Report: NYC Taxi Operations

Include your visualisations, 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

* The warnings package was downloaded and essential libraries like NumPy, Pandas, Matplotlib, and Seaborn were imported to support the EDA process.
* The pyarrow package was installed to enable the loading of Parquet files, as the provided data was in Parquet format.
* A sample Parquet file was loaded initially to verify that the setup was working correctly. Basic checks such as head(), info(), and tail() were performed to validate the data.
* Upon successful validation, the process proceeded to load and sample data from all the available monthly files.

## Sample the Data and Combine the Files

* As outlined in the process, sampling was performed on the *tpep\_pickup\_datetime* field, taking a 5% sample from each month's data.
* All Parquet files were looped through, and new hour and date columns were created from the *tpep\_pickup\_datetime* field to facilitate structured 5% sampling.
* The sampled data from each file was combined into a single DataFrame using the *pd.concat()* method.
* Basic verification of the consolidated DataFrame was carried out using methods such as info().
* The final sampled DataFrame was exported into a Parquet file using the df.to\_parquet() method for future reference.

# 2. Data Cleaning

## 2.1 Fixing Columns

* The sampled Parquet file was imported to proceed with the EDA process.
* The df.info() method was executed to inspect the columns and their data types. Based on the initial observation, the columns appeared consistent with appropriate data types and no immediate issues were detected.

## 2.1.1 Fix the Index

* The index was reset using df.reset\_index(drop=True, inplace=True), and the columns were inspected. During this inspection, it was found that there were two columns related to airport\_fee.

***df.reset\_index(drop=True, inplace=True)***

***df.columns***

*Index(['VendorID', 'tpep\_pickup\_datetime', 'tpep\_dropoff\_datetime',*

*'passenger\_count', 'trip\_distance', 'RatecodeID', 'store\_and\_fwd\_flag',*

*'PULocationID', 'DOLocationID', 'payment\_type', 'fare\_amount', 'extra',*

*'mta\_tax', 'tip\_amount', 'tolls\_amount', 'improvement\_surcharge',*

*'total\_amount', 'congestion\_surcharge', 'airport\_fee', 'date', 'hour',*

*'Airport\_fee'],*

*dtype='object'*

* Dropped the columns date, hour, and store\_and\_fwd\_flag.
* The date and hour columns were created during the sampling process and can be added again later if needed.
* The store\_and\_fwd\_flag column was dropped as it was deemed unnecessary for the EDA process based on the data dictionary.

## 2.1.2 Combine the Two airport\_fee Columns

* Since there were two *airport\_fee* columns, they were merged into one.
* The duplicate *Airport\_fee* column was dropped after merging to avoid redundancy.

*df.airport\_fee = df.airport\_fee.fillna(df.Airport\_fee)*

***Columns after dropping Airport\_fee***

*Index(['VendorID', 'tpep\_pickup\_datetime', 'tpep\_dropoff\_datetime',*

*'passenger\_count', 'trip\_distance', 'RatecodeID', 'PULocationID',*

*'DOLocationID', 'payment\_type', 'fare\_amount', 'extra', 'mta\_tax',*

*'tip\_amount', 'tolls\_amount', 'improvement\_surcharge', 'total\_amount',*

*'congestion\_surcharge', 'airport\_fee'],*

*dtype='object')*

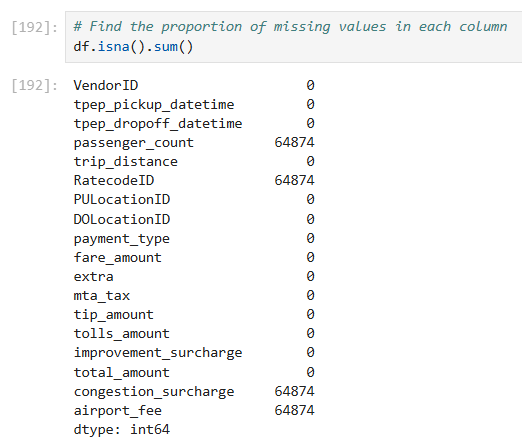
## 2.1.3 Fix Column with Negative Values

* The command countindex = df['fare\_amount'].value\_counts() counts the occurrences of each unique value in the fare\_amount column and stores it in the variable countindex.
* The command df = df[df['RatecodeID'] != 99] filters the dataframe by removing all rows where the RatecodeID is equal to 99, retaining only valid rate code entries.
* The following data cleaning steps were performed to prepare the dataset for analysis:
* Replaced missing or zero values in the extra, mta\_tax, improvement\_surcharge, total\_amount, congestion\_surcharge, and airport\_fee columns using appropriate imputation techniques such as forward-filling or filling with mean/median values.
* Dropped rows with missing values in critical fields like fare\_amount, tip\_amount, or passenger\_count to ensure all trips had valid data.
* Removed invalid entries where the RatecodeID was 99, representing unknown or unassigned rate codes.
* Verified that no missing or zero values remain in key monetary columns (fare\_amount, tip\_amount, total\_amount, etc.).
* Replaced remaining missing values with meaningful imputations or dropped the corresponding rows where necessary.

## 2.2 Handling Missing Values

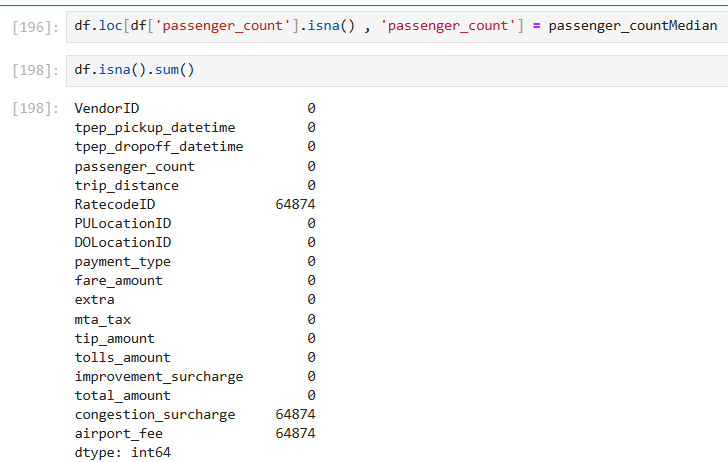
## 2.2.1 Proportion of Missing Values

* The command df.isna().sum() was executed to verify if any missing values remained in the dataset after completing the data cleaning and imputation steps.

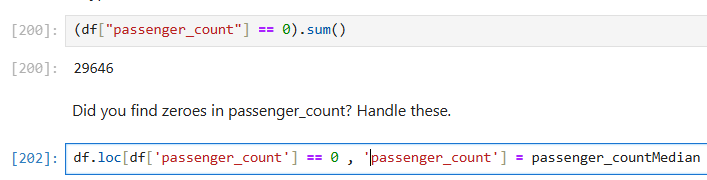


## 2.2.2 Handling Missing Values in passenger\_count

* The median value of the "passenger\_count" column was calculated using df.passenger\_count.median(), and the missing values in the "passenger\_count" column were replaced with this median value using df.loc[df['passenger\_count'].isna(), 'passenger\_count'] = passenger\_countMedian.

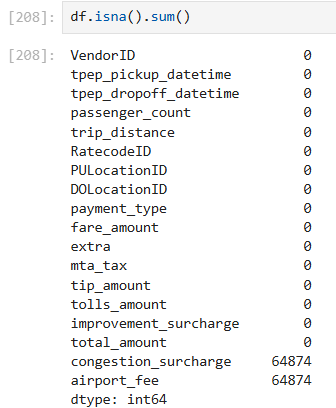


* Zeros in the "***passenger\_count***" column were identified and replaced with the median value to maintain data consistency and accuracy.



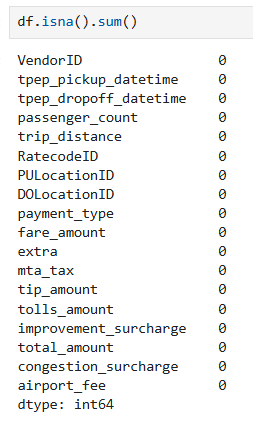
## 2.2.3 Handle Missing Values in RatecodeID

* The mode of the "RatecodeID" column was calculated using df.RatecodeID.mode()[0], and the most frequent value was stored in a variable named RatecodeIDmedian to address missing or inconsistent entries.
* Missing values in the "RatecodeID" column were replaced with the most frequent value (mode) by using df.loc[df['RatecodeID'].isna(), 'RatecodeID'] = RatecodeIDmedian.



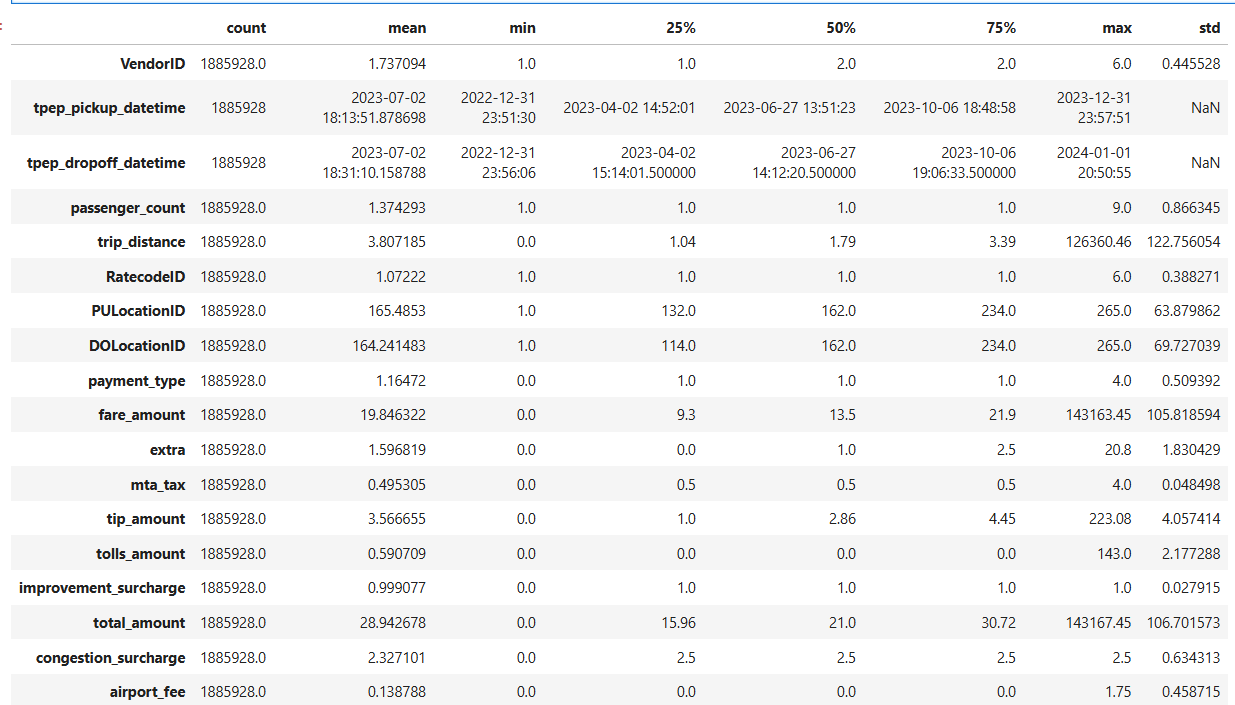
## 2.2.4 Impute NaN in congestion\_surcharge

* The missing values in the "congestion\_surcharge" column were handled by calculating its median and replacing all NaN entries with this median value.
* The NaN values in the "airport\_fee" column were identified and replaced with the median value of the "airport\_fee" to ensure consistency.
* After reviewing all columns, NaN values were replaced with the appropriate median or mode values for each respective column to ensure data completeness and consistency.



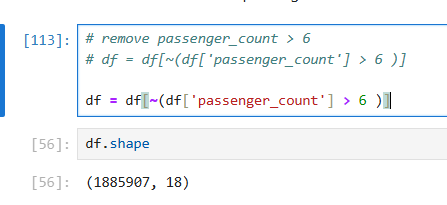
## 2.3 Handling Outliers and Standardising Values

* After handling missing values, outlier treatment was carried out across the entire dataframe to further enhance data quality and reliability.
* A statistical summary of the entire dataframe was generated using df.describe().T to understand the central tendencies, dispersion, and identify potential outliers.

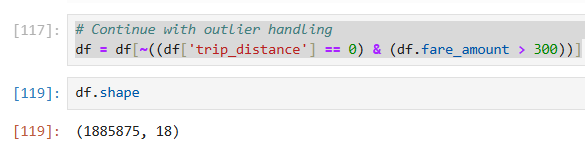


## 2.3.1 Outliers in payment\_type, trip\_distance and tip\_amount

* + Removed trips where **passenger\_count** was greater than 6, as such instances were extremely rare.



* + Dropped entries where **trip\_distance** was 0 but **fare\_amount** exceeded 300, indicating inconsistent data.



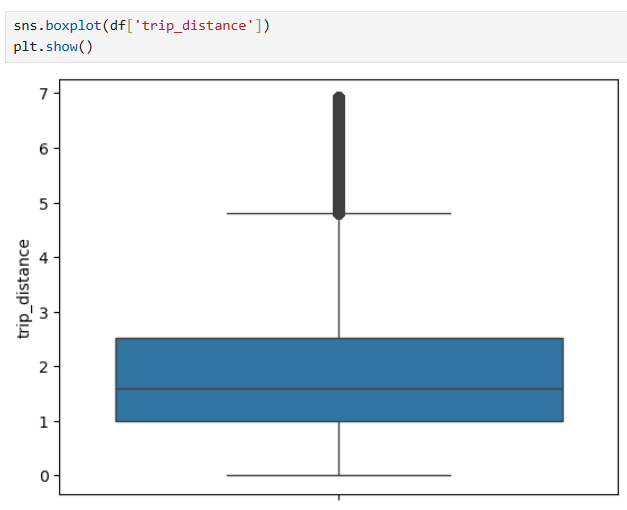
* + Eliminated trips where both **trip\_distance** and **fare\_amount** were 0, but pickup and dropoff zones were different, as distance and fare should not both be zero in such cases.



