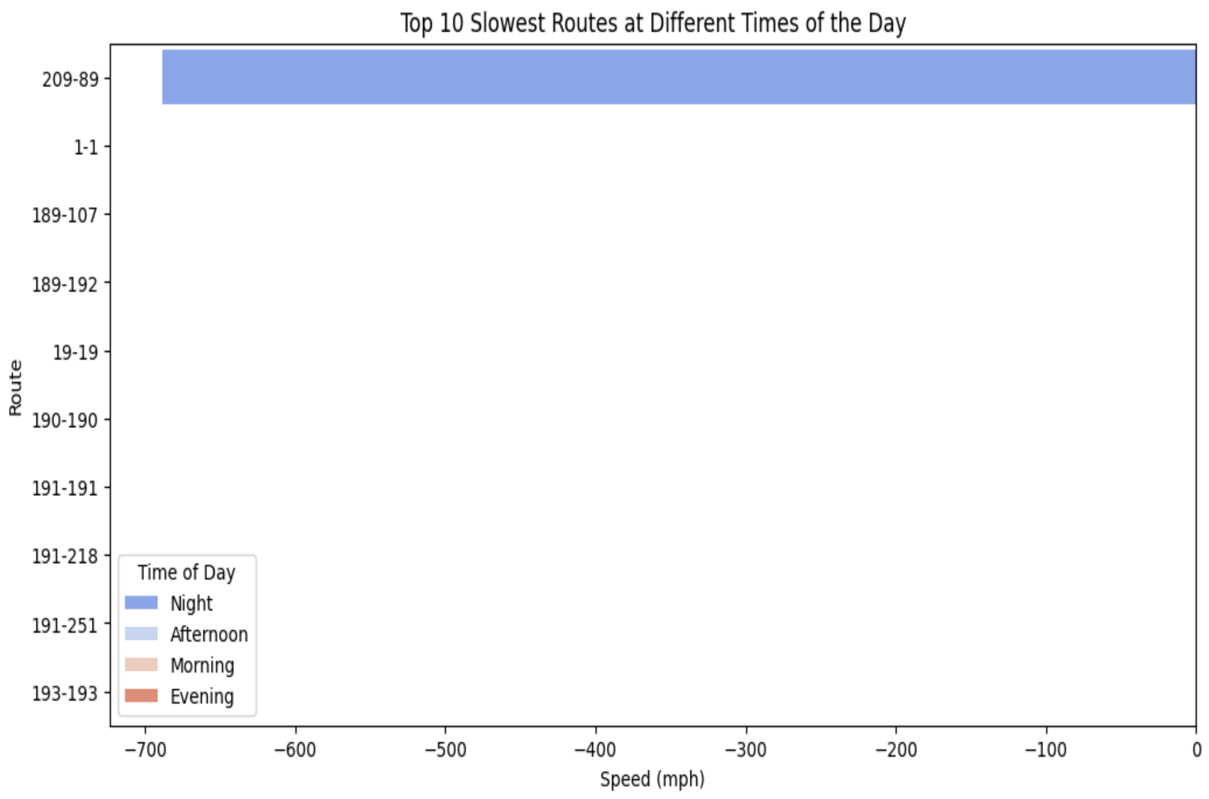
Step 3: I categorized the trips into different times of the day (Morning, Afternoon, Evening, Night) based on the pickup hour.

Identifying Slowest Routes by Time of Day

Step 4: I grouped the data by route and time of day to calculate the average speed for each route at different times of the day.

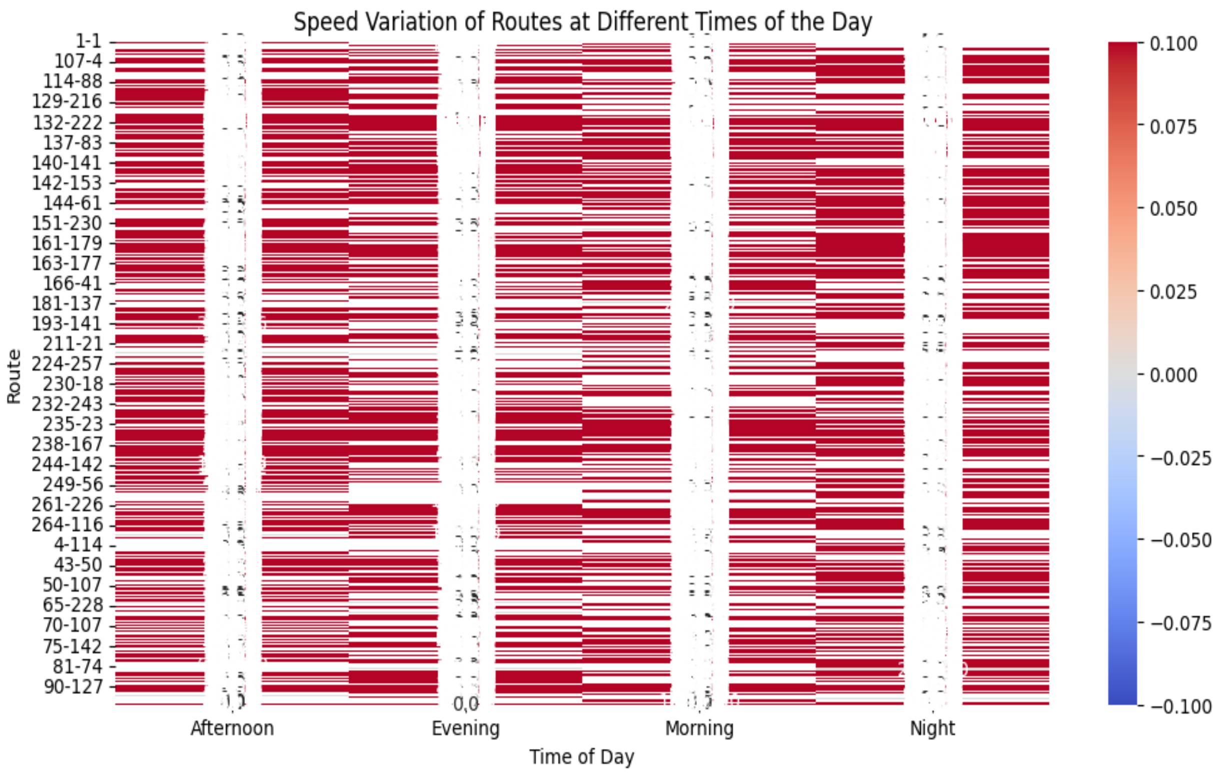
Visualizing Slowest Routes

Step 5: I created a bar plot to visualize the top 10 slowest routes at different times of the day.



Heatmap of Speed Variation

Step 6: I created a heatmap to visualize the speed variation of routes at different times of the day.



