Report: Optimising NYC Taxi Operations

This report presents an Exploratory Data Analysis of 2023 New York City Yellow Taxi trip records. The goal of the exercise is to use the 2023 taxi trip data to uncover insights that can inform strategic decisions to improve service efficiency, maximise revenue, and enhance passenger experiences, that could help optimize taxi operations by analyzing patterns in the data.

1. Data Preparation

1.1. Loading the dataset

1.1.1. Sample the data and combine the files

The dataset consists of data collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers like vendors and taxi hailing apps (https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page). The dataset used for the analysis consists of 12 files in parquet format, each for one month of data from the year 2023. Each file contains approximately 3 million records.

Combining the data for all twelve months would result in a dataframe of over 36 million records. Using this large dataframe for the analysis is not computationally feasible. One way to overcome this problem, is to take a small percentage of entries for pickup in every hour of a date. Using the column *`tpep_pickup_datetime`* for this, date and hour from the datetime values are separated, and then for each date, and each hour of each date, 5% of the records are sampled at random.

This sampled data will contain 5% values selected at random from each hour of each date. The sampled data from each file is concatenated into a single dataframe and stored separately onto a new parquet file. Reading this new parquet file into a new dataframe shows that it contains **1896420** records, which is approximately **5%** of the total number of records. This ensures the dataset remains computationally manageable.

```
# iterate through the list of files and sample one by one:
for file_name in file_list:
    try:
        # file path for the current file
        file_path = os.path.join(os.getcwd(), file_name)

        print(f"Processing file: {file_name}")
        # Reading the current file
        monthly_df = pd.read_parquet(file_path)
        print(f"Shape of {file_name} dataframe:", monthly_df.shape)
        df_rows = monthly_df.shape[0]
        total_rows += df_rows

        monthly_df['pickup_date'] = monthly_df['tpep_pickup_datetime'].dt.day
        monthly_df['pickup_hour'] = monthly_df['tpep_pickup_datetime'].dt.hour
        # We will store the sampled data for the current date in this df by appending the sampled data
        from each hour to this
        # After completing iteration through each date, we will append this data to the final dataframe.
        sampled_data = pd.DataFrame()

        # Loop through dates and then loop through every hour of each date
        for date in range(1, 32):  # Assuming maximum 31 days in a month
            day_data = monthly_df[monthly_df['pickup_date'] == date]
            # Iterate through each hour of the selected date
            for hour in range(0, 24):
                hourly_data = day_data[monthly_df['pickup_hour'] == hour]
                # Sample 5% of the hourly data randomly
                sampled_hourly_data = hourly_data.sample(frac=0.05, random_state=42)
                # add data of this hour to the dataframe
                sampled_data = pd.concat([sampled_data, sampled_hourly_data], ignore_index=True)

        # Concatenate the sampled data of all the dates to a single dataframe
        df = pd.concat([df, sampled_data], ignore_index=True)

        print(f"Processed file: {file_name}\n")
```

2. Data Cleaning

2.1. Fixing Columns

2.1.1. Fix the index

The sampling process randomly picks out records from the source, which includes the index, which results in the dataset not having an ordered index. Resetting the index re-orders the records with a new index, starting from 0.

```
# Fix the index and drop any columns that are not needed
df.reset_index(drop=True, inplace=True)
df.drop(columns=['store_and_fwd_flag'], inplace=True)
df.head()
```

2.1.2. Combine the two airport_fee columns

There are two airport fee columns. This is possibly an error in naming columns. These can be combined into a single column, and the extra column can be dropped from the dataframe.

```
# Combine the two airport fee columns
df['airport_fee'] = df['airport_fee'].combine_first(df['Airport_fee'])
df.drop(columns=['Airport_fee'], inplace=True)
```

2.2. Handling Missing Values

2.2.1. Find the proportion of missing values in each column

Missing data may be present in the dataset due to data entry errors, mismatched data-types, etc. Analysis shows that a few columns are missing 3.42% of the values.

```
# Find the proportion of missing values in each column
df.isna().mean()*100
```

✓ 0.0s Python

```
VendorID                0.00
tpep_pickup_datetime    0.00
tpep_dropoff_datetime   0.00
passenger_count         3.42
trip_distance           0.00
RatecodeID              3.42
PULocationID            0.00
DOLocationID            0.00
payment_type            0.00
fare_amount             0.00
extra                   0.00
mta_tax                 0.00
tip_amount              0.00
tolls_amount            0.00
improvement_surcharge   0.00
total_amount            0.00
congestion_surcharge    3.42
airport_fee             3.42
pickup_date             0.00
pickup_hour             0.00
dtype: float64
```