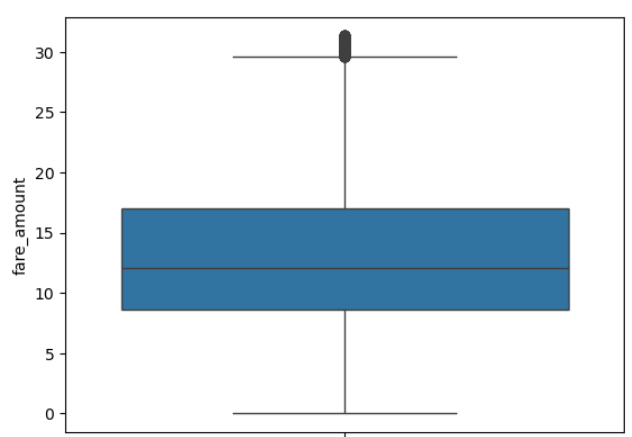
* + Removed extreme outliers where **trip\_distance** was less than 1.04 miles but **fare\_amount** was unusually high (above 200).

df = df[~((df['trip\_distance'] < 1.04) & (df['fare\_amount'] >= 200))]

Boxplot after removing the outliers

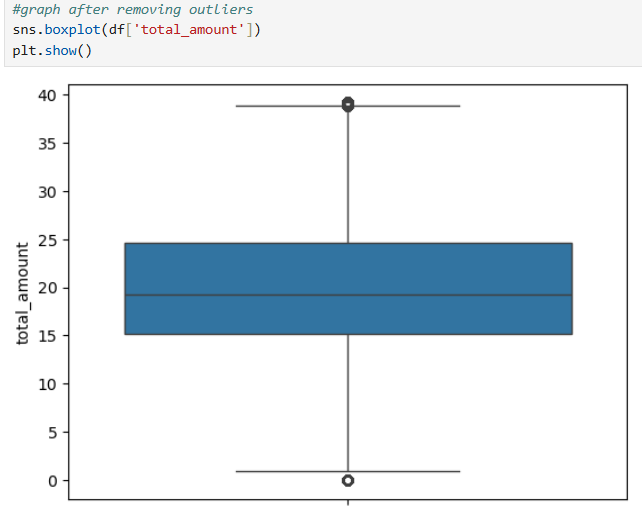


* + Cleaned out entries where **fare\_amount** was 0 while **trip\_distance** was non-zero, indicating invalid records.
  + Removed trips where **fare\_amount** exceeded 300, based on outlier analysis using the upper bound (calculated as 37.35).
  + Boxplot graph after removing the fare\_amount

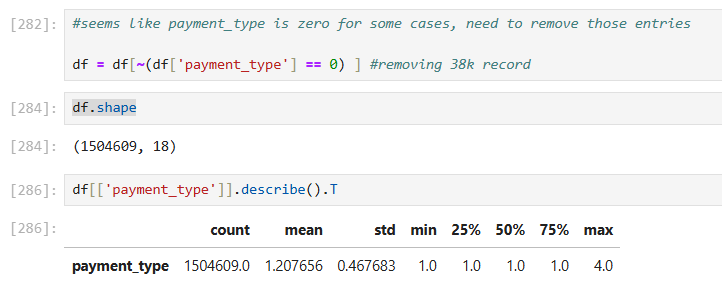


* + Similarly, removed records where **total\_amount** was greater than or equal to the upper limit to ensure consistency.

Boxplot graph after removing the outliers



* + Excluded trips where **payment\_type** was 0, as such entries indicate missing or invalid payment information.



## 3. Exploratory Data Analysis

## 3.1.General EDA: Finding Patterns and Trends

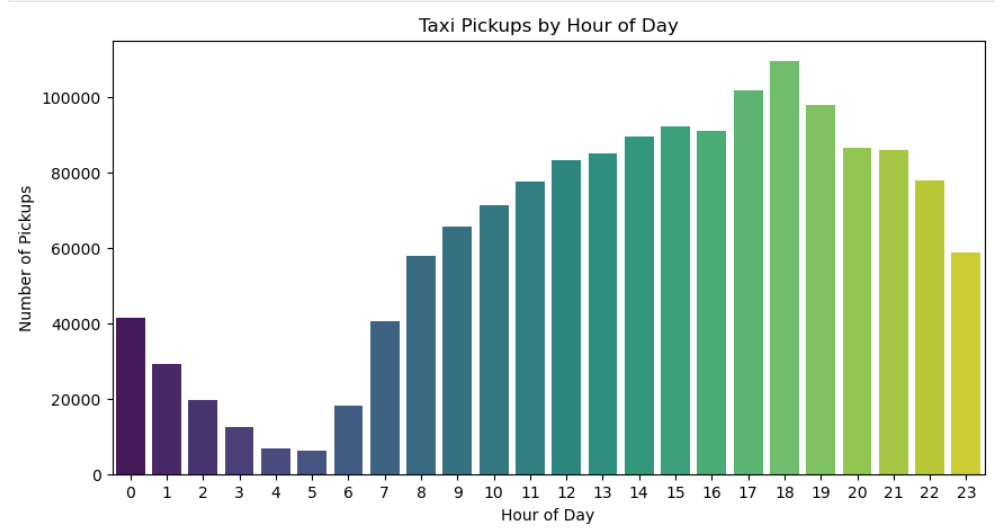
**3.1.1. Classify variables into categorical and numerical**

* Categorise the varaibles into Numerical or Categorical.
  + - VendorID - Categorical
    - tpep\_pickup\_datetime - Date
    - tpep\_dropoff\_datetime - Date
    - passenger\_count - Numerical
    - trip\_distance - Numerical
    - RatecodeID - Categorical
    - PULocationID - Taxi Zone - Categorical
    - DOLocationID - Taxi Zone - Categorical
    - payment\_type - Categorical
    - pickup\_hour - Numerical
    - trip\_duration - Numerical
* The following monetary parameters belong in the same category, is it categorical or numerical?
  + - fare\_amount - Numerical
    - extra - Numerical
    - mta\_tax - Numerical
    - tip\_amount - Numerical
    - tolls\_amount - Numerical
    - improvement\_surcharge - Numerical
    - total\_amount - Numerical
    - congestion\_surcharge - Numerical
    - airport\_fee - Numerical

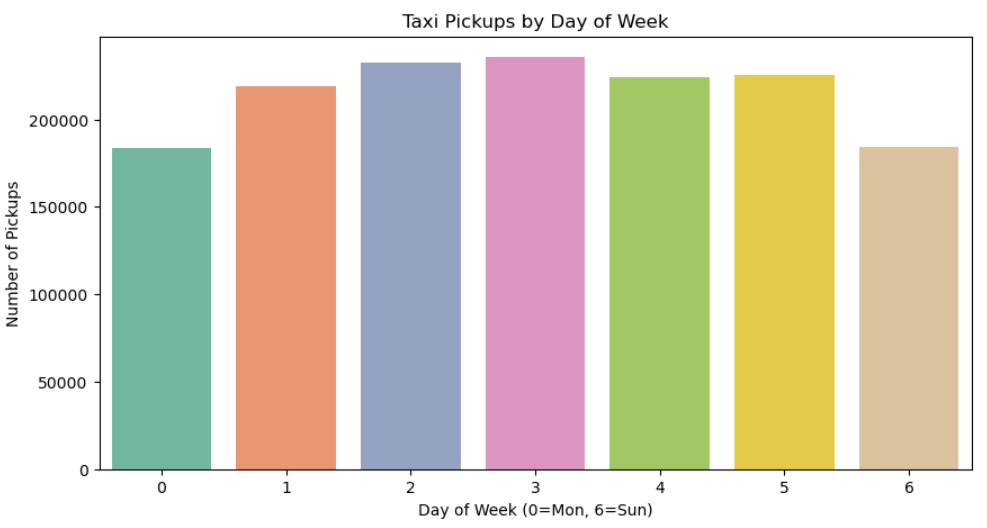
## 3.1.2. Analyse the distribution of taxi pickups by hours, days of the week, and months

**Temporal Analysis**

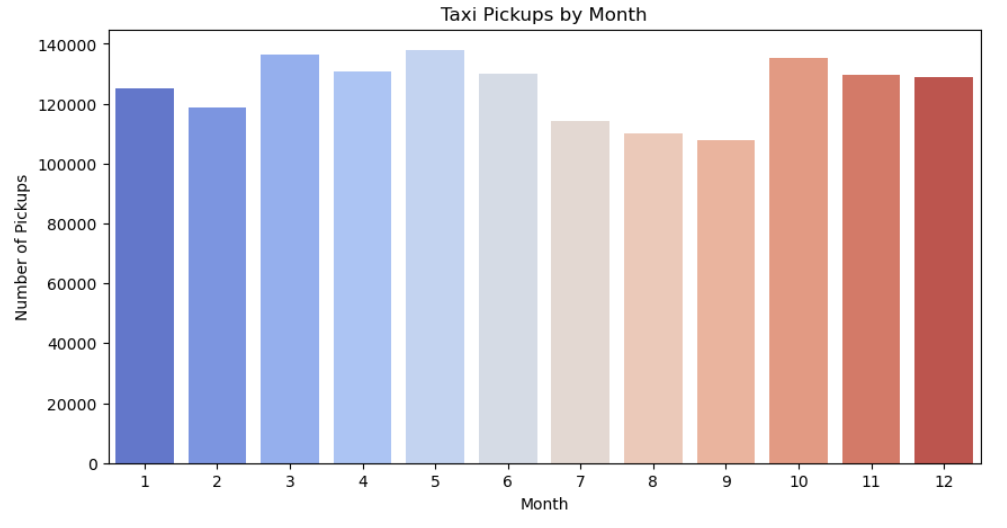
* To analyze the distribution of taxi pickups across different timeframes, new columns for hour, day of the week, and month were created from the pickup datetime field, and the distributions were visualized using count plots
* **Graph taxi pickups by hour of day – 15 – 19 hours have more pickups**



* **Graph counterplot by day of week**

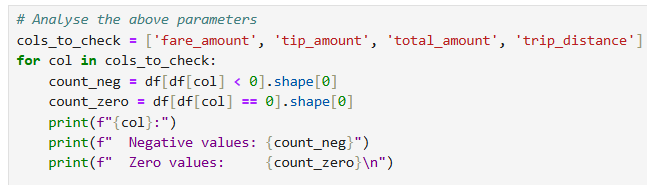


* **Graph counterplot by Monthly**

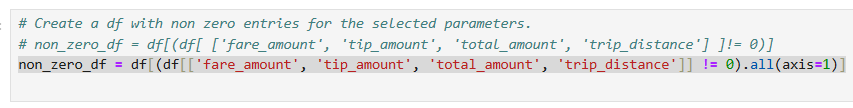
****

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

* To check the validity of financial parameters like fare\_amount, tip\_amount, total\_amount, and trip\_distance, the dataset was inspected for any zero or negative values using a filtering code snippet.



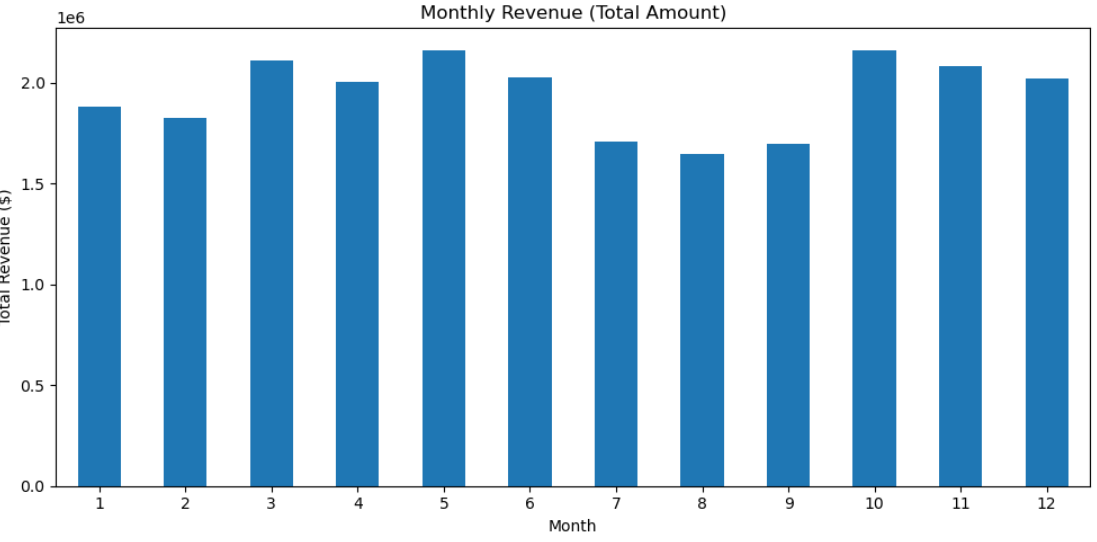
* Output:
  + fare\_amount:
    - Negative values: 0
    - Zero values: 250
  + tip\_amount:
    - Negative values: 0
    - Zero values: 311227
  + total\_amount:
    - Negative values: 0
    - Zero values: 176
  + trip\_distance:
    - Negative values: 0
    - Zero values: 10241
* All rows with zero values in key financial columns (fare\_amount, tip\_amount, total\_amount, and trip\_distance) were removed to ensure the dataset reflects only valid and meaningful taxi transactions.



* After removing I found indexes are changed, now reset index is done using reset\_index() method
* For future analysis, non\_zero\_df will be used.

## 3.1.4. Analyse the monthly revenue trends

* Next, the monthly revenue trend was analyzed by grouping the data by month and summing the total\_amount. The resulting values were plotted as a bar graph to visualize the revenue distribution across months.
* **Plotting the bar graph for visual representation**

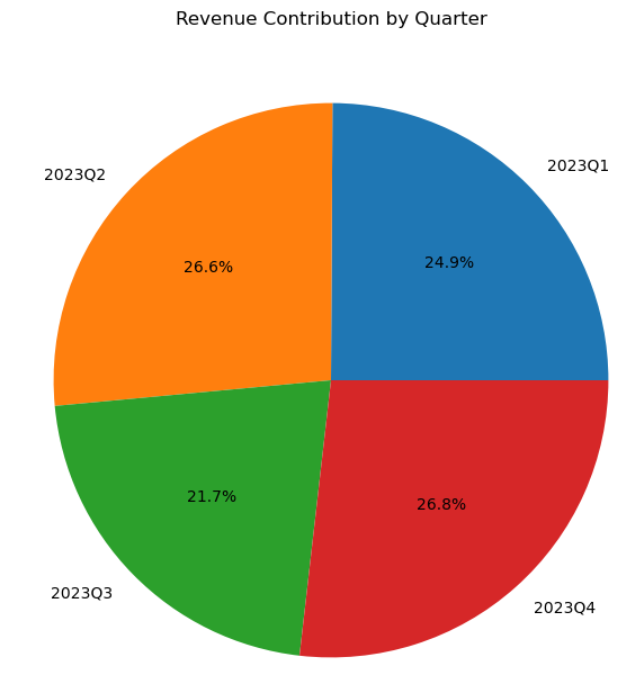


## 3.1.5. Find the proportion of each quarter’s revenue in the yearly revenue

* The quarter column was created by extracting the quarter information from the tpep\_pickup\_datetime field. Then, the proportion of revenue contributed by each quarter of the year 2013 was calculated based on the total\_amount.