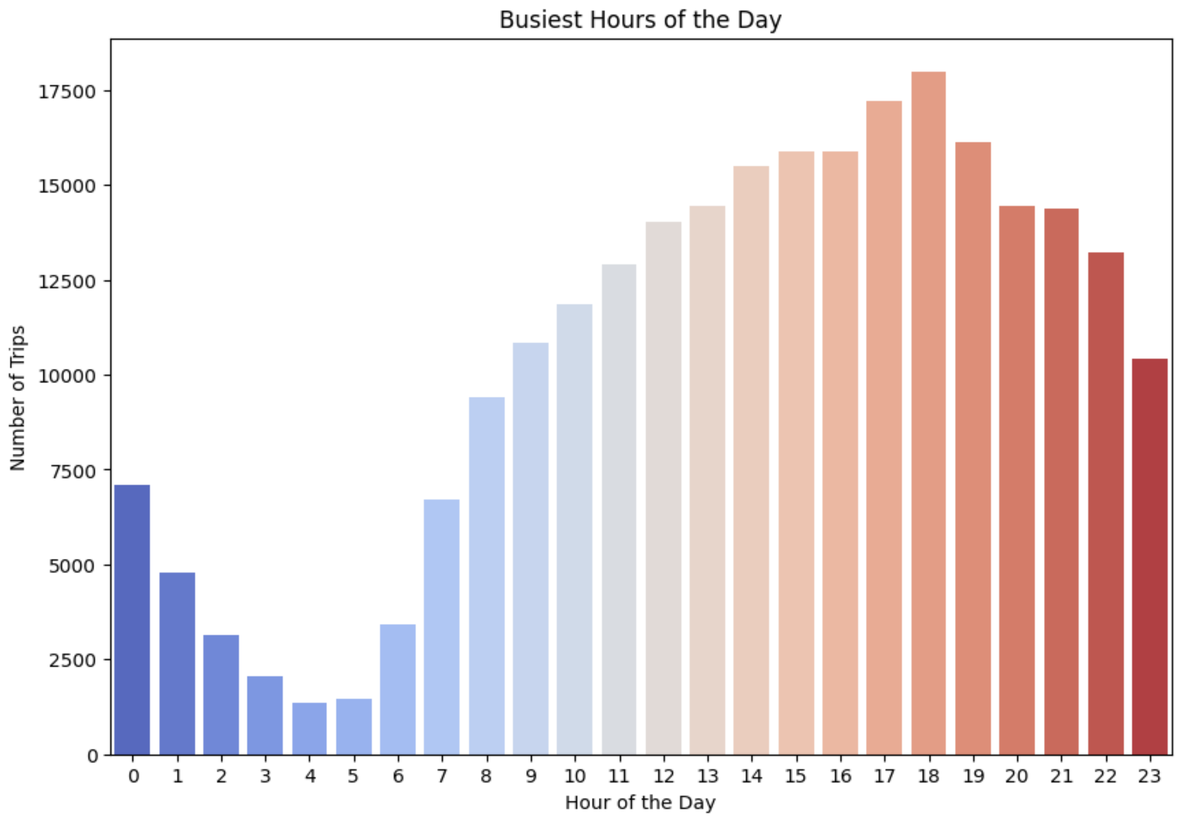
This analysis helps identify the slowest routes and times of day, which can be useful for optimizing traffic management and improving trip efficiency

3.2.2 Calculate the hourly number of trips and identify the busy hours

I extracted the hour, day, and month from the tpep_pickup_datetime column for further analysis.

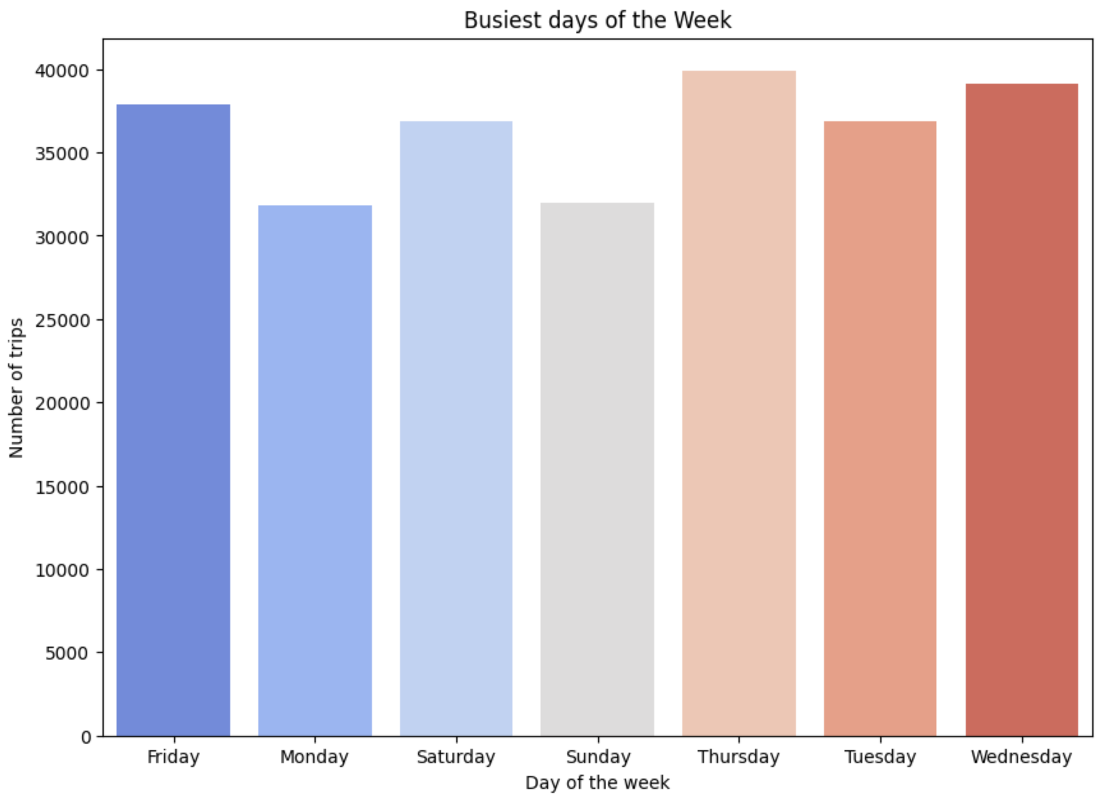
Visualizing Trips by Hour

Step 2: I grouped the data by hour and visualized the number of trips to identify the busiest hour.



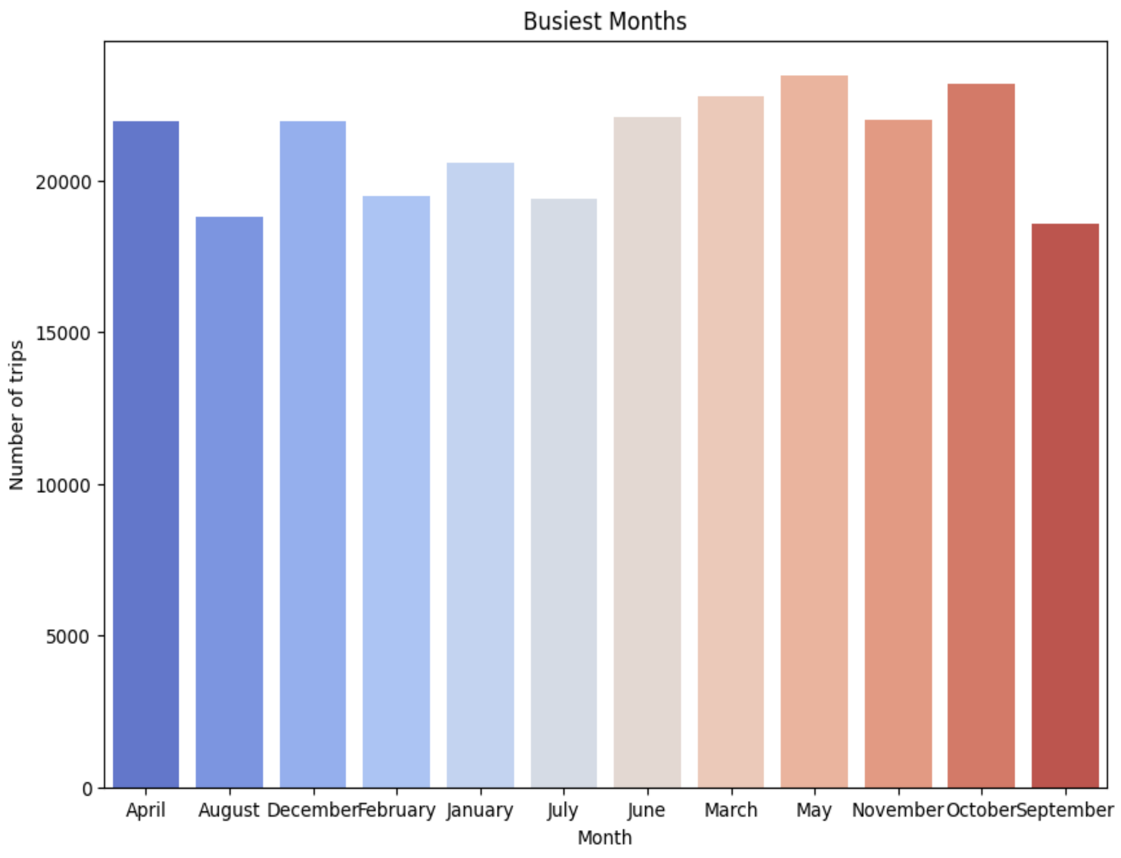
Visualizing Trips by Day

Step 3: I grouped the data by day and visualized the number of trips to identify the busiest day of the week.



Visualizing Trips by Month

Step 4: I grouped the data by month and visualized the number of trips to identify the busiest month.



3.2.3 Scale up the number of trips from above to find the actual number of trips

Scaling Up Based on Sampling Fraction

Step 1: I scaled up the number of trips to estimate the actual trip counts based on the sampling fraction.