2.2.2. Handling missing values in passenger_count

`passenger_count` column shows 3.42% missing data from the previous analysis. These missing values can be imputed using the most occurring value in the column, or by assuming that the driver may have forgotten to enter the value as it was just one passenger.

```
# Display the rows with null values
# Impute NaN values in 'passenger_count'
print(df['passenger_count'].value_counts(dropna=False))

df['passenger_count'] = df['passenger_count'].fillna(df['passenger_count'].mode()[0])
```

0.0s Python

2.2.3. Handle missing values in RatecodeID

Missing values in the *`RatecodeID`* column can be imputed using the most occurring value in the column, which would be the Standard Rate.

```
# Fix missing values in 'RatecodeID'
print(df['RatecodeID'].value_counts(dropna=False))

df['RatecodeID'] = df['RatecodeID'].fillna(df['RatecodeID'].mode()[0])
```

0.0s Python

2.2.4. Impute NaN in congestion_surcharge

Missing values in *`congestion_surcharge`* column can be assumed to be $0.0, assuming there is no charge applied.

```
# handle null values in congestion_surcharge
print(df['congestion_surcharge'].value_counts(dropna=False))

df['congestion_surcharge'].fillna(0, inplace=True)
```

0.0s Python

2.3. Handling Outliers and Standardising Values

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

Performing an outlier analysis by describing the dataframe shows if there are any potential outliers present. In a general sense, there are potential outliers in columns where the mean and median are further apart.

```python
# Describe the data and check if there are any potential outliers present
# Check for potential out of place values in various columns
df.describe().transpose()
```

✓ 1.3s Python

	count	mean	min	25%	50%	75%	max	std
VendorID	1896420.00	1.73	1.00	1.00	2.00	2.00	6.00	0.45
tpep_pickup_datetime	1896420	2023-07-02 19:32:35.870896	2001-01-01 00:06:49	2023-04-02 16:09:10.500000	2023-06-27 15:42:34	2023-10-06 19:36:21.750000	2024-01-01 00:01:34	NaN
tpep_dropoff_datetime	1896420	2023-07-02 19:50:01.577445	2001-01-01 15:42:11	2023-04-02 16:27:35.500000	2023-06-27 15:59:43.500000	2023-10-06 19:53:03.250000	2024-01-01 20:50:55	NaN
passenger_count	1896420.00	1.37	1.00	1.00	1.00	1.00	9.00	0.86
trip_distance	1896420.00	3.84	0.00	1.05	1.79	3.40	126360.46	127.19
RatecodeID	1896420.00	1.61	1.00	1.00	1.00	1.00	99.00	7.27
PULocationID	1896420.00	165.27	1.00	132.00	162.00	234.00	265.00	64.00
DOLocationID	1896420.00	164.04	1.00	114.00	162.00	234.00	265.00	69.80
payment_type	1896420.00	1.16	0.00	1.00	1.00	1.00	4.00	0.51
fare_amount	1896420.00	19.92	0.00	9.30	13.50	21.90	143163.45	105.54
extra	1896420.00	1.59	0.00	0.00	1.00	2.50	20.80	1.83
mta_tax	1896420.00	0.50	0.00	0.50	0.50	0.50	4.00	0.05
tip_amount	1896420.00	3.55	0.00	1.00	2.85	4.42	223.08	4.06
tolls_amount	1896420.00	0.60	0.00	0.00	0.00	0.00	143.00	2.19
improvement_surcharge	1896420.00	1.00	0.00	1.00	1.00	1.00	1.00	0.03
total_amount	1896420.00	28.98	0.00	15.96	21.00	30.94	143167.45	106.42
congestion_surcharge	1896420.00	2.23	0.00	2.50	2.50	2.50	2.50	0.78
airport_fee	1896420.00	0.14	0.00	0.00	0.00	0.00	1.75	0.46
pickup_date	1896420.00	15.52	1.00	8.00	15.00	23.00	31.00	8.70
pickup_hour	1896420.00	14.27	0.00	11.00	15.00	19.00	23.00	5.81

Based on the above analysis, it seems that some of the outliers are present due to errors in registering the trips.