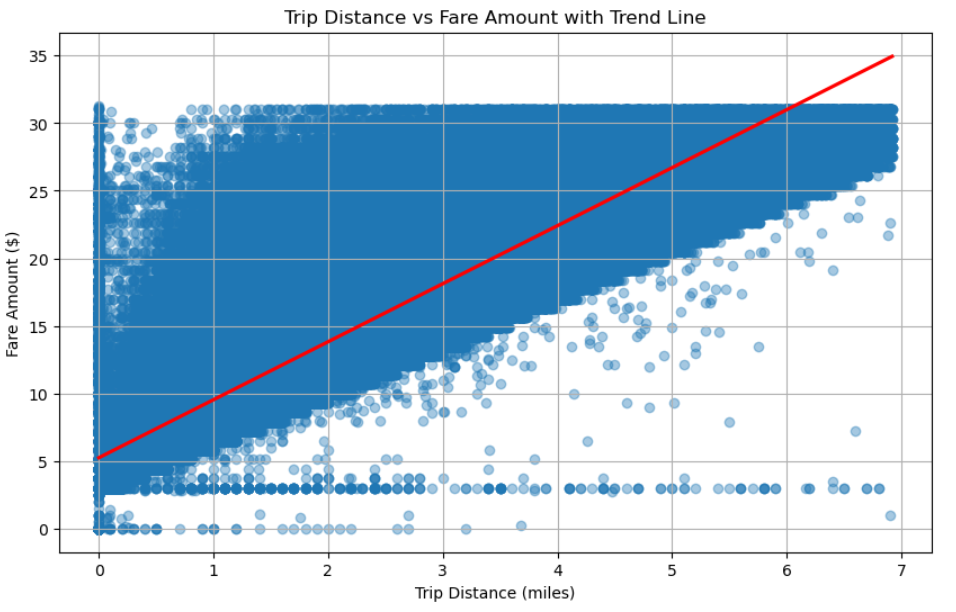


* The quarterly revenue was grouped and summed based on the quarter column, and the proportion of each quarter's contribution to the total annual revenue was calculated and printed for analysis.
* Now, plotting the quarterly revenue distribution to visualize the seasonal trends and identify the strongest revenue-generating periods.

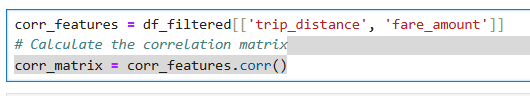


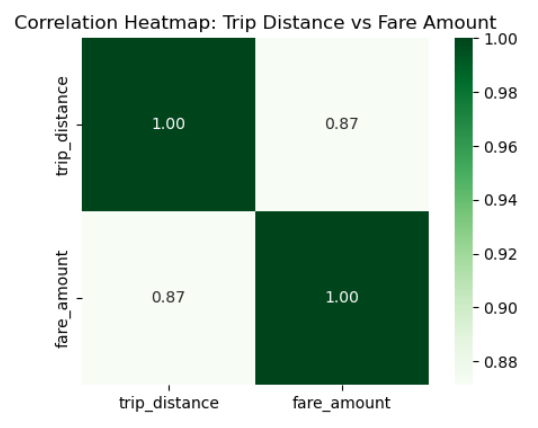
## 3.1.6. Analyse and visualise the relationship between and fare amount

* Next, visualized the relationship between trip\_distance and fare\_amount using a scatter plot to understand the distribution and trend between these variables.



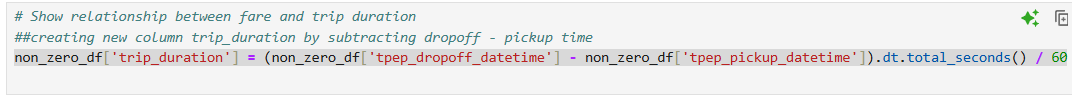
* Additionally, plotted a heatmap to find the correlation coefficient, confirming the strength and direction of the relationship.



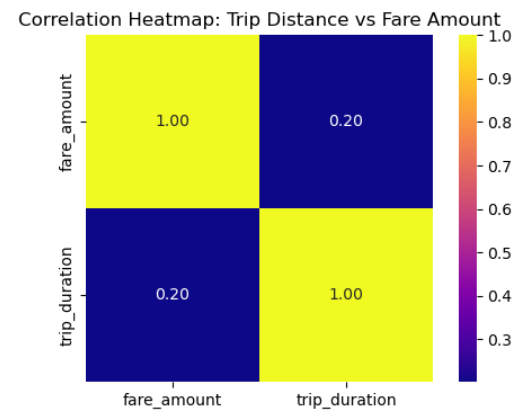


## 3.1.7. Analyse the relationship between fare/tips and trips/passengers

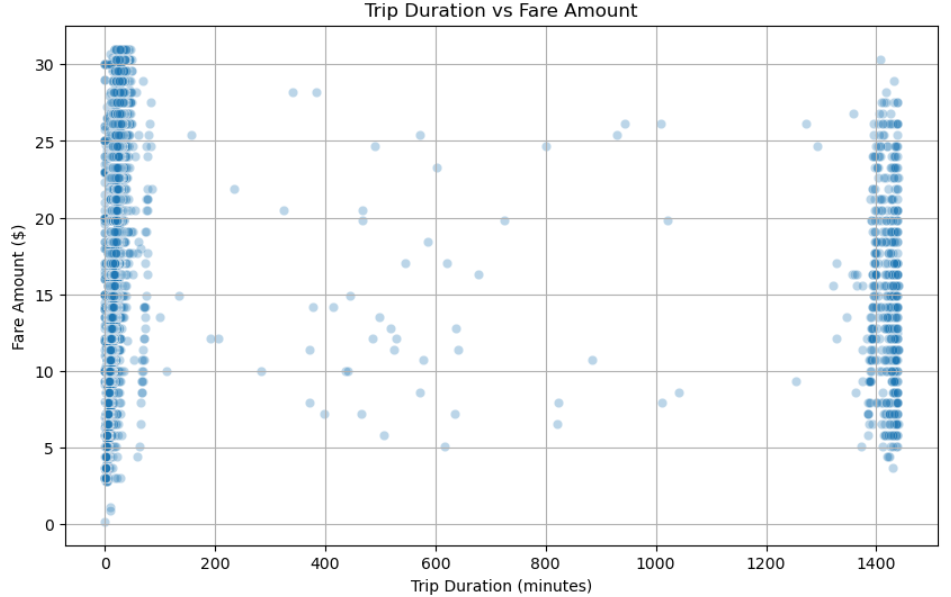
* Calculated and plotted the correlation between fare\_amount and trip duration (difference between pickup and dropoff times).
  + Created a new column trip\_duration by calculating the difference between tpep\_dropoff\_datetime and tpep\_pickup\_datetime, and converting the result into minutes using the below code



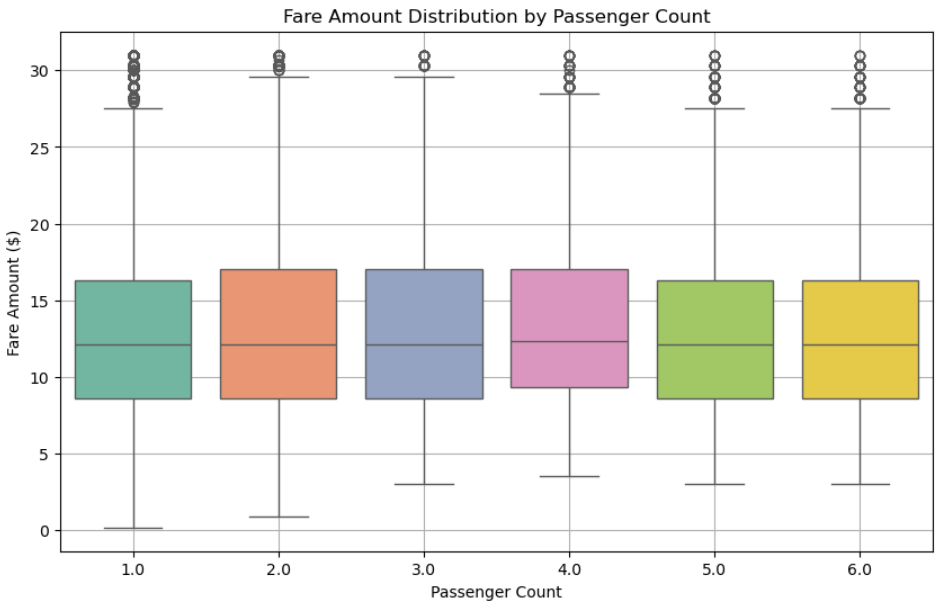
* + This trip\_duration column was then used to calculate and visualize the correlation between fare\_amount and trip duration.
  + Correlation in heatmap



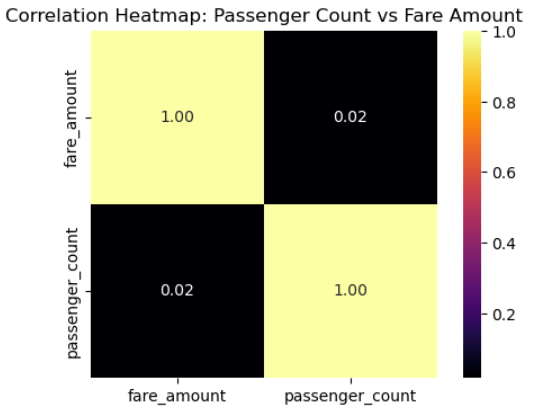
* + The relationship between fare\_amount and trip\_duration was visualized using a scatter plot to explore how fare changes with the duration of trips.



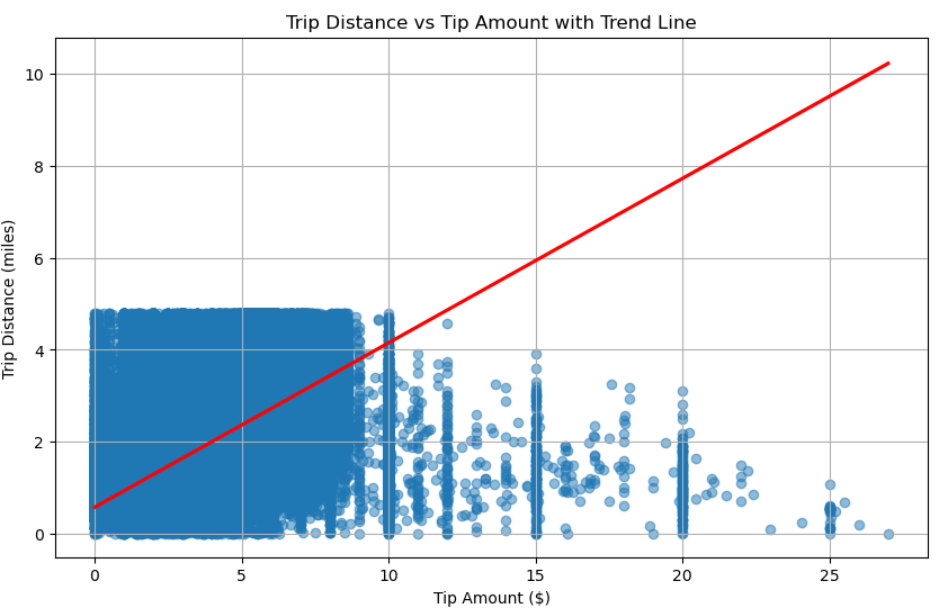
* The relationship between fare\_amount and passenger\_count was visualized using a box plot to understand the distribution of fare amounts for different numbers of passengers.



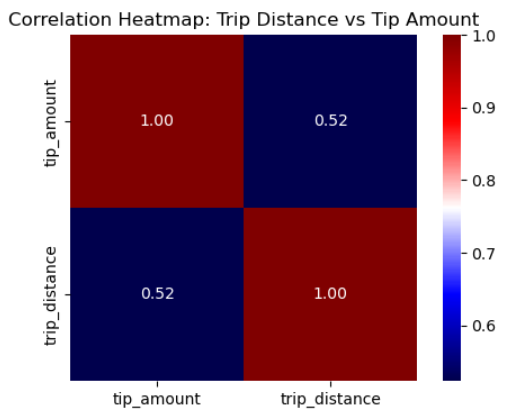
* Correclation between fare\_amount and Passenger Count has been calculated, plotted on the heatmap



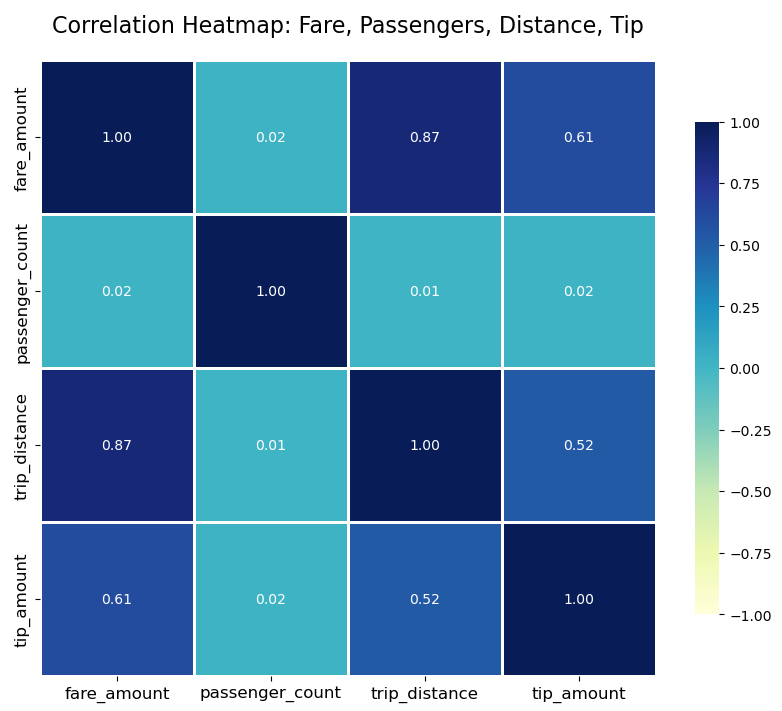
* The correlation between tip\_amount and trip\_distance will be analyzed by calculating the correlation coefficient between these two variables.
* A scatter plot will be created to visualize the relationship between tip amount and trip distance.



* Additionally, a heatmap will be plotted to show the strength and direction of their correlation, providing insights into how trip distance impacts the tip amount.

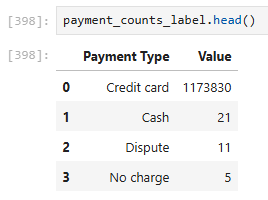


* Correlation for all the mentioned field (fare\_amount,passenger\_count, trip\_distance, tip\_amount)

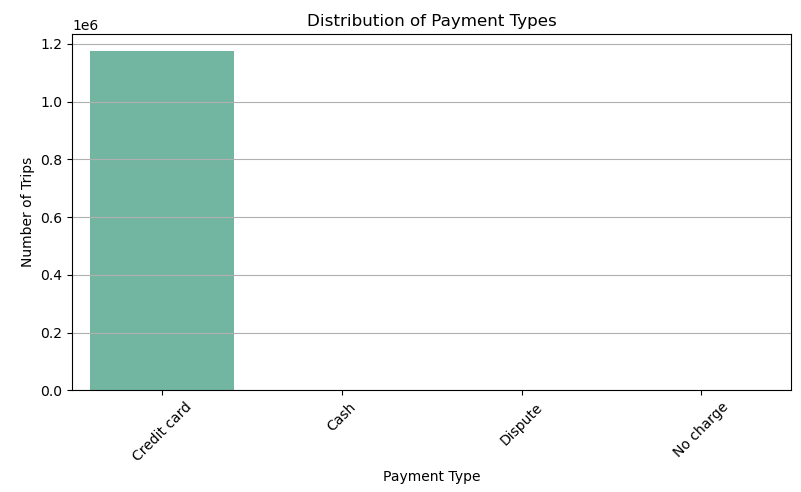


## 3.1.8. Analyse the distribution of different payment types

* Extract the payment\_types, map with data dictonary.