This step helps estimate the true trip counts based on the sampled data, which is useful for understanding the actual demand and planning resources accordingly.

3.2.4 Compare hourly traffic on weekdays and weekends

I categorized the days as either "Weekday" or "Weekend" based on the day of the week.

```
df['pickup_dayofweek'] = df['tpep_pickup_datetime'].dt.dayofweek
```

```
df['day_type'] = df['pickup_dayofweek'].apply(lambda x: 'Weekend' if x >= 5 else 'Weekday')
```

The apply(lambda x: 'Weekend' if x >= 5 else 'Weekday') categorizes days as "Weekend" (Saturday and Sunday) or "Weekday" (Monday to Friday).

Grouping Data by Hour and Day Type

Step 2: I grouped the data by hour and day type to calculate the number of trips for each combination

```
hourly_traffic = df.groupby(['hour',
'day_type']).size().reset_index(name='trip_count')
```

```
print(hourly_traffic.head())
```

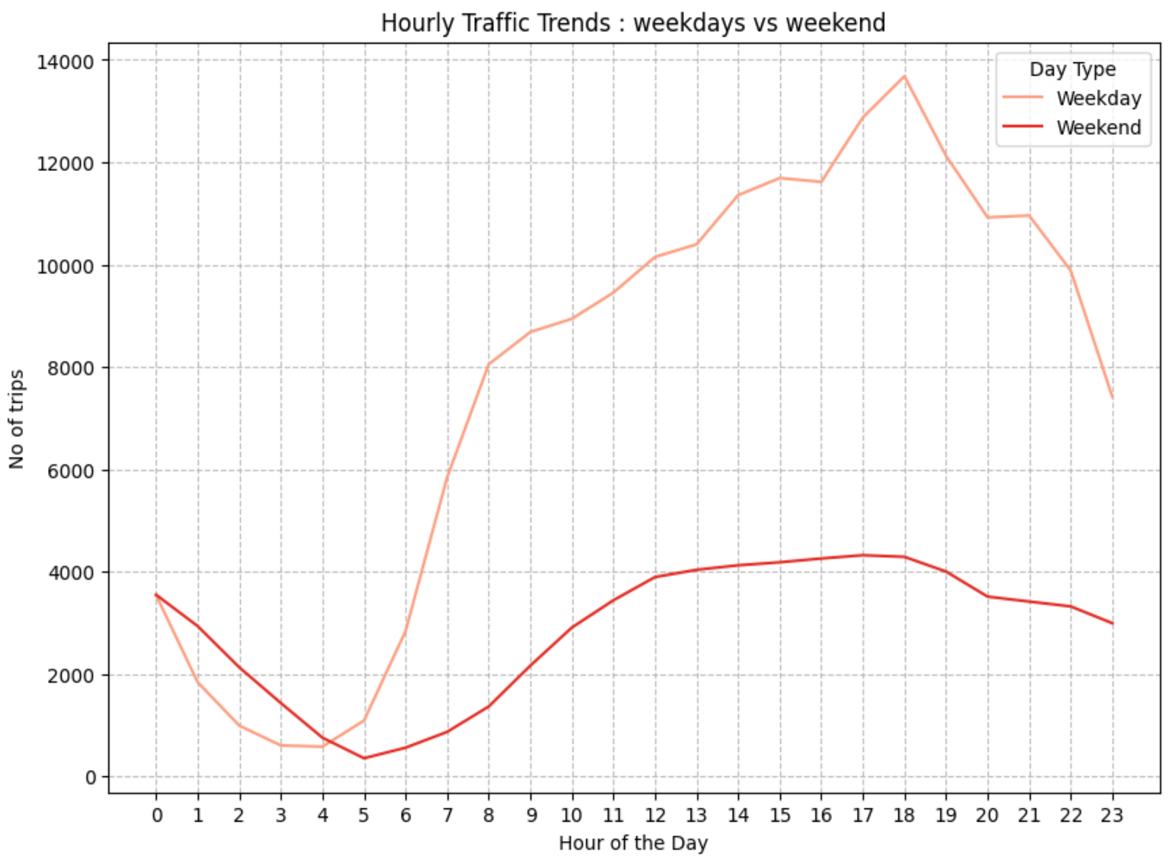
The groupby(['hour', 'day_type']).size() method counts the number of trips for each hour and day type (Weekday or Weekend).

The result is stored in a dataframe (hourly_traffic) with columns: hour, day_type, and trip_count.

The head() method displays the first few rows of the grouped data.

Visualizing Hourly Traffic Trends

Step 3: I created a line plot to compare hourly traffic trends on weekdays vs. weekends.



This analysis helps identify differences in traffic patterns between weekdays and weekends, which can be useful for optimizing operations and resource allocation.

3.2.5 Identify the top 10 zones with high hourly pickups and drops

Analyzing Hourly Trends for Top Pickup Zones

Step 3: I analyzed the hourly trends for the top 10 pickup zones.

```
pickups_hourly_trends =
df[df['PULocationID'].isin(top10_pickups_zone)].groupby(['PULocationID',
'hour']).size().reset_index(name='trip_count')
```

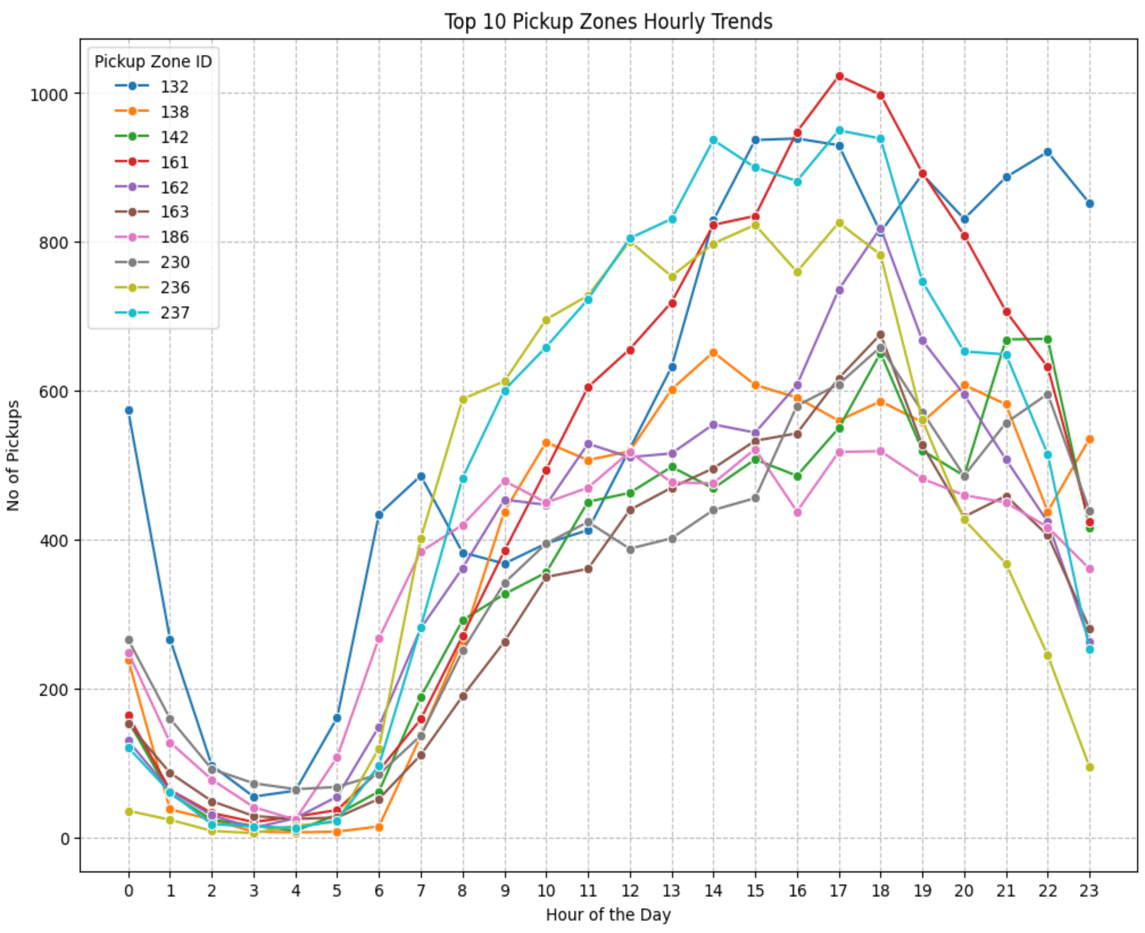
The isin(top10_pickups_zone) filters the data to include only trips from the top 10 pickup zones.