- Entries with *`passenger_count`* more than 6.
- Entries where *`trip_distance`* is nearly 0 and *`fare_amount`* is more than 300
- Entries where *`trip_distance`* and *`fare_amount`* are 0 but the pickup and dropoff zones are different (both distance and fare should not be zero for different zones)
- Entries where *`trip_distance`* is more than 250 miles.
- Entries where *`payment_type`* is 0 (there is no *`payment_type`* 0 defined in the data dictionary).
- Entries where `RatecodeID` is 99 (there is no RatecodeID 99 defined in the data dictionary).

```python
# remove passenger_count > 6
df = df[df['passenger_count']<=6]
```

✓ 0.1s Python

```python
# Continue with outlier handling

# Entries where `trip_distance` is nearly 0 and `fare_amount` is more than 300
df = df[~((df['fare_amount']>300)&(df['trip_distance']<1))]

# Entries where `trip_distance` and `fare_amount` are 0 but the pickup and dropoff zones are different (both
distance and fare should not be zero for different zones)
df = df[~((df['fare_amount']==0)&(df['trip_distance']==0)&(df['PULocationID']!=df['DOLocationID']))]

# Entries where `trip_distance` is more than 250  miles.
df = df[~(df['trip_distance']>250)]

# Entries where `payment_type` is 0 (there is no payment_type 0 defined in the data dictionary)
df = df[df['payment_type']!=0]

# Entries where `RatecodeID` is 99 (there is no RatecodeID 99 defined in the data dictionary)
df = df[df['RatecodeID']!=99]
```

✓ 0.8s Python

3. Exploratory Data Analysis

3.1. General EDA: Finding Patterns and Trends

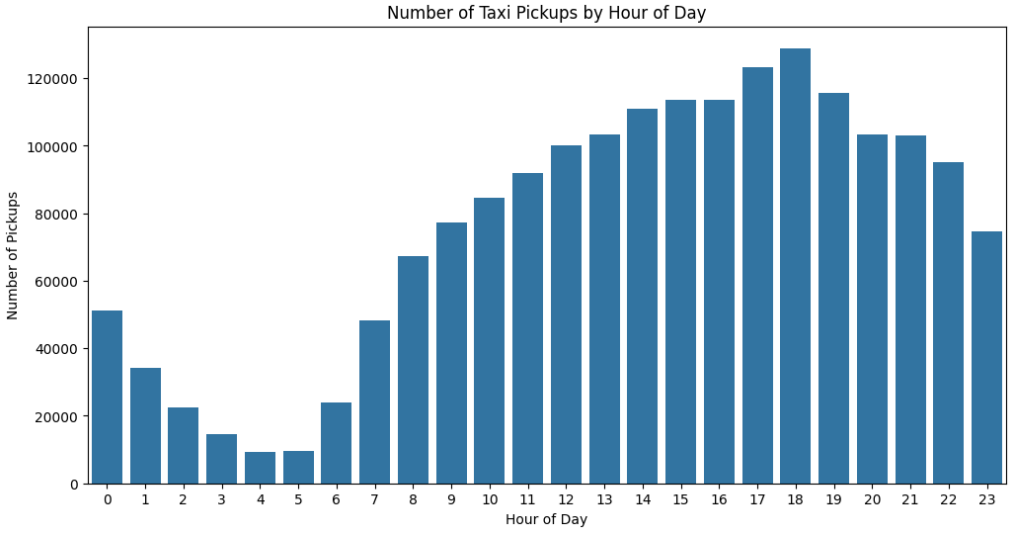
3.1.1. Classify variables into categorical and numerical

Categorical Columns – 'VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime', 'passenger_count', 'trip_distance', 'RatecodeID', 'PULocationID', 'DOLocationID', 'payment_type'

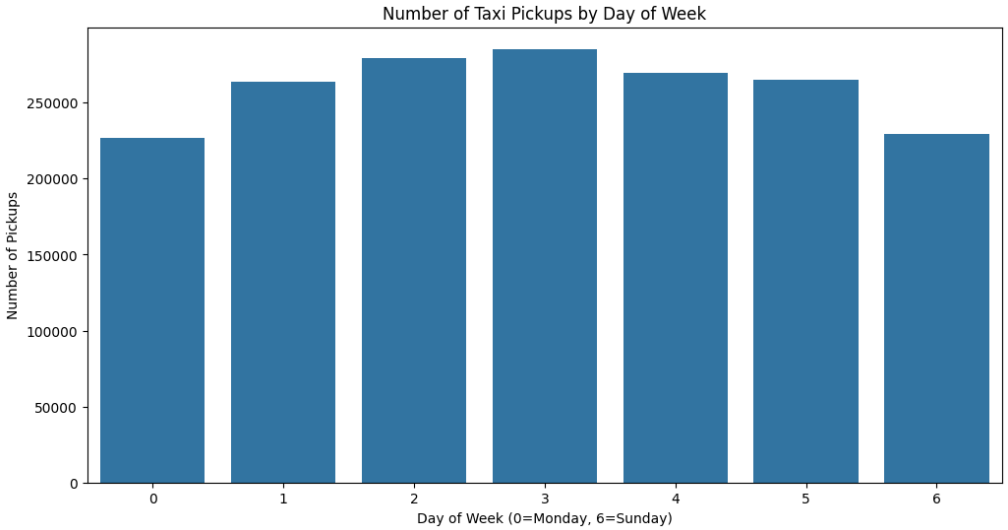
Numerical Columns – 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge', 'total_amount', 'congestion_surcharge', 'airport_fee'