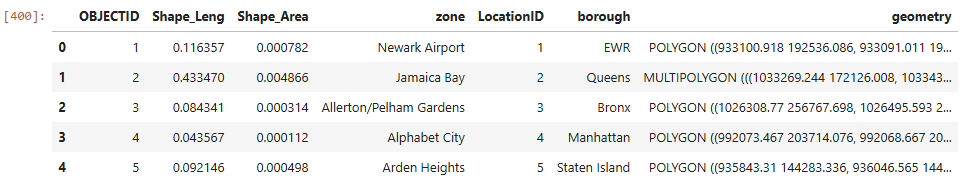


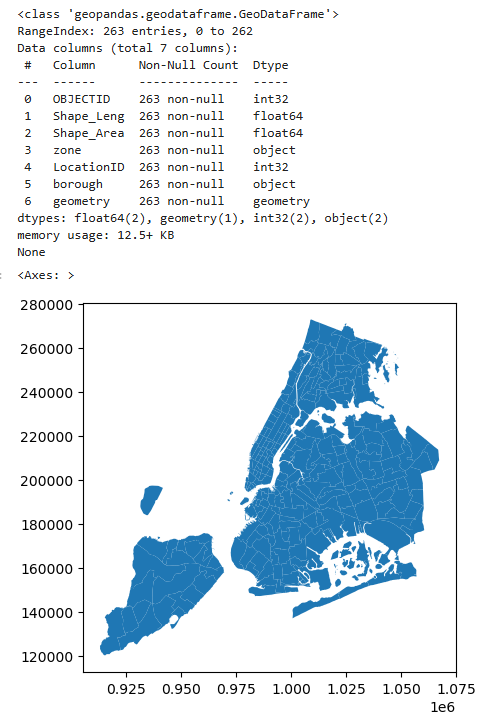
* 99% of the people prefer Credit card as their primary mode of payment.
* That reflect in the graph



## 3.1.9. Load the taxi zones shapefile and display it

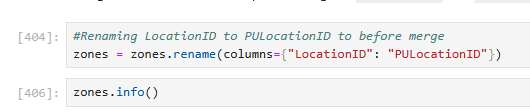
* Install the geopandas library to handle geospatial data.
* Read the shapefile (SHP) using geopandas to extract geographic information.
* Display the geographic data to visualize and analyze geographic patterns.
* Incorporate geo-coordinate details into the analysis for better insights.





## 3.1.10. Merge the zone data with trips data

* Merge non\_zero\_df with the zones dataframe to combine the data.
* This merged dataframe will be used for future analysis, integrating both trip-related and geographic information.





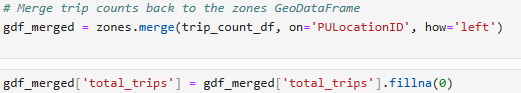
## 3.1.11. Find the number of trips for each zone/location ID

* Grouped the merged dataframe (df\_merged) by PULocationID to find the total number of trips at each pickup location using the size() method.
* Reset the index to create a clean dataframe trip\_count\_df with columns PULocationID and total\_trips.



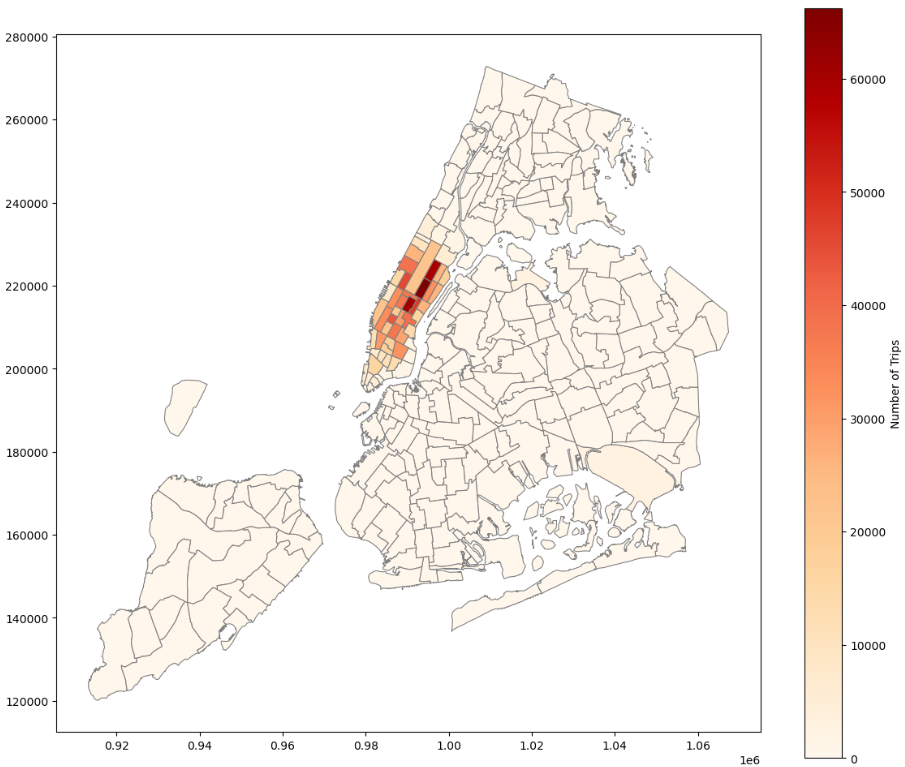
## 3.1.12. Add the number of trips for each zone to the zones dataframe

* Merged the trip\_count\_df with the zones GeoDataFrame using PULocationID as the key and left join method to retain all zone information.
* Created a new GeoDataFrame gdf\_merged that contains geographic details along with the trip counts for each location.



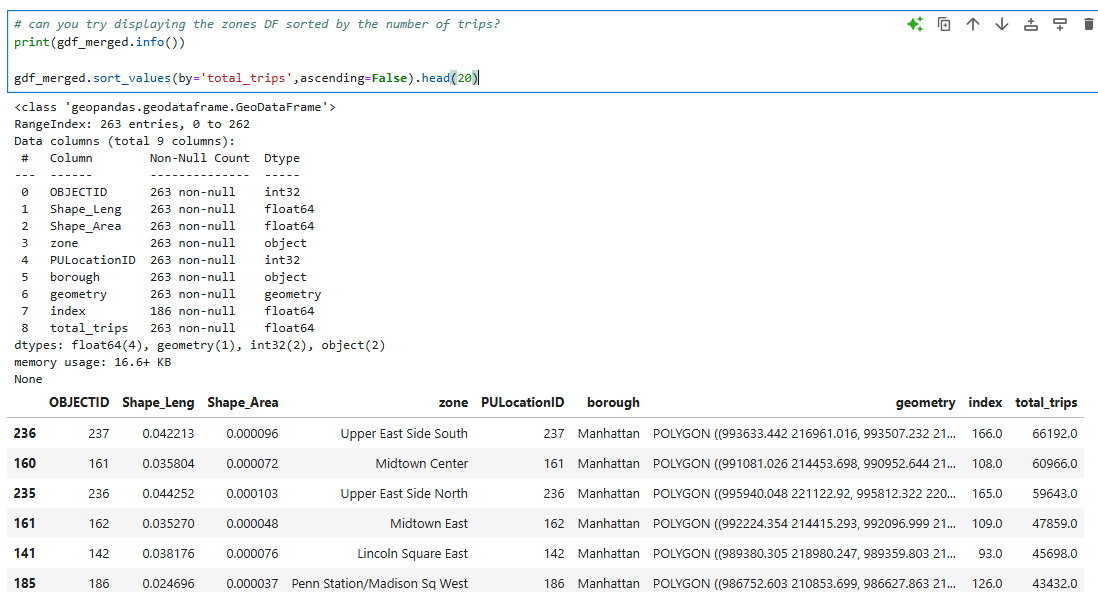
## 3.1.13. Plot a map of the zones showing number of trips

* Used the gdf\_merged GeoDataFrame to plot the map, where the color of each zone represents the number of trips, using the "OrRd" color map.
* AS converted NaN as Zero

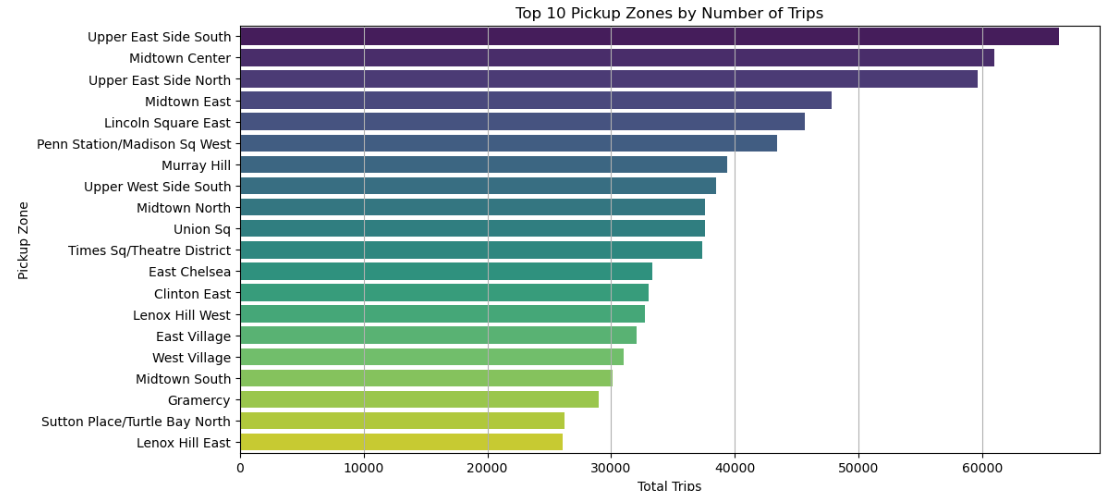


## 3.1.14. Conclude with results

* Merged the GeoDataFrame containing location zone geometries with the DataFrame holding the number of trips per pickup location ID, ensuring that spatial information and trip counts were combined for further geospatial analysis and visualization.



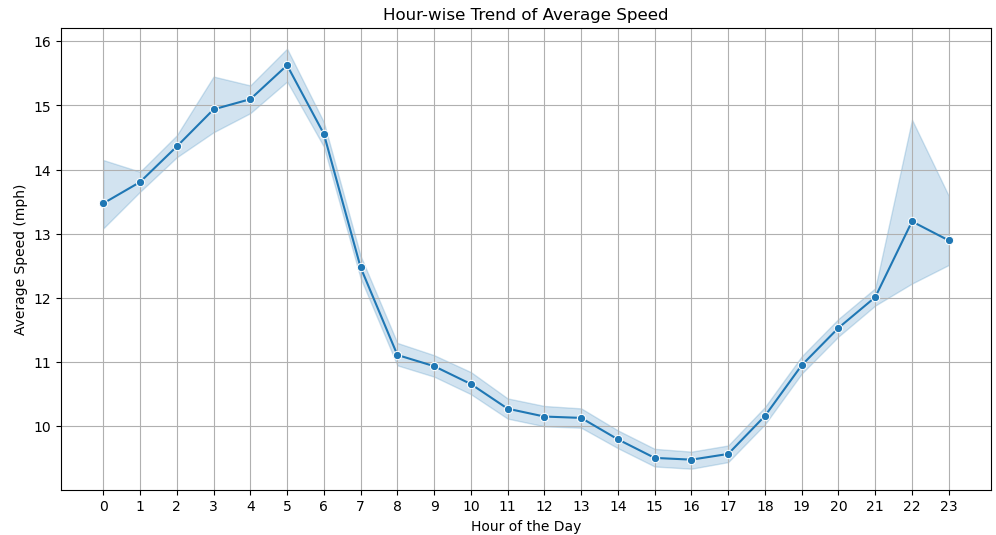
* A bar plot was created to display the top 20 pickup locations with the highest number of trips, helping to easily visualize which areas had the most taxi activity.



## 3.2.Detailed EDA: Insights and Strategies

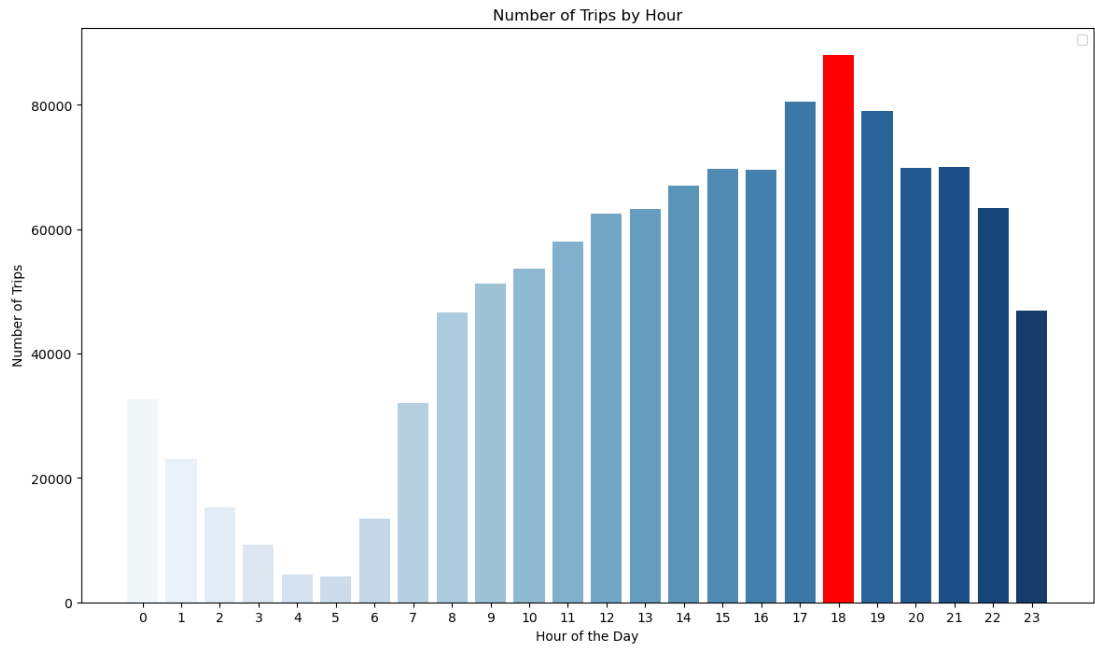
## 3.2.1. Identify slow routes by comparing average speeds on different routes