The groupby(['PULocationID', 'hour']).size() method counts the number of pickups for each zone and hour.

The result is stored in a dataframe (pickups_hourly_trends) with columns: PULocationID, hour, and trip_count.

Visualizing Hourly Trends for Top Pickup Zones

Step 4: I created a line plot to visualize the hourly trends for the top 10 pickup zones



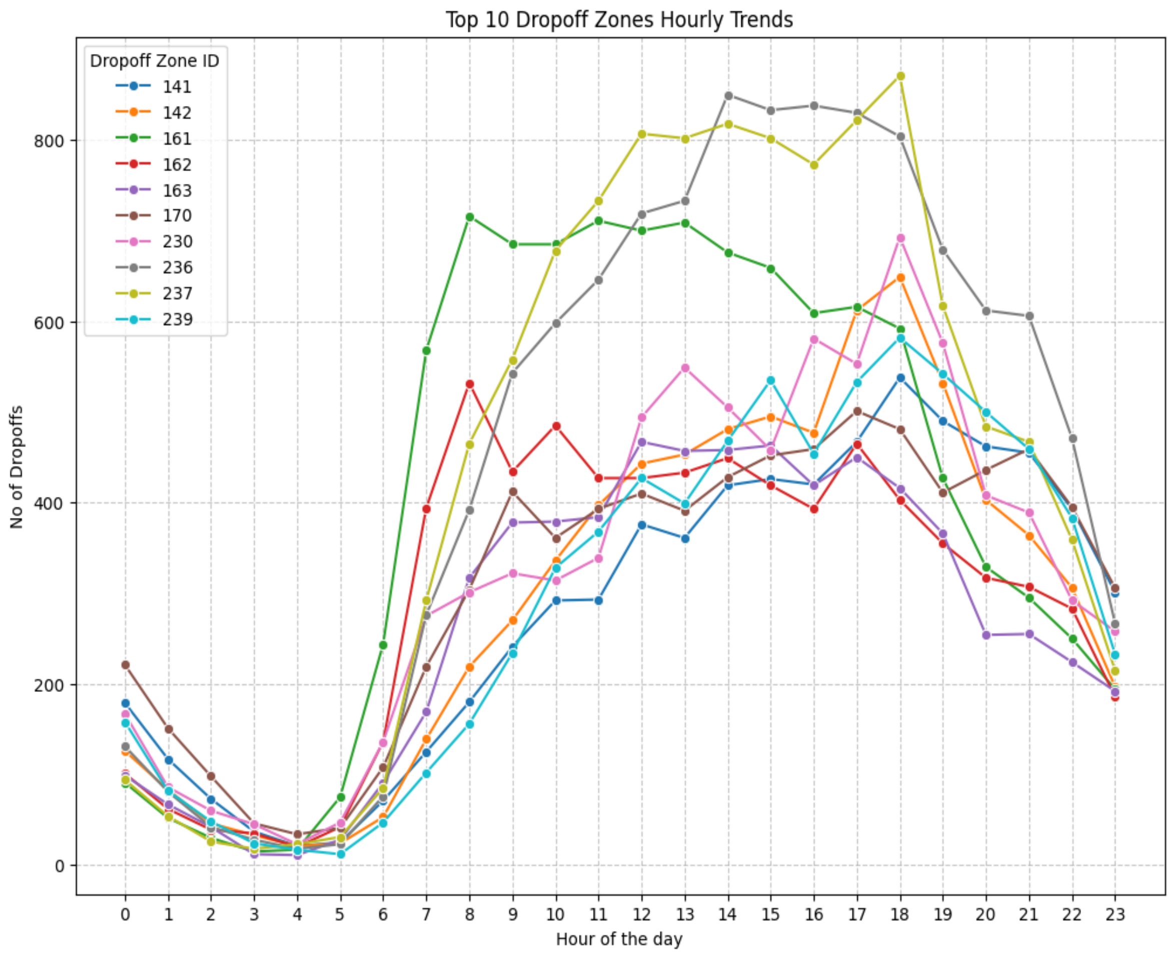
Analyzing Hourly Trends for Top Dropoff Zones

Step 5: I analyzed the hourly trends for the top 10 dropoff zones.

```
dropoff_hourly_trends =
df[df['DOLocationID'].isin(top10_dropoff_zone)].groupby(['DOLocationID',
'hour']).size().reset_index(name='trip_count')
```

Visualizing Hourly Trends for Top Dropoff Zones

Step 6: I created a line plot to visualize the hourly trends for the top 10 dropoff zones



This analysis helps identify the busiest pickup and dropoff zones and their hourly trends, which can be useful for optimizing fleet management and resource allocation.

3.2.6 Find the ratio of pickups and dropoffs in each zone

Analyzing Pickup/Dropoff Ratios

Counting Pickups and Dropoffs by Location

Step 1: I grouped the data by pickup and dropoff locations to count the number of pickups and dropoffs for each location.

Renaming Columns for Consistency

Step 2: I renamed the columns in both dataframes to ensure consistency when merging.

Merging Pickup and Dropoff Counts

Step 3: I merged the pickup and dropoff counts into a single dataframe.

Calculating Pickup/Dropoff Ratios

Step 4: I calculated the pickup/dropoff ratio for each location.

Identifying Top 10 and Bottom 10 Ratios

Step 5: I identified the top 10 and bottom 10 locations based on their pickup/dropoff ratios.

This analysis helps identify locations with imbalanced pickup and dropoff activity, which can be useful for optimizing fleet allocation and addressing demand-supply gaps.

3.2.7 Identify the top zones with high traffic during night hours

Analyzing Nighttime Pickup and Dropoff Zones

Filtering Nighttime Trips

Step 1: I filtered the data to include only trips that occurred during nighttime hours (11 PM to 5 AM).

```
night_pickups = df[(df['tpep_pickup_datetime'].dt.hour >= 23) |
(df['tpep_pickup_datetime'].dt.hour < 5)]
```

```
night_dropoffs = df[(df['tpep_dropoff_datetime'].dt.hour >= 23) |
(df['tpep_dropoff_datetime'].dt.hour < 5)]
```

Identifying Top 10 Nighttime Pickup Zones

Step 2: I grouped the nighttime pickup data by location and identified the top 10 pickup zones.

Identifying Top 10 Nighttime Dropoff Zones

Step 3: I grouped the nighttime dropoff data by location and identified the top 10 dropoff zones.

This analysis helps identify the busiest dropoff zones during nighttime hours, which can be useful for optimizing nighttime operations and resource allocation

3.2.8 Find the revenue share for nighttime and daytime hours

Analyzing Revenue Share by Time Period (Nighttime vs. Daytime)

Step 1: I categorized the trips into "Night" (11 PM to 5 AM) and "Day" (5 AM to 11 PM) based on the pickup hour.

```
df['time_period'] = df['hour'].apply(lambda x: 'Night' if (x >= 23 or x < 5) else 'Day')
```

the apply(lambda x: 'Night' if (x >= 23 or x < 5) else 'Day') method categorizes the trips into "Night" or "Day" based on the pickup hour.

Calculating Revenue Share by Time Period

Step 2: I grouped the data by time period and calculated the total revenue for each period.