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

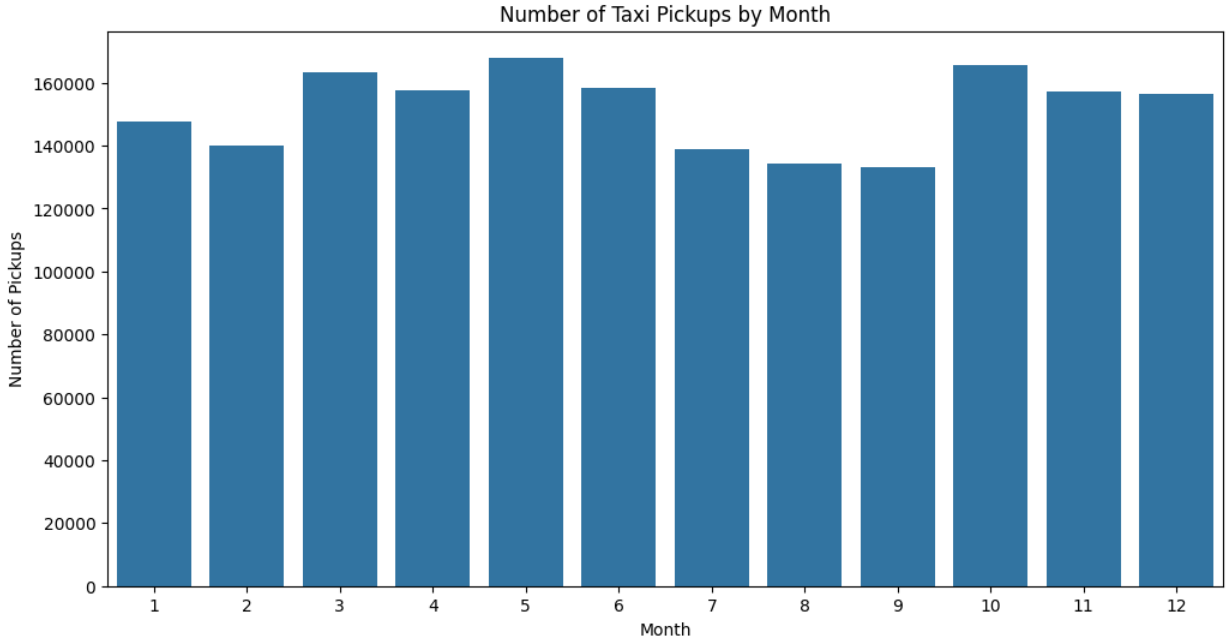
Hourly Trends: The demand curve shows a peak starting during morning commute (8:00 AM) and peaking during the evening rush (**18:00 – 19:00**). The lowest activity is consistently observed between 3:00 AM and 5:00 AM.



Weekly Trends: Wednesdays and Thursdays are the busiest days, and Sundays show the lowest trip volume.