* Grouped the dataset by PULocationID, DOLocationID, and pickup\_hour to calculate:
* Average trip distance (avg\_distance)
* Average trip duration (avg\_duration)
* Calculated the average speed (in miles per hour) for each route using the formula:
* avg\_speed\_mph = (avg\_distance / avg\_duration) \* 60
* Computed the overall mean average speed across all routes to serve as a benchmark.
* Identified slow routes by filtering the routes where the average speed was less than the benchmark.
* Sorted the slow routes in ascending order based on their average speed to highlight the slowest ones
* Plot it in line graph



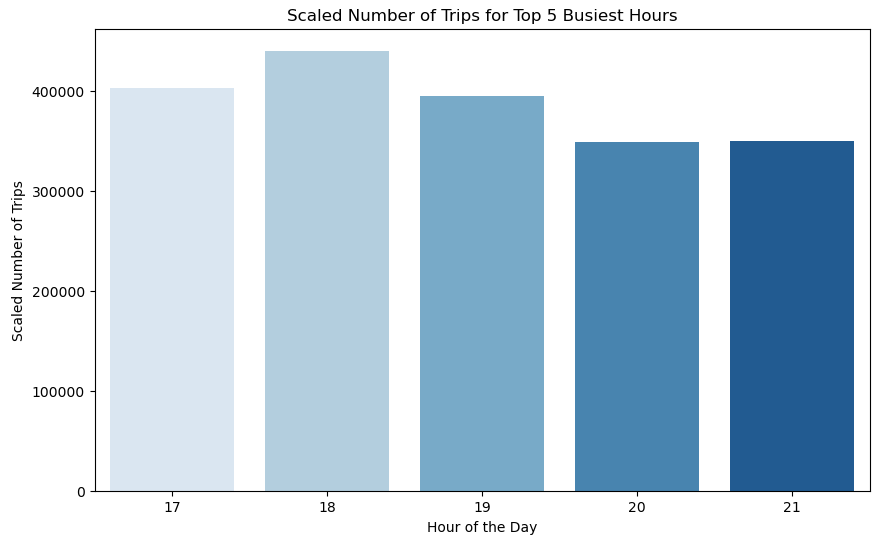
## 3.2.2. Calculate the hourly number of trips and identify the busy hours

* Calculated the number of trips for each hour of the day by grouping the data based on the pickup\_hour column.
* Identified the busiest hour by finding the hour with the maximum number of trips.
* Plotted the distribution of trips per hour using a bar plot, with the busiest hour highlighted distinctly to show the peak activity



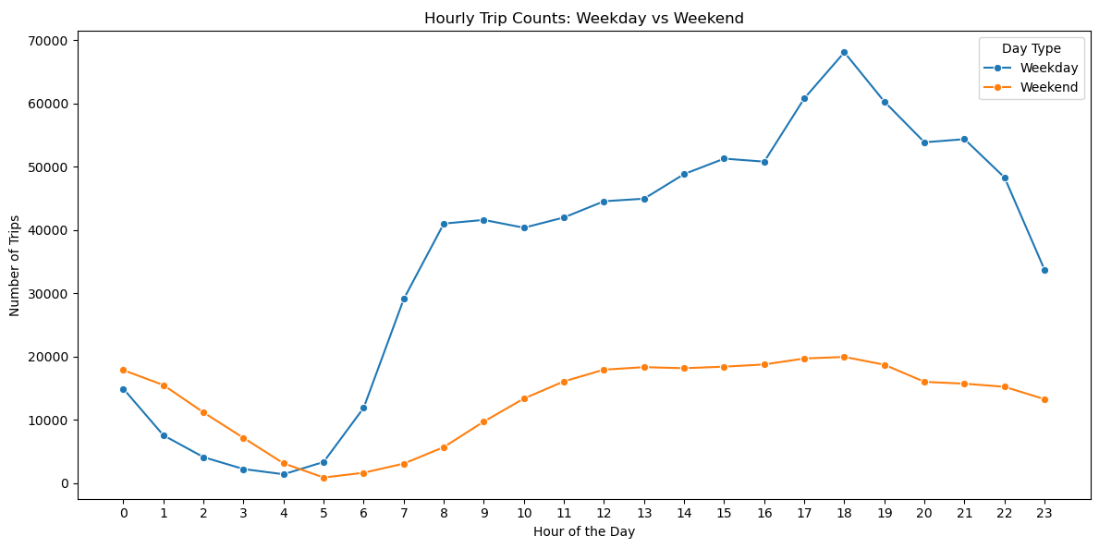
## 3.2.3. Scale up the number of trips from above to find the actual number of trips

* Identified the top 5 busiest hours by sorting the trip counts by hour in descending order.
* Scaled up the number of trips by dividing the counts by the sampling fraction (0.2), to estimate the total trips for each of those hours.
* Plotted the scaled trip counts for the top 5 busiest hours using a bar graph for clear visualization.



## 3.2.4. Compare hourly traffic on weekdays and weekends

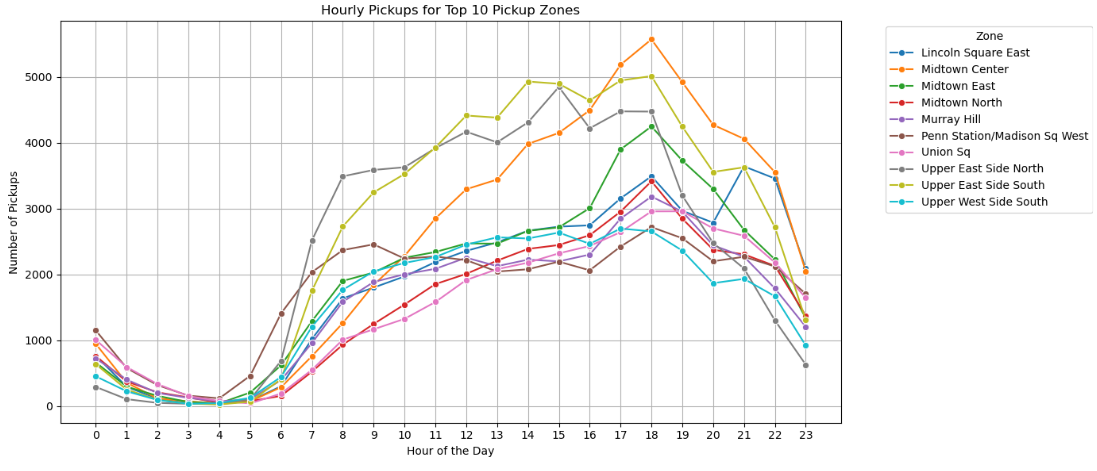
* Added a new column is\_weekend in the dataframe based on the day\_of\_week, marking weekends (Saturday and Sunday) as 1 and weekdays as 0.
* Grouped the data by pickup\_hour and is\_weekend to count the number of trips for each hour separately for weekdays and weekends.
* Mapped the is\_weekend values to a new day\_type column with labels Weekday and Weekend for better readability.
* Plotted a **line graph** using seaborn.lineplot() to compare the **hourly trip counts** between weekdays and weekends:
* **X-axis**: Pickup hour
* **Y-axis**: Number of trips
* **Hue**: Day type (Weekday or Weekend) with different colors



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

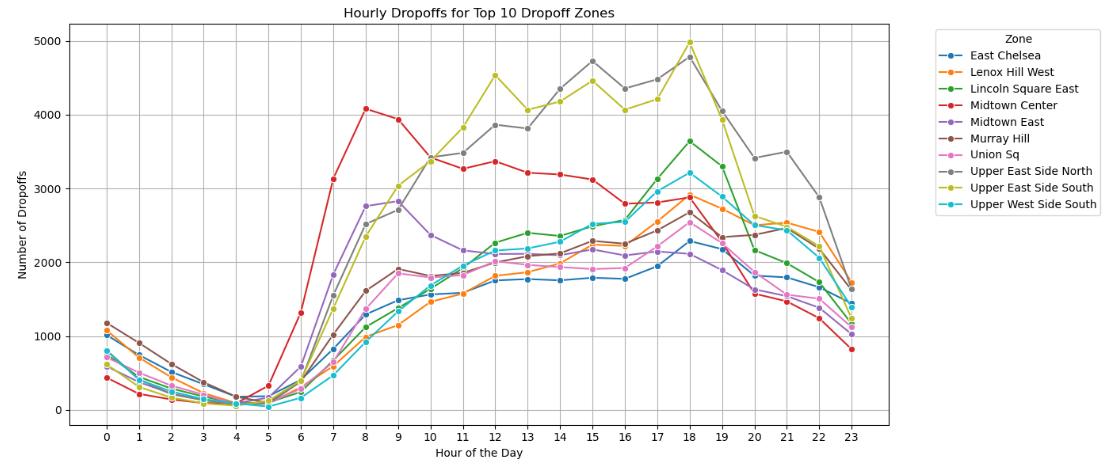
**Top 10 Pick up analysis**

* Grouped the dataset by PULocationID and pickup\_hour to calculate the number of pickups at each location and hour.
* Aggregated the data to find the top 10 pickup zones based on the highest total number of pickups.
* Filtered the pickup data to include only these top 10 pickup zones.
* Merged the filtered pickup data with zone information to map the PULocationID to corresponding zone names



**Top 10 Drop off Analysis**

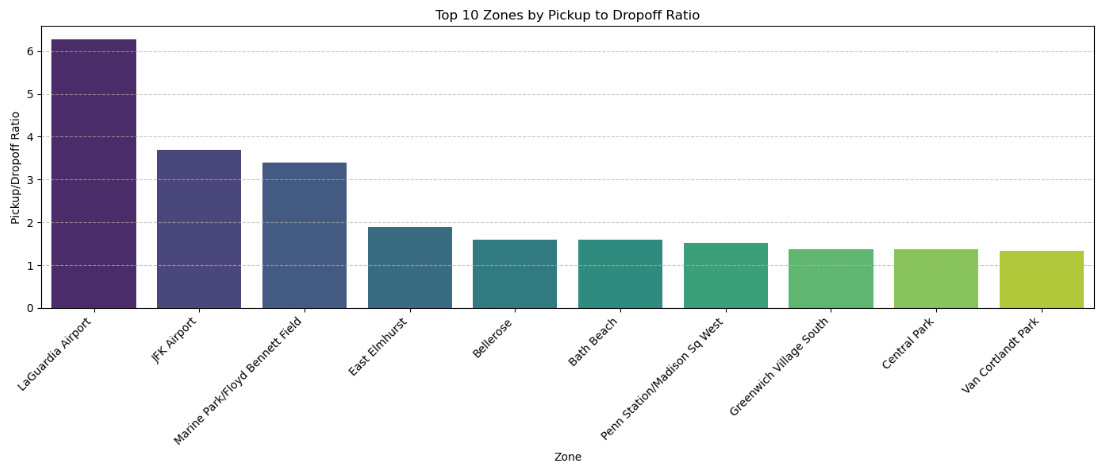
* Grouped the dataset by DOLocationID and dropoff\_hour to calculate the number of drop-offs at each location and hour.
* Aggregated the data to find the top 10 drop-off zones based on the highest total number of drop-offs.
* Filtered the drop-off data to include only these top 10 drop-off zones.
* Merged the filtered drop-off data with zone information to map the DOLocationID to corresponding zone names.



## 3.2.6. Find the ratio of pickups and dropoffs in each zone

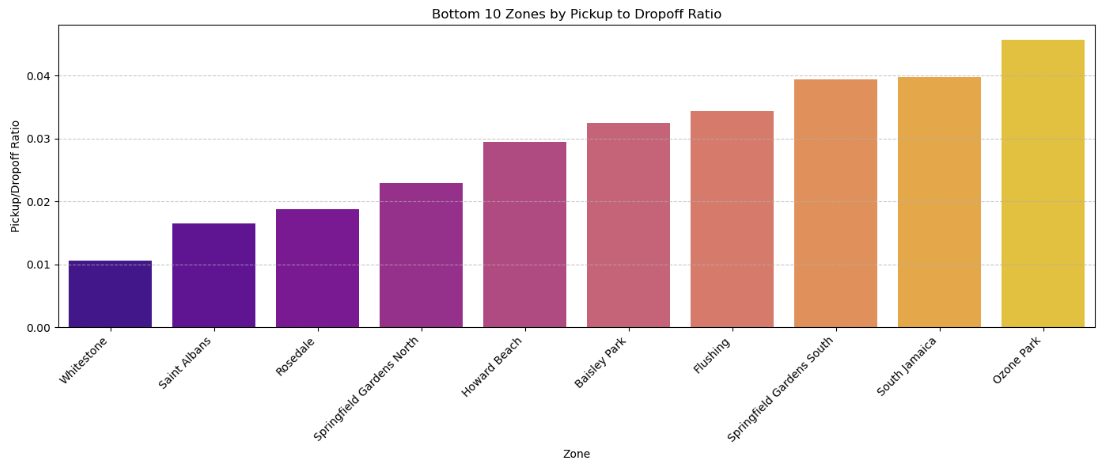
Top 10 Zones with the Highest Pickup-to-Dropoff Ratio:

* The code top10\_ratio = zone\_flow.sort\_values(by='pickup\_to\_dropoff\_ratio', ascending=False).head(10) selects the top 10 zones where the pickup-to-dropoff ratio is the highest. This indicates which zones are more focused on pickups compared to drop-offs. A higher ratio suggests that passengers are more frequently getting picked up in these zones relative to those being dropped off.
* We then displayed the relevant data for these top zones using print(top10\_ratio[['zone', 'pickup\_count', 'dropoff\_count', 'pickup\_to\_dropoff\_ratio']]).



**Top 10 Zones with the Lowest Pickup-to-Dropoff Ratio:**

* Similarly, the code bottom10\_ratio = zone\_flow.sort\_values(by='pickup\_to\_dropoff\_ratio', ascending=True).head(10) identifies the bottom 10 zones with the lowest pickup-to-dropoff ratio. This implies that these zones see more drop-offs compared to pickups.
* We printed the relevant information for these bottom zones using print(bottom10\_ratio[['zone', 'pickup\_count', 'dropoff\_count', 'pickup\_to\_dropoff\_ratio']]).

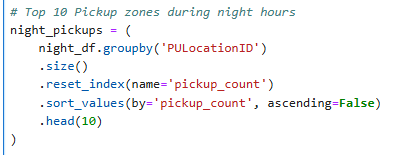


## 3.2.7. Identify the top zones with high traffic during night hours

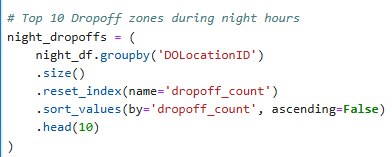
* The dataset was filtered to capture only the rows where the pickup hour is between 11 PM and 5 AM.