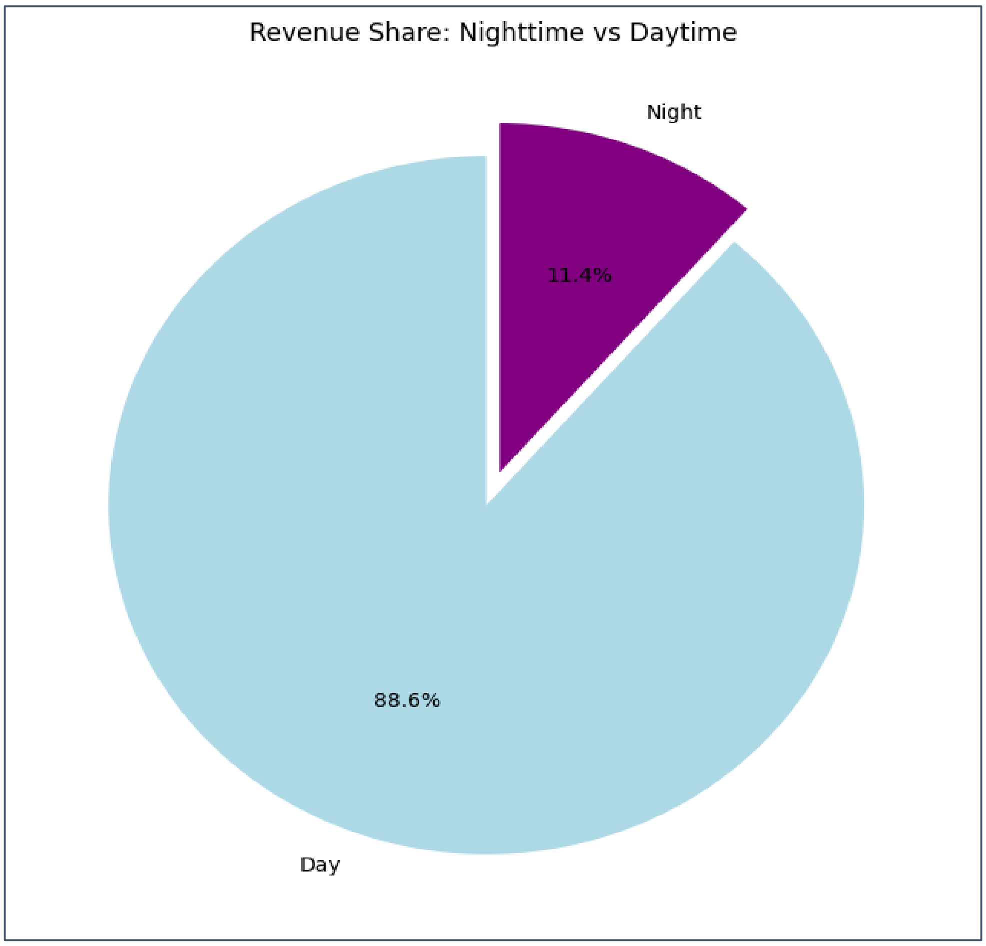
```
revenue_share = df.groupby('time_period')['total_amount'].sum().reset_index()

revenue_share['percentage'] = (revenue_share['total_amount'] /
revenue_share['total_amount'].sum()) * 100

(revenue_share)
```

The groupby('time_period')['total_amount'].sum() method calculates the total revenue for each time period (Night and Day).

percentage column is calculated by dividing the revenue for each period by the total revenue and multiplying by 100.

Visualizing Revenue Share

I created a pie chart to visualize the revenue share for nighttime and daytime hours.



This analysis helps understand the revenue distribution between nighttime and daytime hours, which can be useful for strategic planning and resource allocation

3.2.9 For the different passenger counts, find the average fare per mile per passenger

Analyzing Fare per Mile per Passenger

Step 1: I filtered the data to include only trips with a positive number of passengers and a positive trip distance

```
filtered_df = df[(df['passenger_count'] > 0) & (df['trip_distance'] > 0)]
```

Calculating Fare per Mile per Passenger

Step 2: I calculated the fare per mile per passenger for each trip.

```
filtered_df['fare_per_mile_per_passenger'] = filtered_df['fare_amount'] / (filtered_df['trip_distance'] * filtered_df['passenger_count'])
```

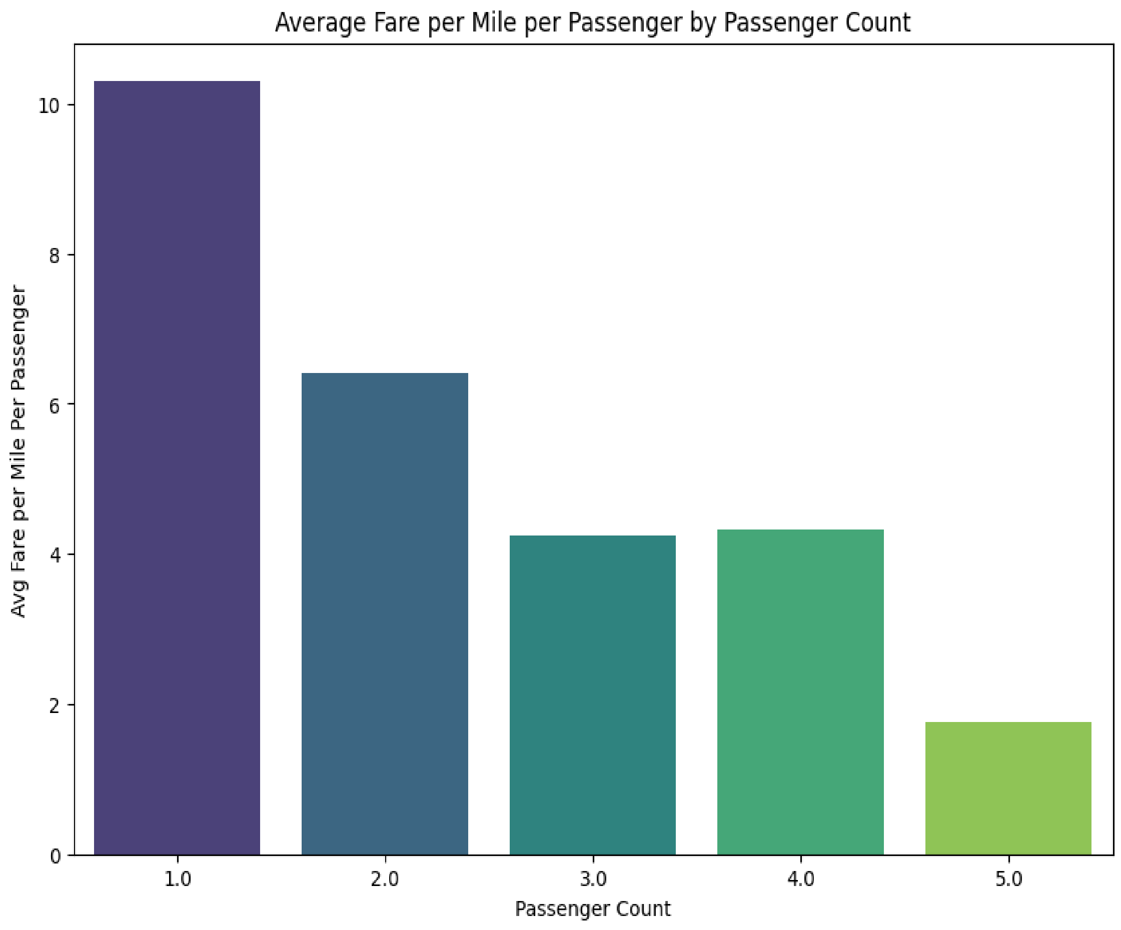
Calculating Average Fare per Mile per Passenger

Step 3: I grouped the data by passenger count and calculated the average fare per mile per passenger.

```
avg_fare =
filtered_df.groupby('passenger_count')['fare_per_mile_per_passenger'].mean().rese
t_index()

print(avg_fare)
```

Visualizing Average Fare per Mile per Passenger

Step 4: I created a bar plot to visualize the average fare per mile per passenger by passenger count.



This analysis helps understand how the fare per mile per passenger varies with the number of passengers, which can be useful for pricing strategies and customer communication.

3.2.10 Find the average fare per mile by hours of the day and by days of the week

Analyzing Average Fare per Mile by Day of the Week and Hour

Step 1: I ensured the tpep_pickup_datetime column is in datetime format and filtered out trips with invalid distances.

```
df['tpep_pickup_datetime'] =
pd.to_datetime(df['tpep_pickup_datetime'],errors='coerce')

df = df[df['trip_distance'] > 0].copy()
```

Step 2: I calculated the fare per mile for each trip.

```
df['fare_per_mile'] = df['fare_amount'] / df['trip_distance']
```

Step 3: I extracted the day of the week and hour from the pickup datetime.