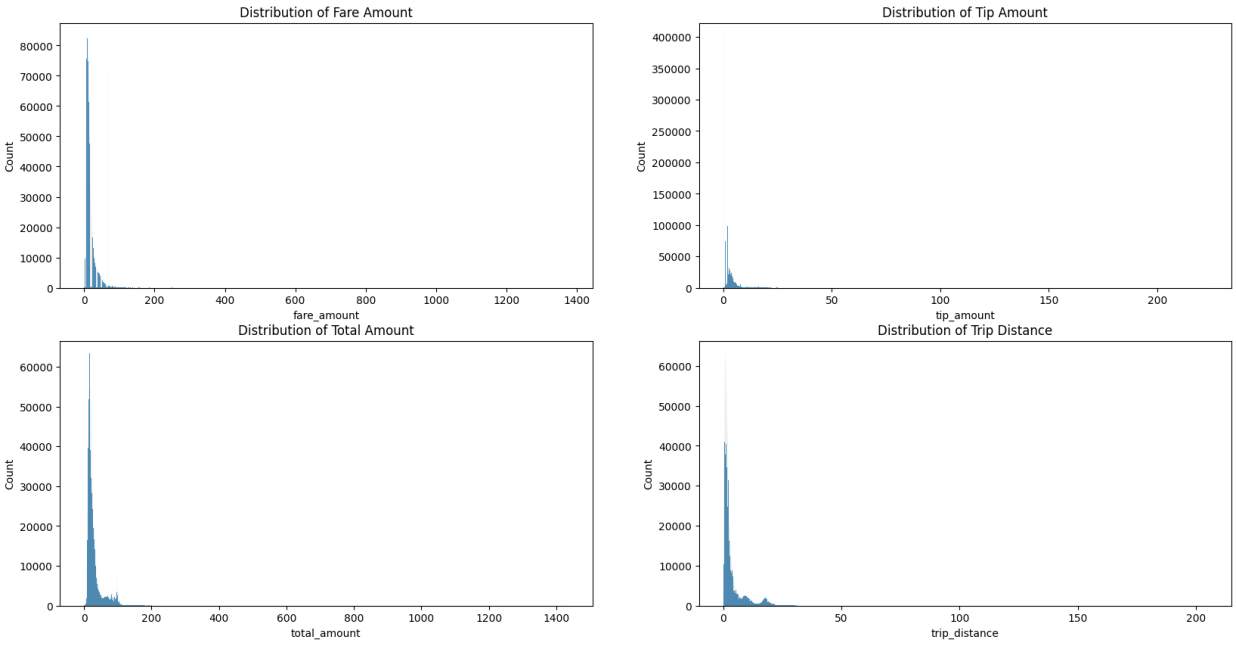


Monthly Trends: Activity dips slightly during **January and February** before rising again, and falls again in the months of **July, August and September** and rises in the fourth quarter.



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

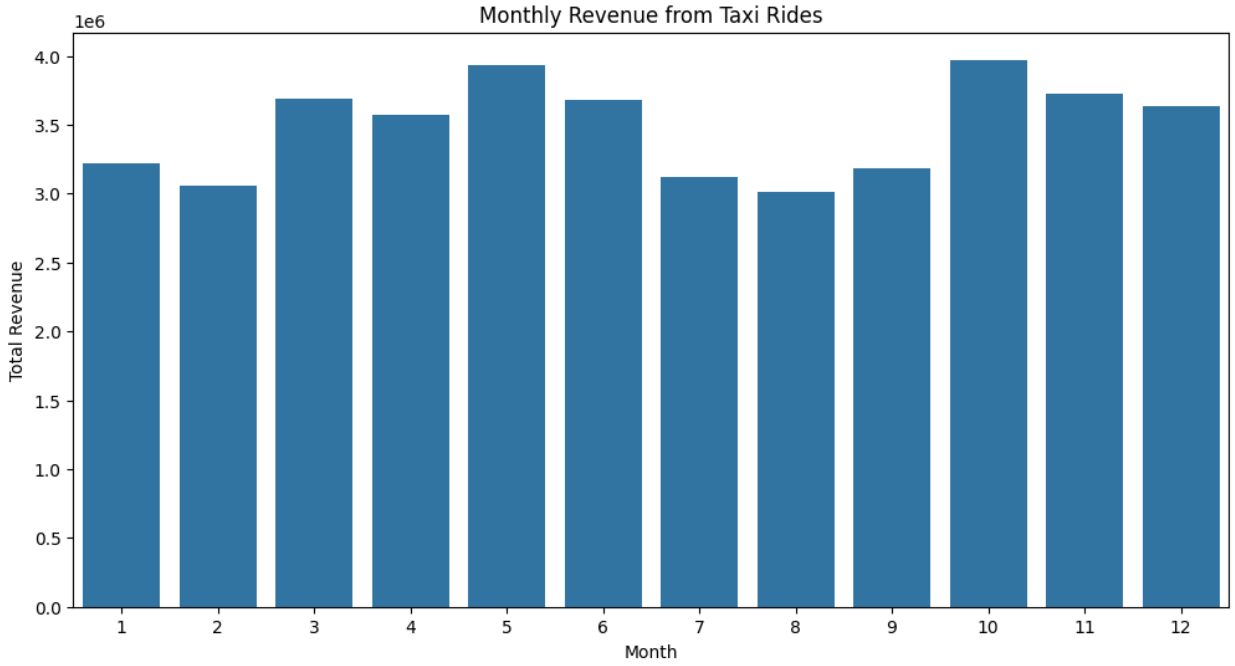
Viewing the distribution of records of columns - `fare_amount`, `tip_amount`, `total_amount`, and `trip_distance` shows a large concentration of records with 0 values.