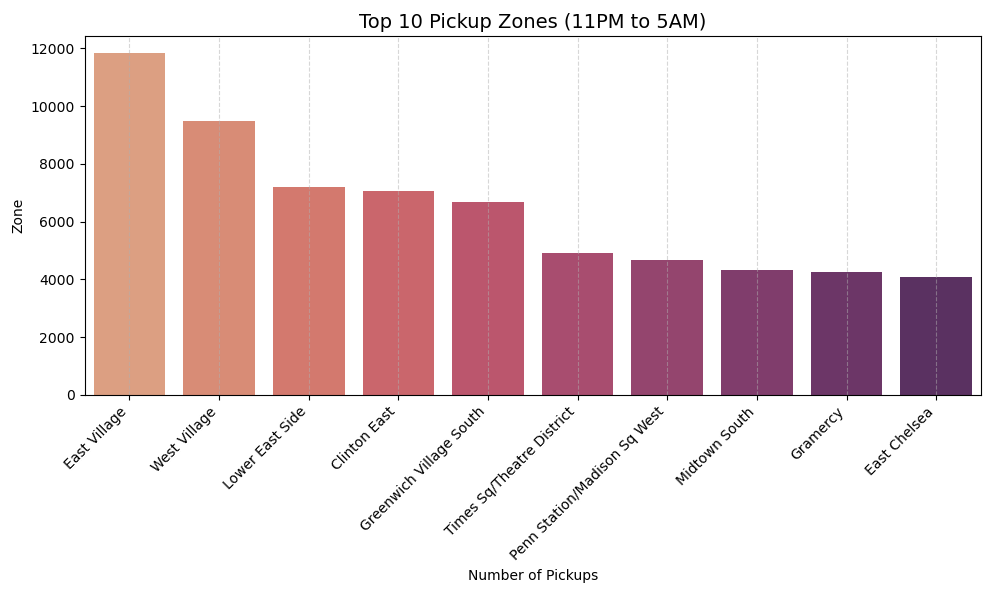
* The dataset was grouped by PULocationID (pickup location) to determine the number of pickups in each zone during the night. The code:



* A similar approach was used to identify the top 10 dropoff zones by grouping the dataset by DOLocationID (dropoff location)



* Plotted on the bar graph



## 3.2.8. Find the revenue share for nighttime and daytime hours

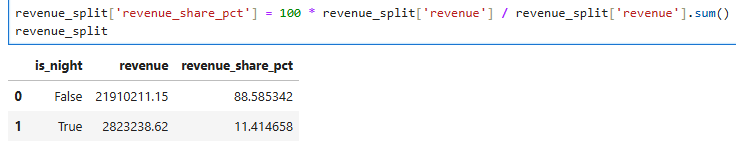
* The nighttime hours were defined as the time range between 11 PM and 5 AM. This was done by creating a list of hours for the night period



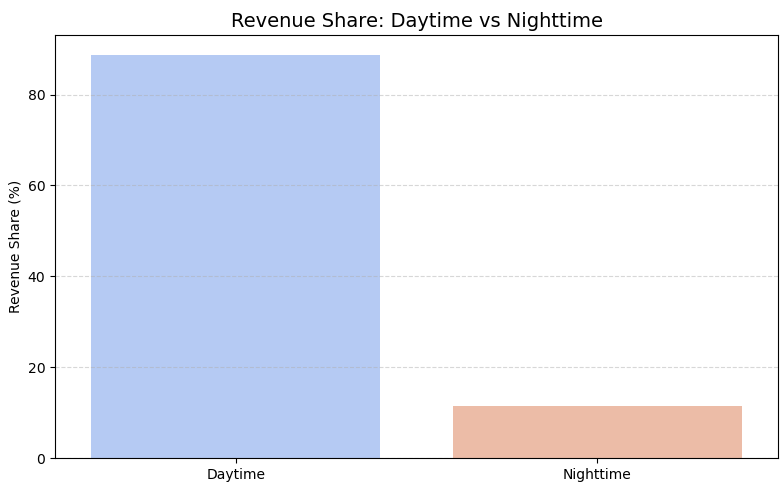
* A new column, **is\_night**, was added to the dataframe, where trips during the nighttime hours were marked as True and other trips as False



* After summing the revenues for nighttime and daytime hours, the revenue share percentage was calculated by dividing each group’s revenue by the total revenue and multiplying by 100



* Plotting on the different on the barplot



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

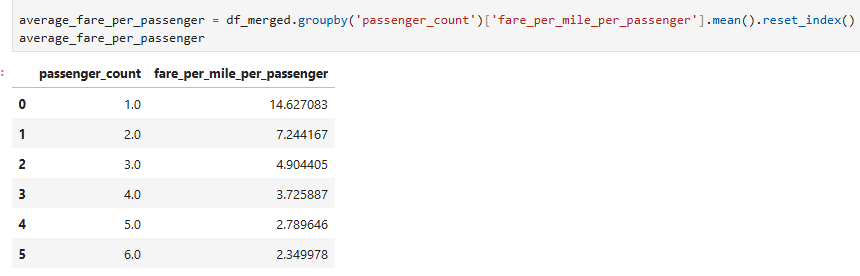
* First, we need to calculate the fare per mile for each trip. This can be done by dividing the total fare (total\_amount) by the total distance traveled (trip\_distance):



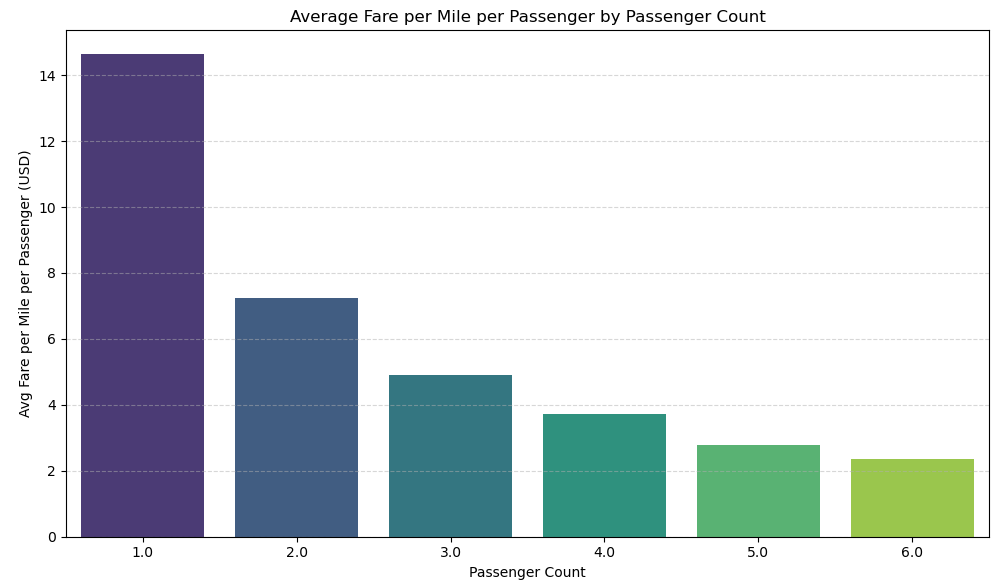
* Next, we calculate the fare per mile per passenger by dividing the fare per mile by the passenger count for each trip. This assumes that the passenger count is given in the dataset as passenger\_count:



* After calculating the fare per mile per passenger, we group the data by passenger\_count and calculate the average fare per mile per passenger for each passenger count group:

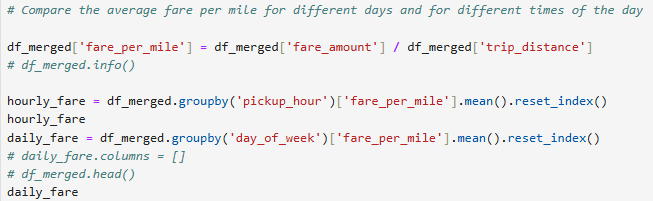


* The final result, average\_fare\_per\_passenger, will show the average fare per mile per passenger for each passenger count.

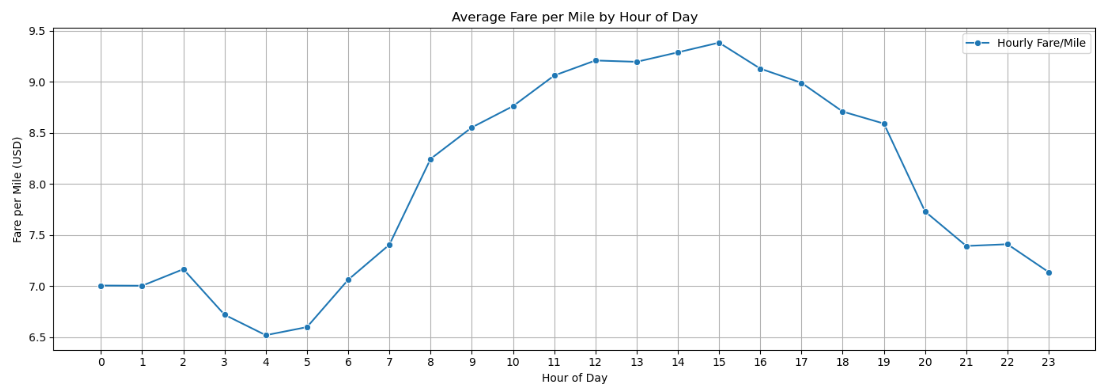


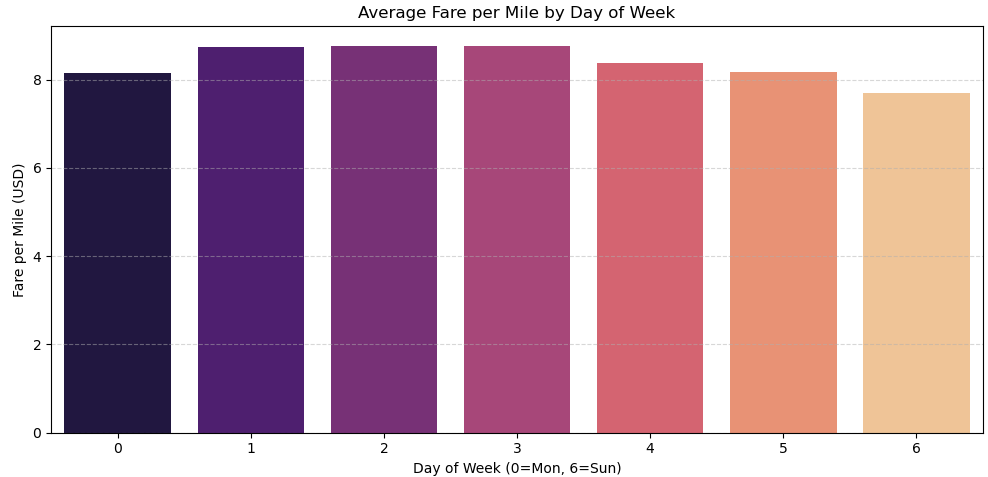
## 3.2.10. Find the average fare per mile by hours of the day and by days of the week

* The fare per mile is calculated by dividing the total fare by the trip distance for each trip.
* The average fare per mile is computed for each hour of the day by grouping the data by pickup hour and calculating the mean fare per mile for each hour.
* Similarly, the average fare per mile is calculated for each day of the week by grouping the data by the day of the week and calculating the mean fare per mile for each day.



* The results can be visualized using line or bar plots to compare how the fare per mile varies across different hours of the day and days of the week.



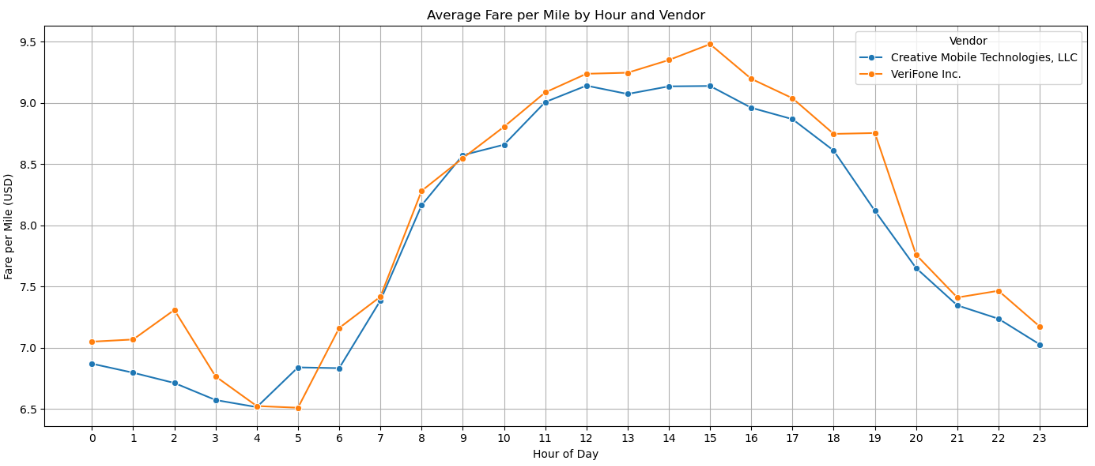


## 3.2.11. Analyse the average fare per mile for the different vendors

* The data is categorized by vendor and hour of the day, with each trip’s fare per mile calculated.
* The vendor\_name is mapped to the corresponding VendorID to identify the vendor for each trip.
* The average fare per mile is computed for each vendor across different pickup hours by grouping the data by vendor and hour, and calculating the mean fare per mile.

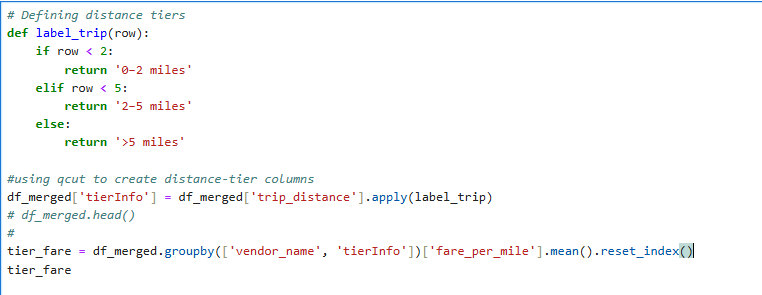


* The results show how the fare per mile varies for each vendor at different times of the day, allowing a comparison of the pricing trends for each vendor.