Calculating Average Fare per Mile by Day of the Week

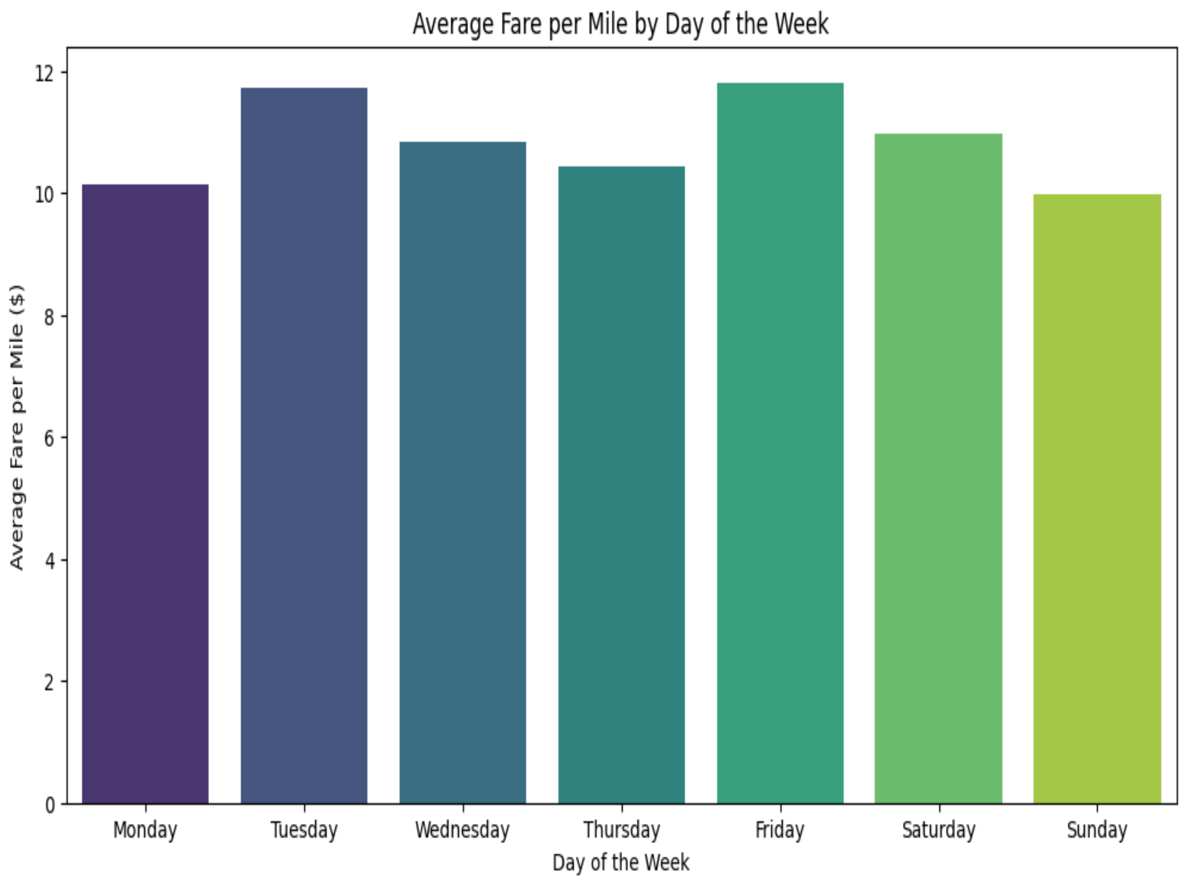
Step 4: I grouped the data by day of the week and calculated the average fare per mile

Step 5: I ordered the days of the week in their natural order.

```
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'Sunday']

avg_fare_day['day_of_week'] = pd.Categorical(avg_fare_day['day_of_week'],
categories=day_order, ordered=True)

avg_fare_day = avg_fare_day.sort_values('day_of_week')
```

Visualizing Average Fare per Mile by Day of the Week

Step 6: I created a bar plot to visualize the average fare per mile by day of the week.



This analysis helps understand how the fare per mile varies by day of the week and hour, which can be useful for pricing strategies and operational planning.

3.2.11 Analyse the average fare per mile for the different vendors

Step 1: I filtered the data to include only trips with a valid distance and calculated the fare per mile.

```
filtered_dff = df[df['trip_distance'] > 0]

filtered_dff['fare_per_mile'] = filtered_dff['fare_amount'] / filtered_dff['trip_distance']
```

Calculating Average Fare per Mile by Vendor

Step 2: I grouped the data by vendor and calculated the average fare per mile.

```
avg_fare_by_vendor = filtered_dff.groupby('VendorID')['fare_per_mile'].mean().reset_index()

print(avg_fare_by_vendor)
```

This analysis helps understand how the fare per mile varies by vendor, which can be useful for pricing strategies and vendor performance evaluation.

3.2.12 Compare the fare rates of different vendors in a distance-tiered fashion

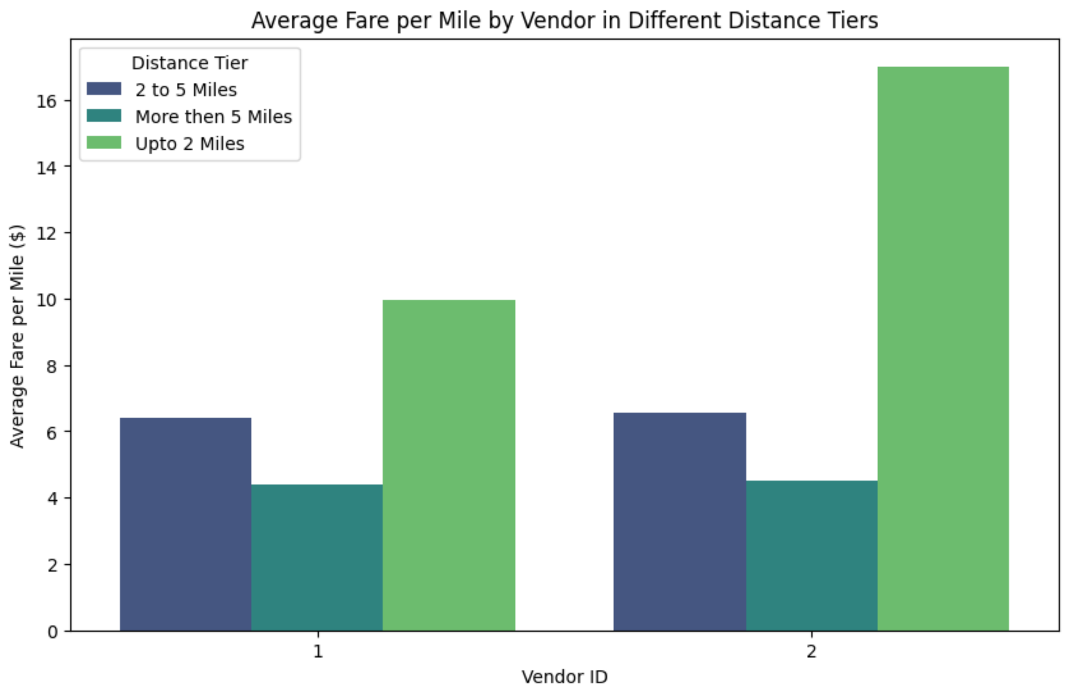
I categorized trips into distance tiers based on the trip distance.

```
def distance_tier(dist):
if dist <= 2:
 return 'Upto 2 Miles'
elif dist <= 5:
 return '2 to 5 Miles'
else:
return 'More than 5 Miles'
filtered_dff['distance_tier'] = filtered_dff['trip_distance'].apply(distance_tier)
```

Step 4: I grouped the data by vendor and distance tier and calculated the average fare per mile.

Step 5: I created a pivot table to organize the data for better visualization.

Step 6: I created a bar plot to visualize the average fare per mile by vendor and distance tier.



This analysis helps understand how the fare per mile varies by vendor and distance tier, which can be useful for pricing strategies and vendor performance evaluation

3.2.13 Analyse the tip percentages

Step 1: I filtered the data to include only trips with a valid fare amount and trip distance, and calculated the tip percentage.

Step 2: I further filtered the data to include only trips with a tip percentage between 0% and 100%.