A new dataframe is created, excluding the zero values from the above columns to draw more accurate analyses from the data.

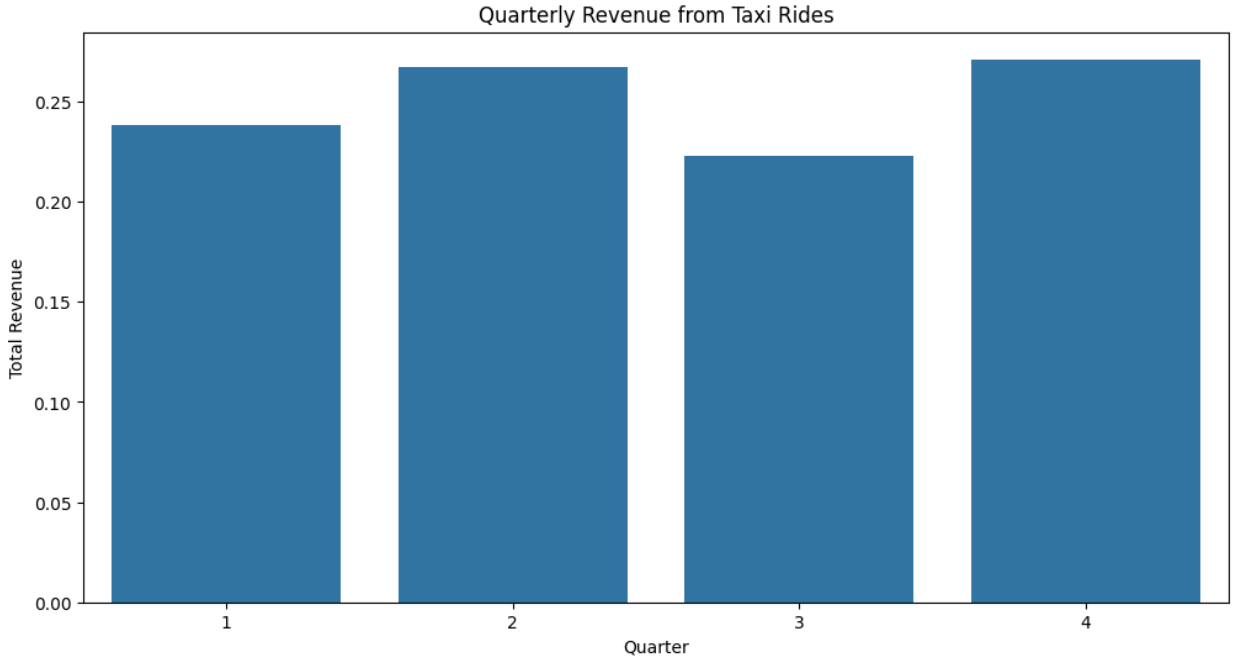
3.1.4. Analyse the monthly revenue trends

Upon plotting the total month-wise revenue, it is observed that there is a high revenue in the months of May and October. The revenue appears to drop in the 3rd Quarter.



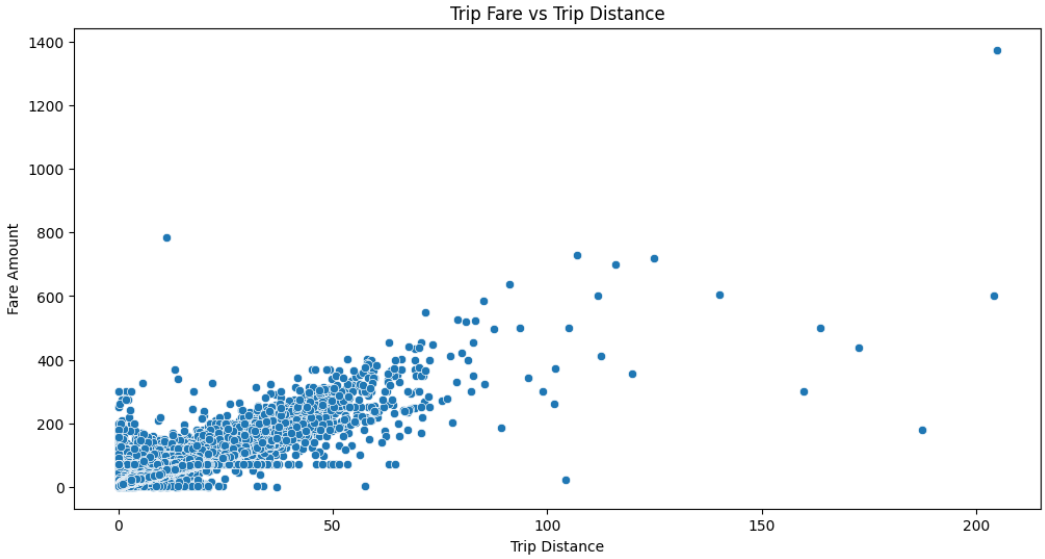
3.1.5. Find the proportion of each quarter's revenue in the yearly revenue