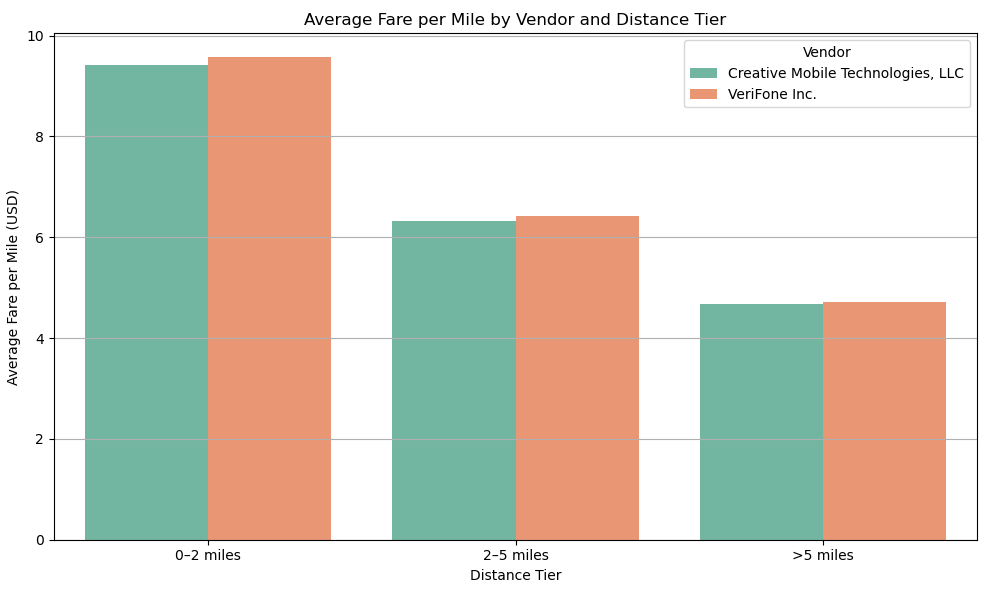


## 3.2.12. Compare the fare rates of different vendors in a distance-tiered fashion

* The analysis is based on distance tiers to compare fare rates of different vendors, categorizing trips into three groups: distances up to 2 miles, from 2 to 5 miles, and more than 5 miles.
* A function is used to label each trip according to its distance, categorizing them into the respective tiers.
* The average fare per mile is then calculated for each vendor within each distance tier, allowing a comparison of how fare rates differ based on the trip distance.

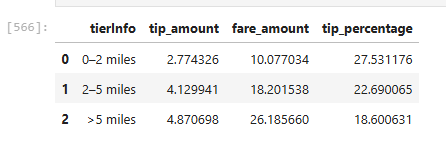


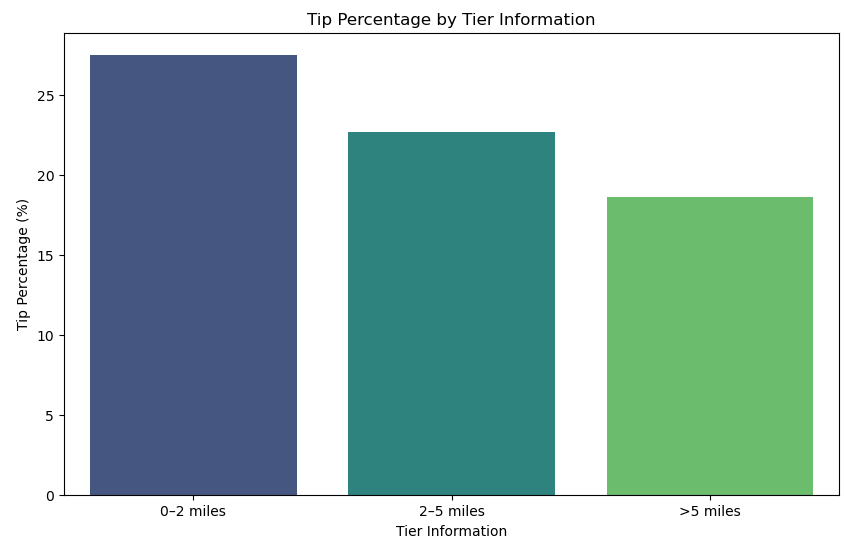
* The results provide insights into how fare per mile varies for different vendors across the three distance ranges, enabling a tiered comparison of fare rates.



## 3.2.13. Analyse the tip percentages

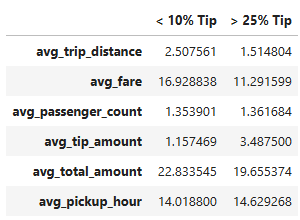
* Tip percent vs TierInfo

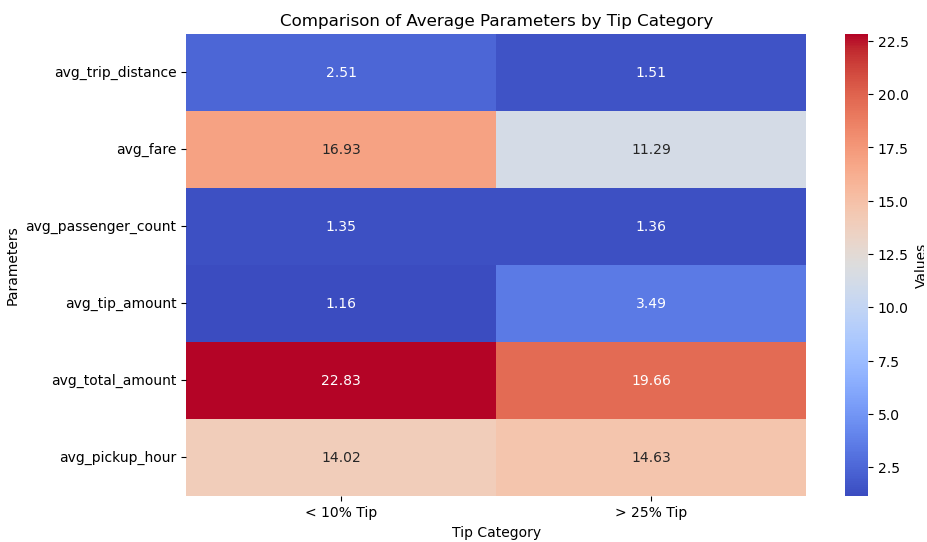




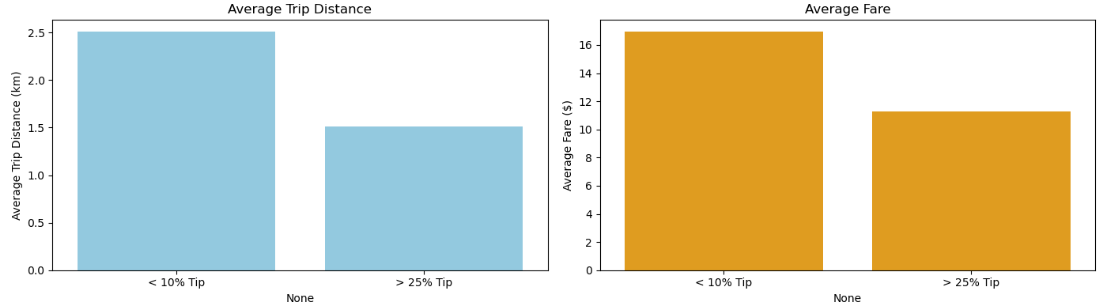
Additionally

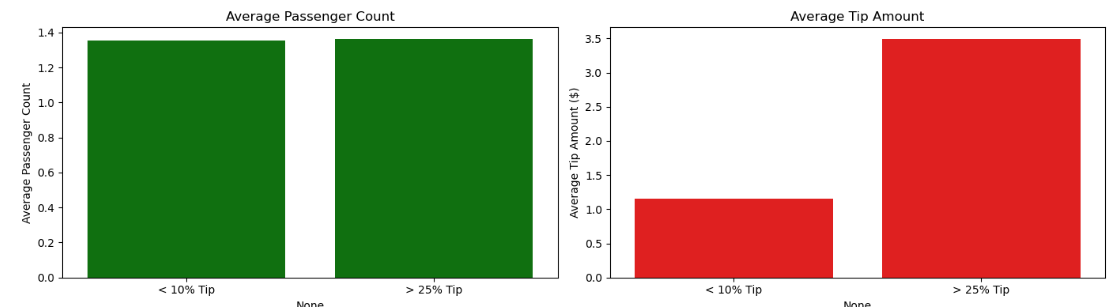
* Low tip vs High tip scenario
  + Compare trips with tip percentage < 10% to trips with tip percentage > 25%

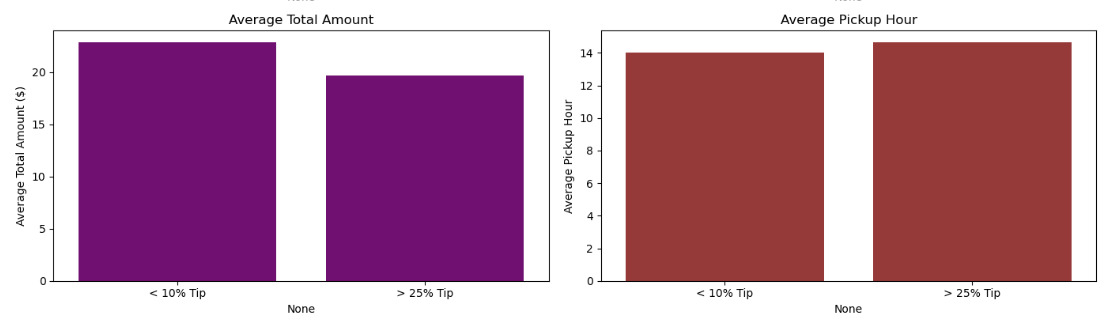




* Average other factors – high and low tip comparison

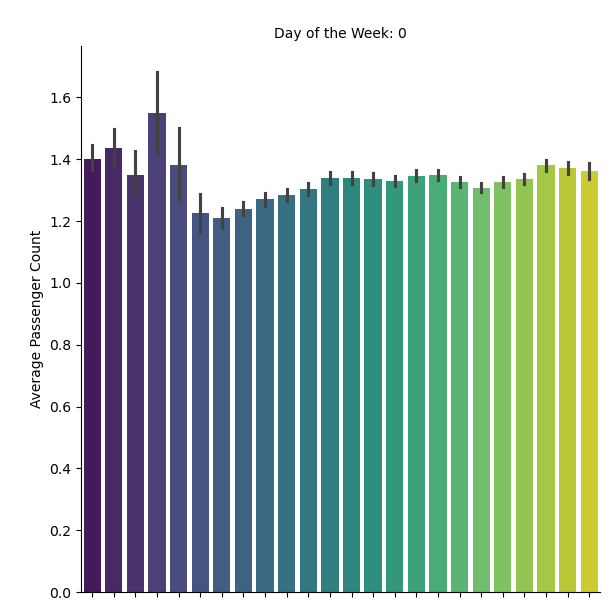


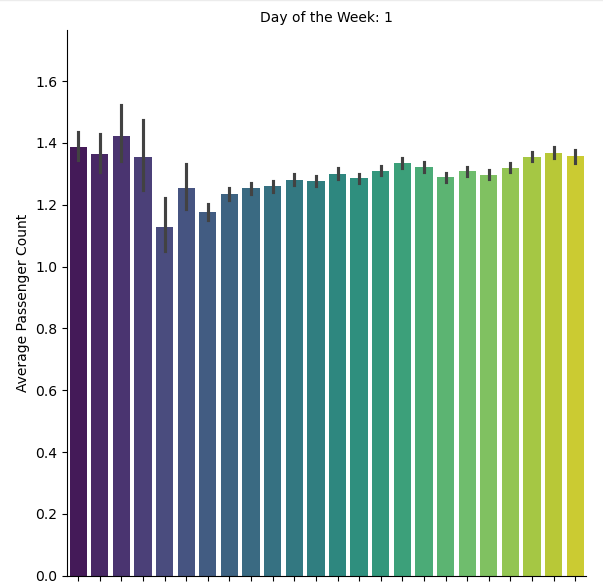


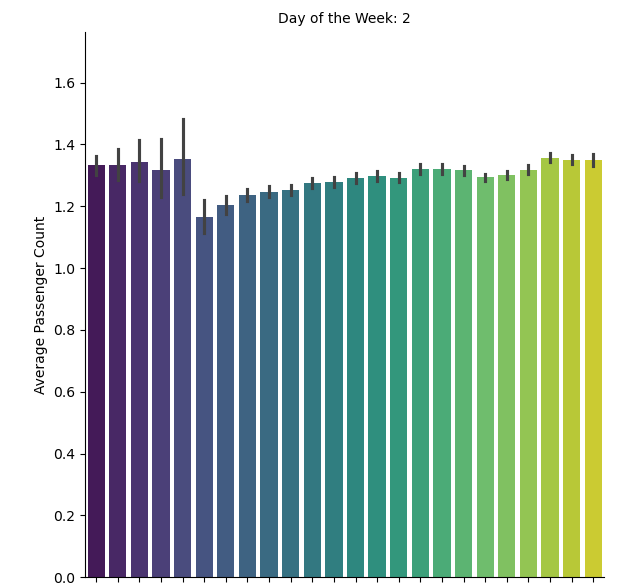


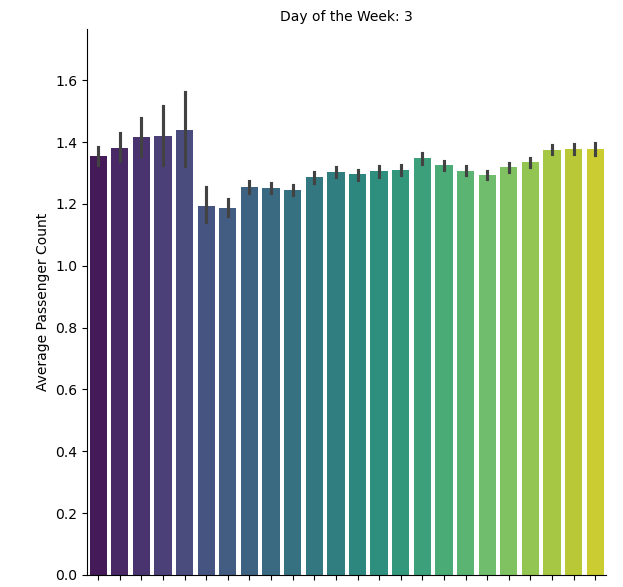
## 3.2.14. Analyse the trends in passenger count

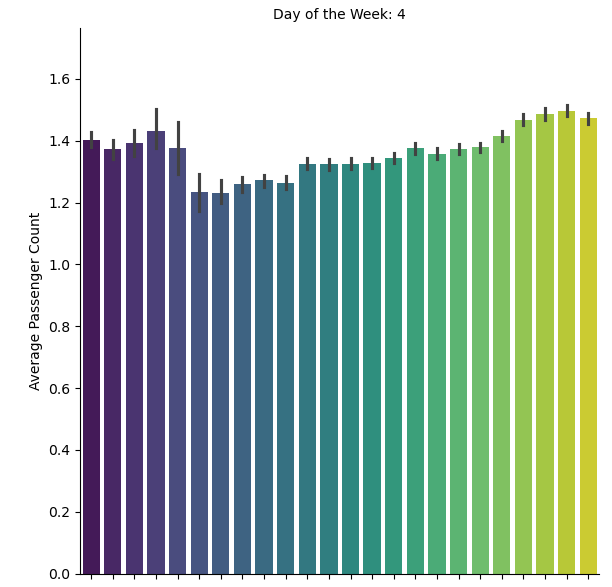
* Analyse the variation of passenger count across hours and days of the week.