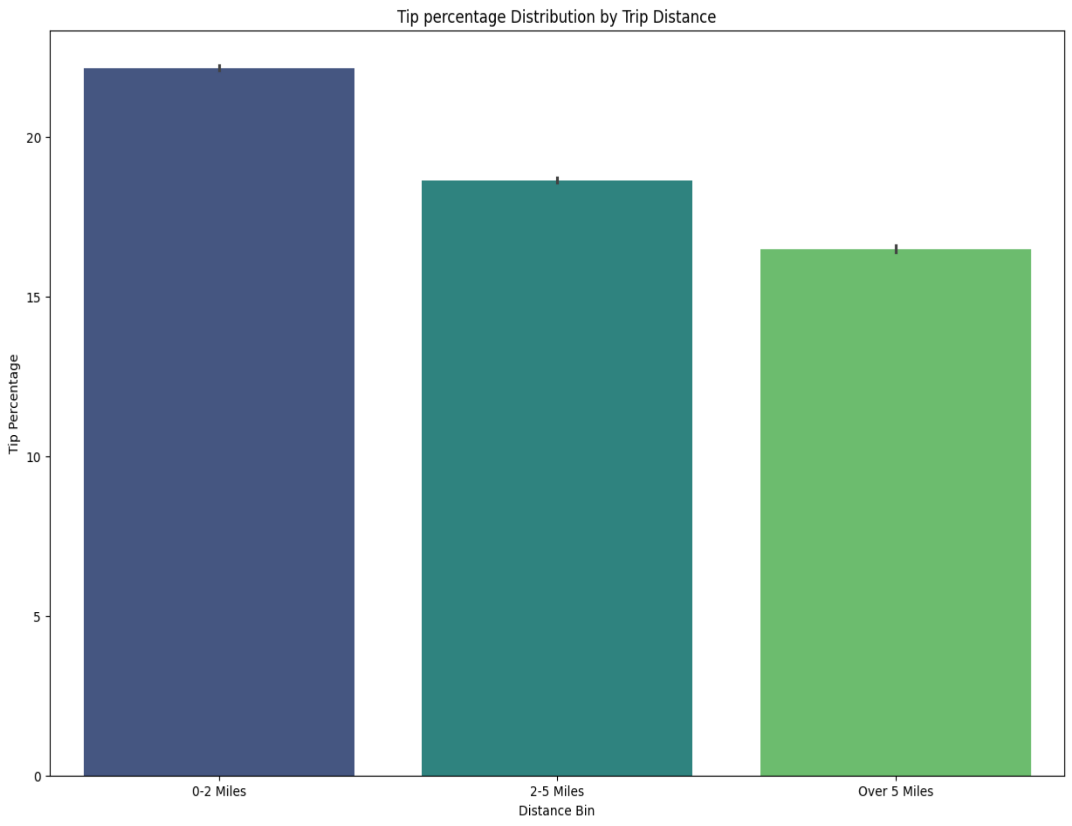
```
df_clean = tip_filtered[(tip_filtered['tip_percentage'] >= 0) &
(tip_filtered['tip_percentage'] <= 100)]
```

Categorizing Trips by Distance

Step 3: I categorized trips into distance bins based on the trip distance.

Visualizing Tip Percentages by Distance Bin

Step 4: I created a bar plot to visualize the average tip percentage by distance bin.



This analysis helps understand how tipping behavior varies with trip distance, which can be useful for driver incentives and customer engagement strategies.

3.2.13 Analyse the trends in passenger count varies by day and hour

Step 1: I extracted the pickup hour and day of the week from the tpep_pickup_datetime column.

Step 2: I mapped the day numbers to day names for better readability.

day_mapping = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday', 4: 'Friday', 5: 'Saturday', 6: 'Sunday'}

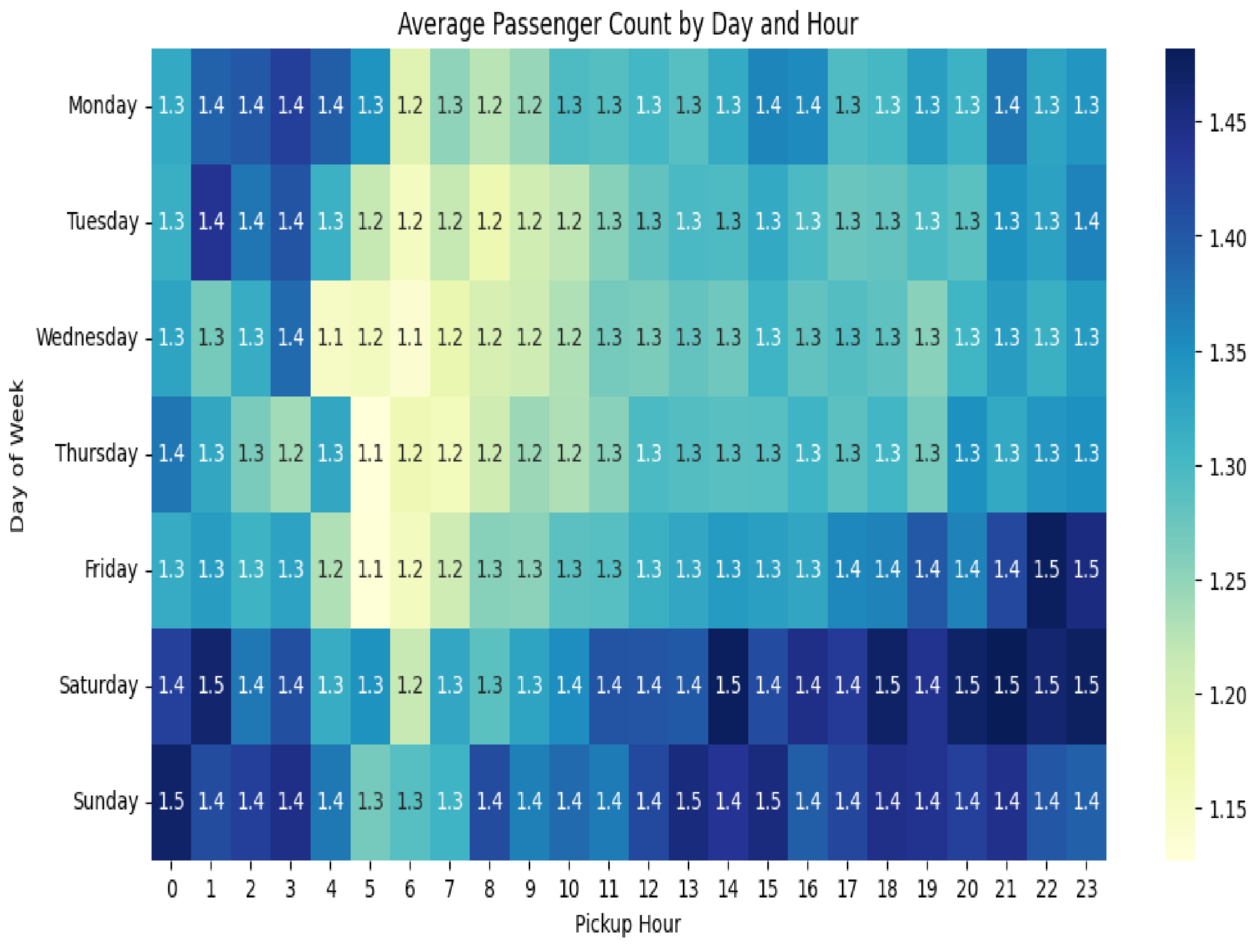
df['day_name'] = df['day_of_week'].map(day_mapping)

day_mapping = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday', 4: 'Friday', 5: 'Saturday', 6: 'Sunday'}

df['day_name'] = df['day_of_week'].map(day_mapping)

Step 5: I reordered the days of the week in their natural order.