The second and fourth quarters show periods of high revenue. A lower revenue is observed in the third quarter.



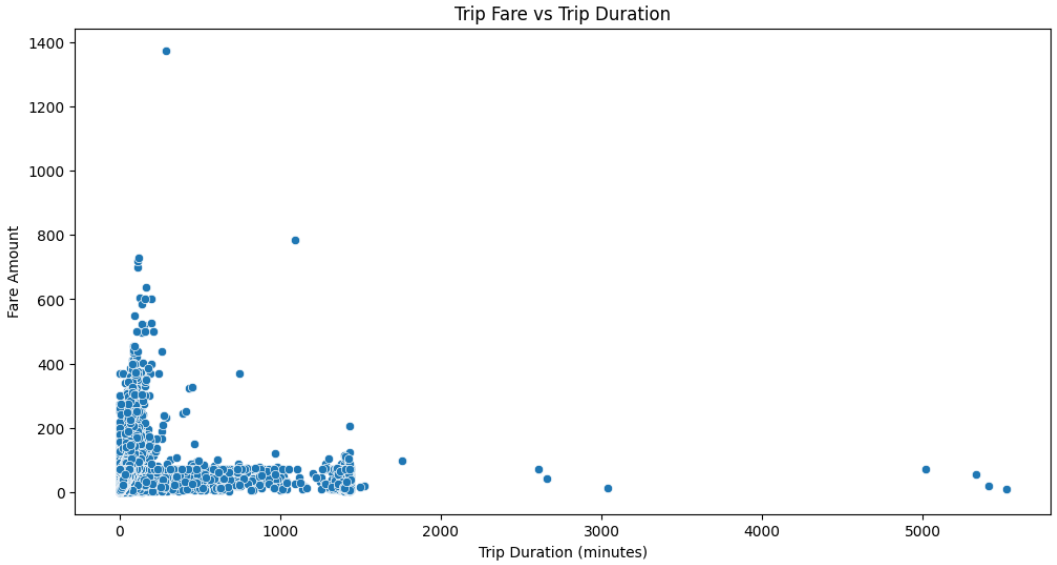
3.1.6. Analyse and visualise the relationship between distance and fare amount

A strong positive correlation is observed between the trip distance and fare amount. Farther the trip distance, the higher the fare amount.



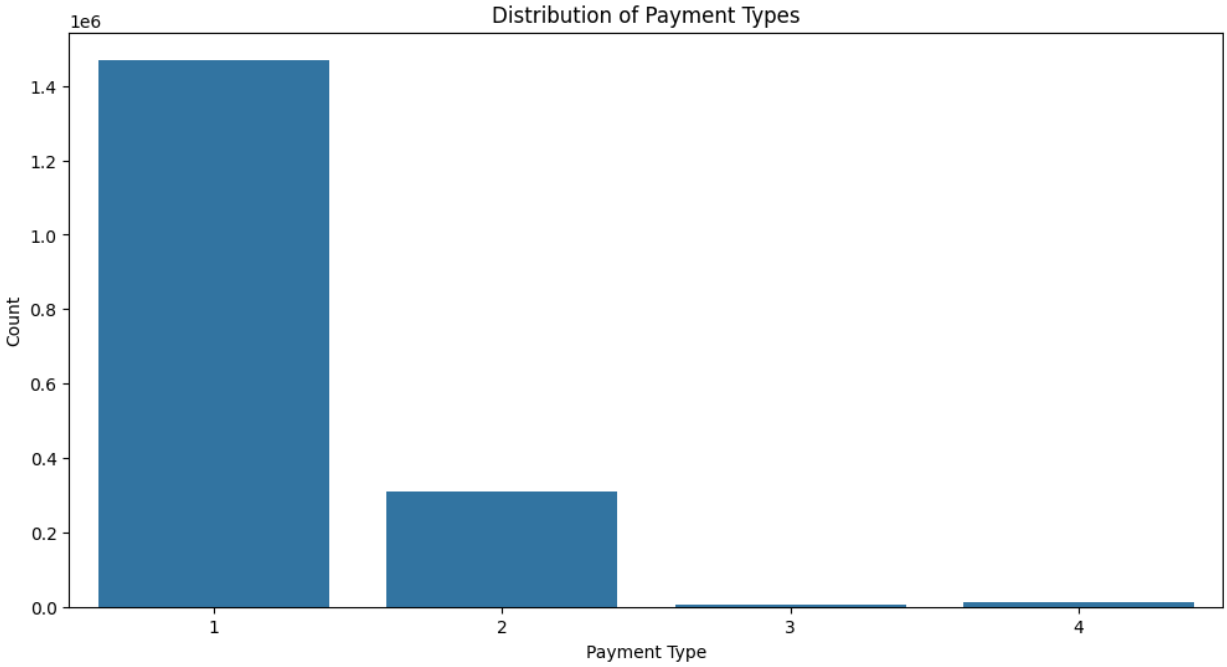
3.1.7. Analyse the relationship between fare/tips and trips/passengers