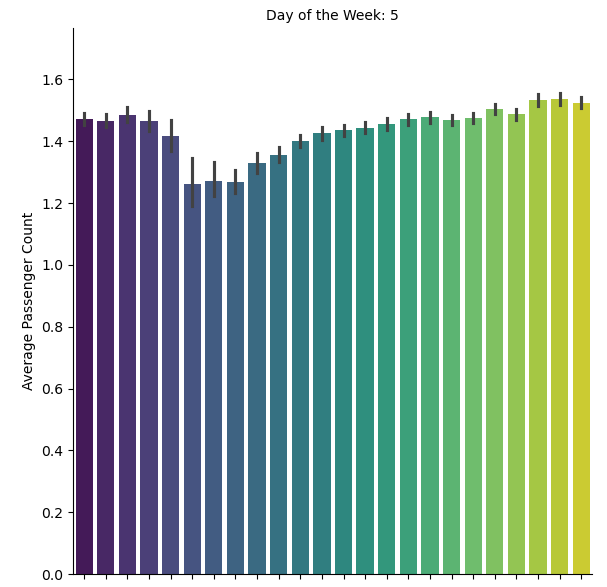


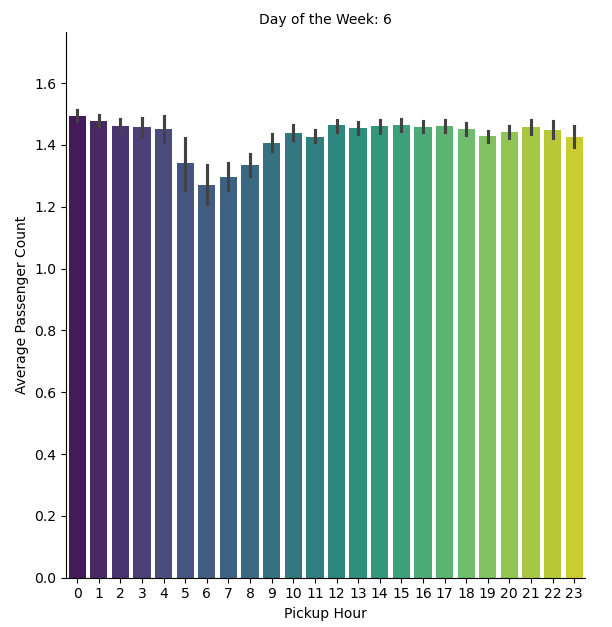








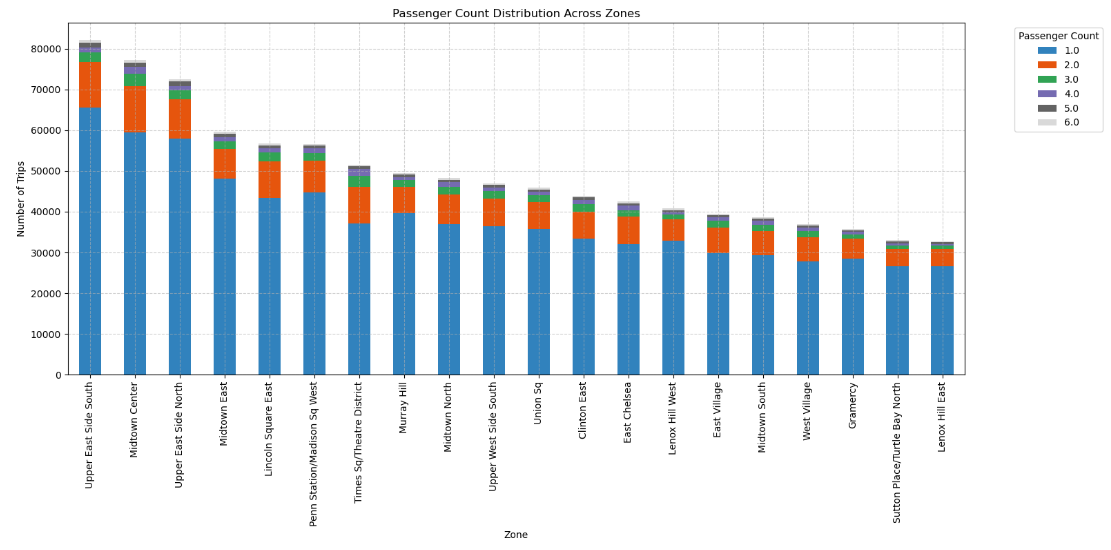


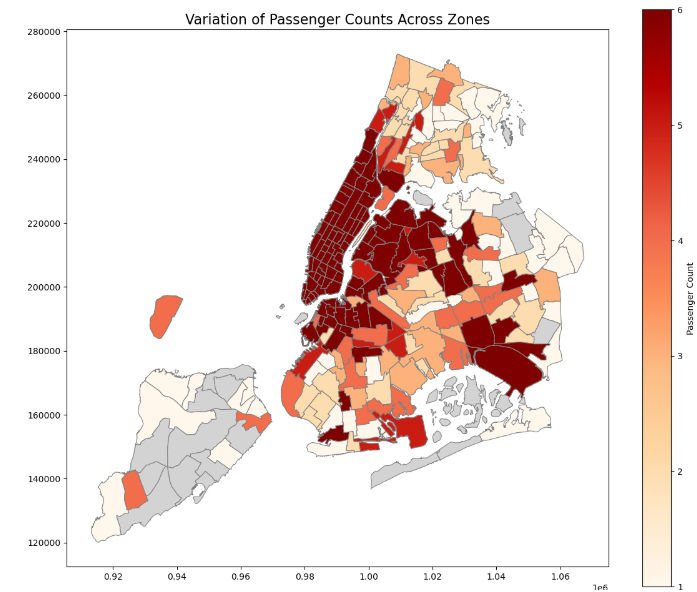


## 3.2.15. Analyse the variation of passenger counts across zones

* Analyse the variation of passenger counts across zones

Top 10 zone wise plot

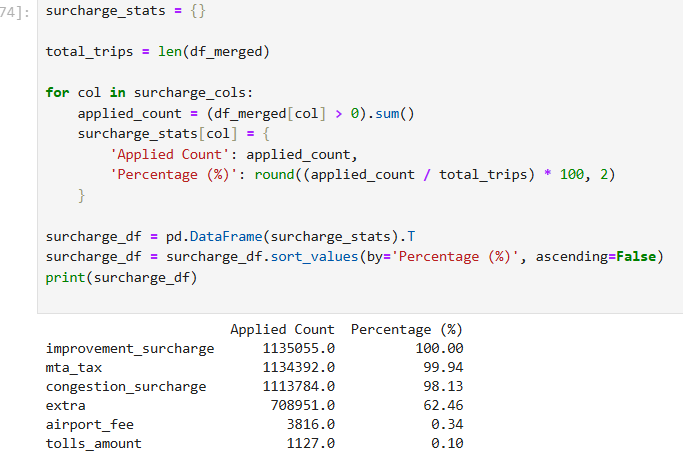


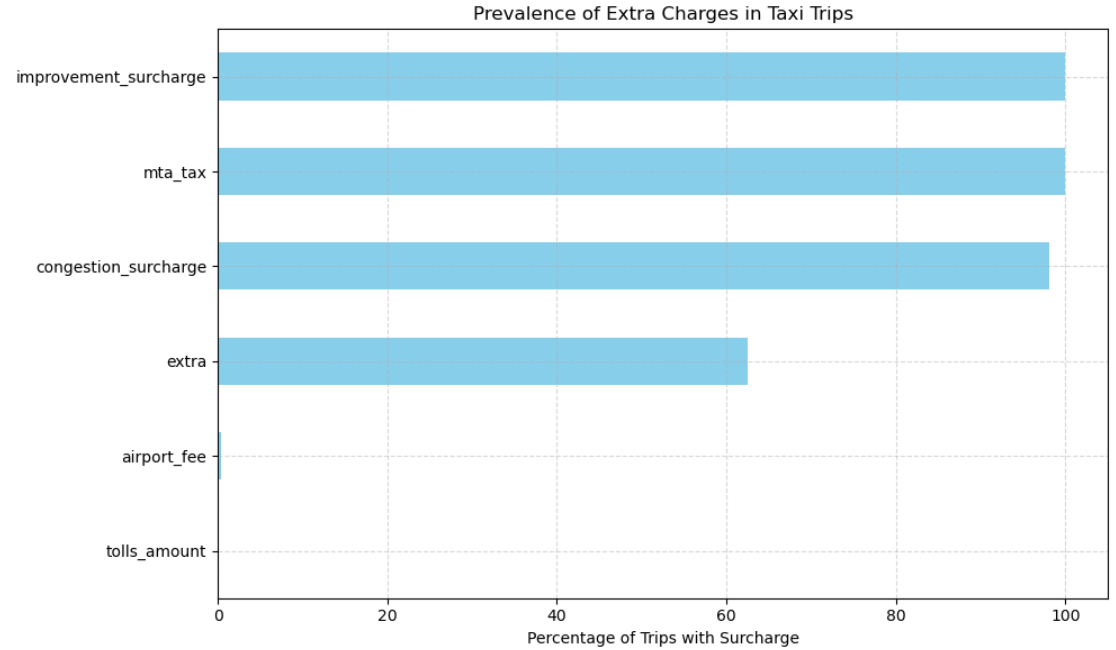


* High Passenger Count Zones: Darker zones likely indicate areas with high passenger counts (more trips or passengers). These might be zones with higher demand, such as central business districts, transportation hubs, or popular neighborhoods.
* Low Passenger Count Zones: Lighter-colored zones suggest areas with lower passenger counts, which could be less densely populated areas, residential zones, or regions with less traffic.
* Identifying Trends: By observing the geographic patterns, you can infer areas where passenger demand is consistently high or low, potentially indicating hotspots or underutilized zones.

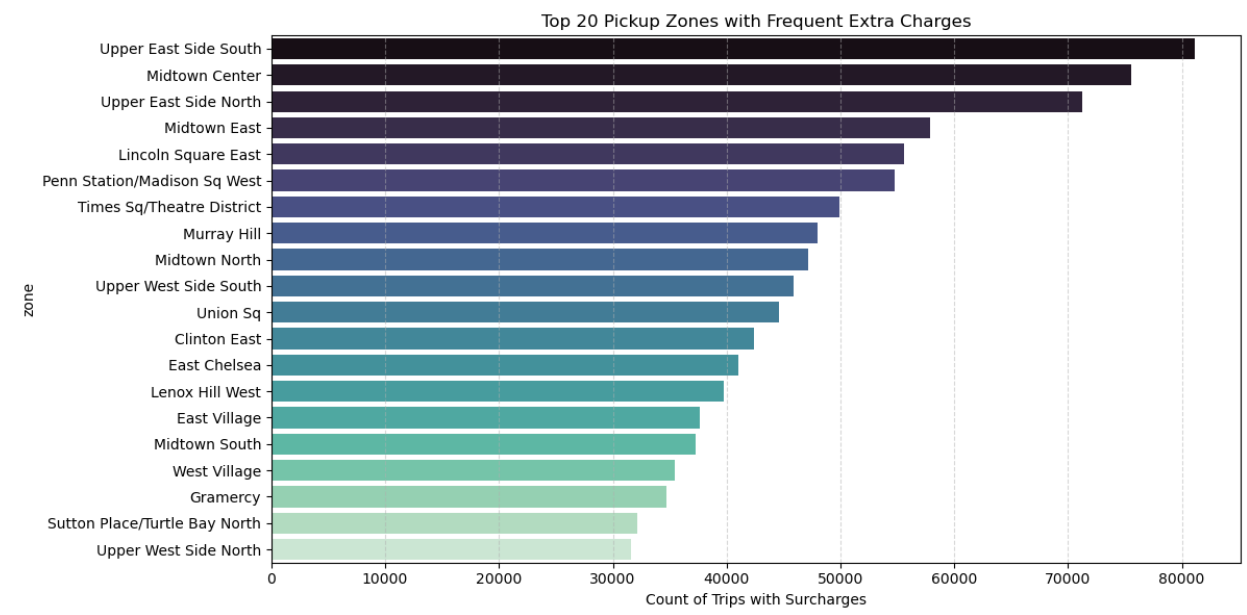
## 3.2.16. Analyse the pickup/dropoff zones or times when extra charges are applied more frequently.

* Find out how often surcharges/extra charges are applied to understand their prevalence





## Zones and its surcharges details – top(10)



## 4. Conclusions

## 4.1 Final Insights and Recommendations

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

* + Target High-Demand Zones: Focus taxi availability in Midtown Center, Midtown, Upper East Side North, and Upper East Side South during peak demand hours.
  + Optimize Evening Coverage: Demand significantly rises between 4 PM and 8 PM (16:00–20:00), making it essential to concentrate dispatch efforts in these zones during this window.
  + Weekday Peak Hour: The 6 PM hour (18:00) stands out as the busiest time on weekdays, indicating a surge in commuter and post-work travel demand.
  + Weekend Stability: Unlike weekdays, weekends show a more evenly distributed demand pattern with no significant peak around 6 PM, suggesting a steady but moderate flow of trips throughout the day
  + Late-Night Hotspot: East Village consistently records the highest pickup and dropoff activity during the night hours (11 PM to 5 AM), highlighting it as a prime zone for late-night taxi availability.

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

* During the morning rush hours (7–10 AM), taxi demand surges as commuters travel from residential areas—like Upper Manhattan and Queens—towards business hubs such as Midtown and Downtown.
* In the evening rush period (4–8 PM), areas like Midtown, the Financial District, and the Upper East and West Sides experience high activity due to office-goers returning home and the early dinner crowd.
* Nighttime hours (9 PM–2 AM) see elevated demand around entertainment zones such as East Village, as well as at major airports, driven by late-night diners, nightlife seekers, and flight arrivals.
* For Midtown, it's essential to maintain a moderate fleet throughout the day, with increased deployment during business hours and special events to handle surges in demand.
* At airports, consistent taxi availability should be ensured during peak flight arrival windows to serve travelers efficiently.
* In residential areas such as Queens and the Bronx, taxis should be strategically dispatched during the morning and early evening hours to meet commuter traffic.
* Entertainment zones benefit from higher cab availability after 9 PM on weekends, catering to short but high-value trips related to nightlife activity.
* Lastly, slow zones—areas prone to congestion and reduced speeds—should generally be avoided during rush hours unless there's a forecasted spike in demand that justifies the inefficiency.

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

* High-demand hours (4–8 PM, weekdays) and late-night slots (9 PM–2 AM, weekends) show peak activity, especially in Midtown and entertainment zones.
  + Increase base fare or per-mile rate slightly during these windows.
* Low-demand hours (10 AM–3 PM, weekdays) can be priced lower to attract more rides.
  + Offer fare discounts or promo pricing to improve utilization.
* Trips of 2–5 miles show high fare-per-mile potential.
  + Introduce a tiered pricing model with a slightly reduced per-mile rate for trips >5 miles to encourage longer bookings while balancing fuel/time costs.
* Extra charges like night surcharges and congestion fees are more common in high-traffic zones and night hours.
  + Keep surcharges transparent, but bundle them smartly like “night fare” includes tolls and extras
  + Use flat-rate surcharges instead of unpredictable ones to maintain customer trust.