Visualizing Passenger Count by Hour and Day



3.2.14 Analyse the variation of passenger counts across zones

Calculating Average Passenger Count by Pickup Zone

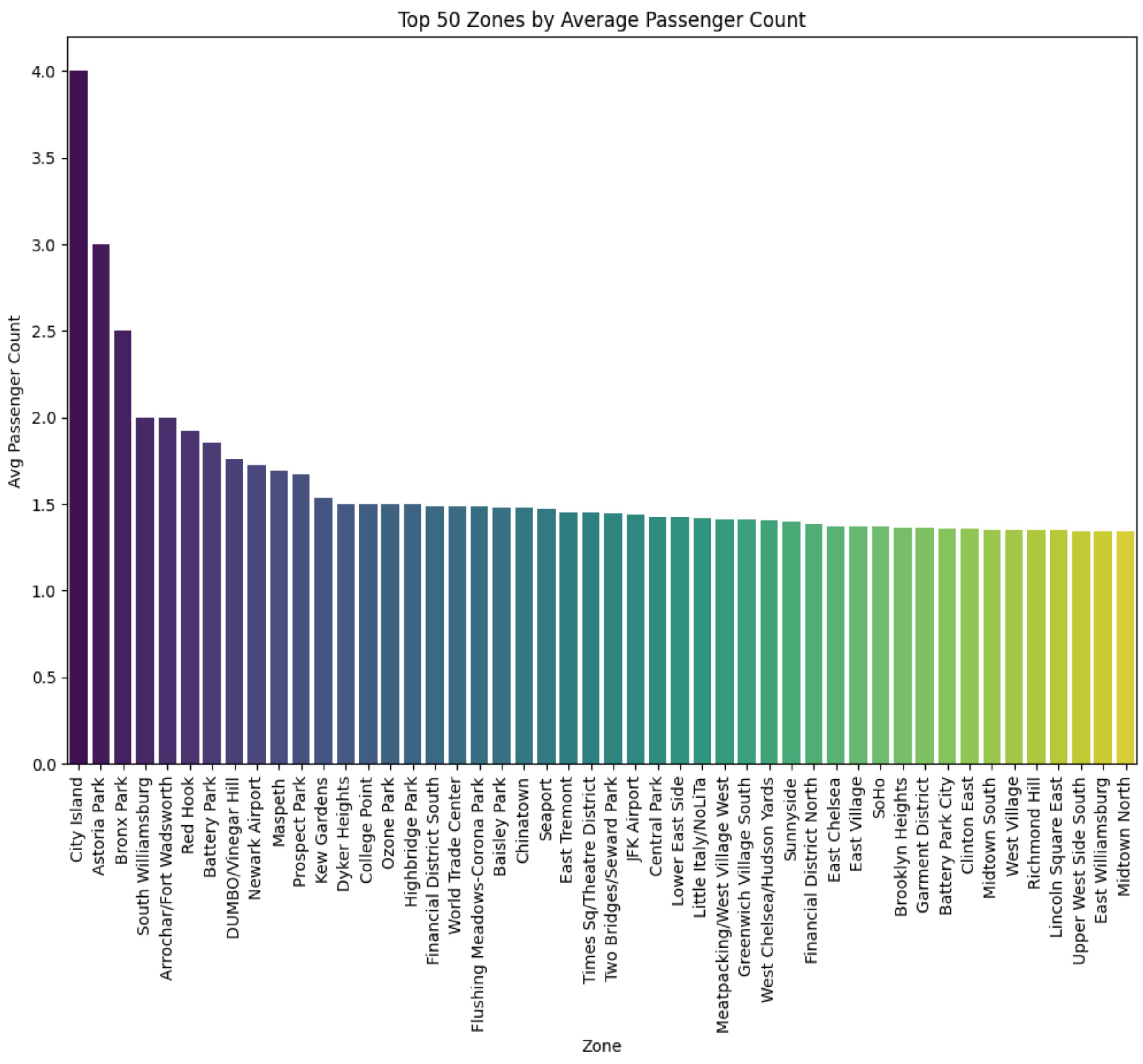
Step 1: I grouped the data by pickup location (PULocationID) and calculated the average passenger count for each zone.

Merging with Zone Information

Step 2: I merged the average passenger count data with the taxi zone data to include zone names.

Visualizing Average Passenger Count by Zone

Step 3: I created a bar plot to visualize the average passenger count by zone



3.2.15 Analyse the pickup/dropoff zones or times when extra charges are applied more frequently

Analyzing Surcharge Application Frequency

Identifying Surcharge Columns

Step 1: I identified the columns related to surcharges in the dataset.

Calculating Surcharge Application Frequency

Step 2: I calculated the percentage of trips where each surcharge is applied.

This analysis helps understand how frequently each surcharge is applied, which can be useful for financial planning and customer communication.

4. Conclusion

Our analysis of the taxi trip data gave us some pretty interesting insights that can help us meet customer demand better and optimize how we manage supply. Here's what we found:

Time of Day:

1. Peak Hours:

Weekdays: The busiest times for taxi pickups are during the morning rush (around 7–10 AM) and the evening rush (around 5–8 PM). No surprises there—everyone's trying to get to work or head home.

Nighttime: Even though trips drop off during late-night hours (11 PM to 5 AM), some areas still stay pretty busy, especially near bars and clubs.

2.Off-Peak Hours:

Midday and early afternoons are slower, which could be a good time for drivers to take breaks or for us to run promotions like discounts or ride-sharing deals to fill up cars

Day of the Week:

1.Weekdays vs. Weekends:

Weekdays have clear peaks during commuting hours, while weekends are more spread out, with a lot of late-night demand. This means we should probably focus on having more taxis available during weekday rush hours and keep a steady supply on weekends for the nightlife crowd.

Location-Based Insights:

1.High-Demand Zones:

Some areas have way more pickups than dropoffs, and vice versa. For example, residential areas see a lot of pickups in the morning (people heading to work), while commercial and entertainment zones get more dropoffs in the evening.

2. Zone-Specific Strategy:

We should probably move more taxis to residential areas in the morning and shift them to commercial areas in the evening. This could cut down on wait times and make things run smoother. Also, we could use pickup/dropoff data to create targeted promotions for drivers

Fare Efficiency and Tip Behavior:

1.Fare per Mile:

The fare per mile changes depending on the time of day and traffic. Unsurprisingly, peak hours are less efficient because of all the congestion

2.Tip Percentages:

Tips tend to be higher for longer trips and during off-peak hours. Maybe people are happier with the service when there's less traffic, or it could just be a different type of customer.

4.1. Recommendations to optimize routing and dispatching based on demand patterns and operational inefficiencies.

<u>Recommendations to Optimizing Routing:</u>
Dynamic Dispatching During Peak Hours:
During weekday rush hours , we should prioritize dispatching taxis to high-demand residential areas in the morning and commercial/entertainment zones in the evening. This will help reduce wait times and improve customer satisfaction.

Optimize Routing for Fare Efficiency:
During peak hours, traffic congestion can reduce fare efficiency. We should explore routing algorithms that avoid heavily congested areas, even if it means slightly longer routes. This could improve driver earnings and reduce trip times.

Leverage Data for Late-Night Demand:
Late-night demand is concentrated in specific zones (e.g., nightlife areas). We should ensure adequate taxi availability in these zones during weekends and holidays, possibly by offering surge pricing or driver incentives

Improve Driver Communication:
Provide drivers with real-time updates on high-demand zones and traffic conditions through their apps. This will help them make better decisions about where to go and when, improving overall efficiency.

Encourage Ride-Sharing During Off-Peak Hours:
To boost occupancy during slower periods (midday and early afternoons), we could promote ride-sharing options or offer discounts for shared rides. This would help drivers earn more while providing cost-effective options for riders

4.2 Suggestions on strategically positioning cabs across different zones to make best use of insights uncovered by analysing trip trends across time, days and months.

<u>Suggestions for strategically positioning cabs:</u>

Morning Rush :

Focus on residential areas where pickups are highest. People are heading to work, so positioning cabs in neighborhoods with high commuter activity will reduce wait times and improve service.

Evening Rush:

Shift cabs to commercial and business districts in the evening. Workers are heading home, and there’s often a surge in demand around office buildings and transit hubs

Weekends and Late Nights:

Position cabs near entertainment zones, bars, and restaurants, especially on Friday and Saturday nights. These areas see consistent demand during weekends and holidays

4.3 Propose data-driven adjustments to the pricing strategy to maximize revenue while maintaining competitive rates with other vendors.

Data-Driven Pricing Adjustment;

Surge Pricing During Peak Hours:

Implement dynamic pricing during weekday rush hours and weekend late nights. Higher fares during these times can maximize revenue while balancing supply and demand.

Surge Pricing During Peak Hours:

Implement dynamic pricing during weekday rush hours and weekend late nights. Higher fares during these times can maximize revenue while balancing supply and demand.

Incentivize Longer Trips:

Since longer trips tend to have higher tip percentages, consider offering small discounts or perks for riders taking longer trips. This could encourage more of these high-value rides

Incentivize Longer Trips:

Since longer trips tend to have higher tip percentages, consider offering small discounts or perks for riders taking longer trips. This could encourage more of these high-value rides

Competitive Pricing Analysis:

Regularly compare fares with competitors to ensure our rates remain competitive. If competitors lower their prices in certain zones or times, we should adjust accordingly to stay attractive to rider

Loyalty Programs:

Introduce loyalty programs or discounts for frequent riders. This can help retain customers and encourage them to choose our service over competitor