Trip duration has a weaker correlation with fare, indicating that time spent in traffic does not proportionally increase revenue.

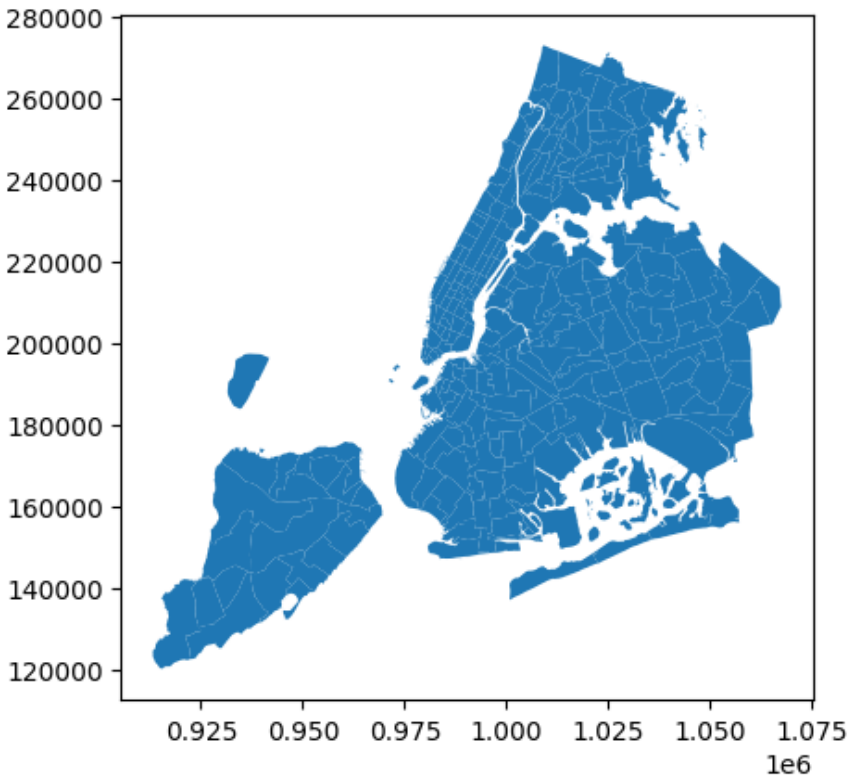


3.1.8. Analyse the distribution of different payment types

Credit Cards appear to be the most preferred mode of payment, followed by cash.



3.1.9. Load the taxi zones shapefile and display it



3.1.10. Merge the zone data with trips data

```
# Merge zones and trip records using locationID and PULocationID
zone_data = nz_df.merge(zones[['LocationID','zone']], left_on='PULocationID', right_on='LocationID',
how='left')
zone_data.rename(columns={'zone': 'pickup_zone'}, inplace=True)
zone_data.drop(columns=['LocationID'], inplace=True)
```

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

JFK Airport, followed by **Upper East Side South** and **Midtown Center** appear to have the highest frequency of trips.

```
# Group data by location and calculate the number of trips
zone_trips = zone_data.groupby('pickup_zone').size().reset_index(name='num_trips').sort_values(by='num_trips',
ascending=False)
zone_trips.head()
```

✓ 0.0s Python

	pickup_zone	num_trips
115	JFK Airport	94859
222	Upper East Side South	86398
148	Midtown Center	85375
221	Upper East Side North	77038
149	Midtown East	65237

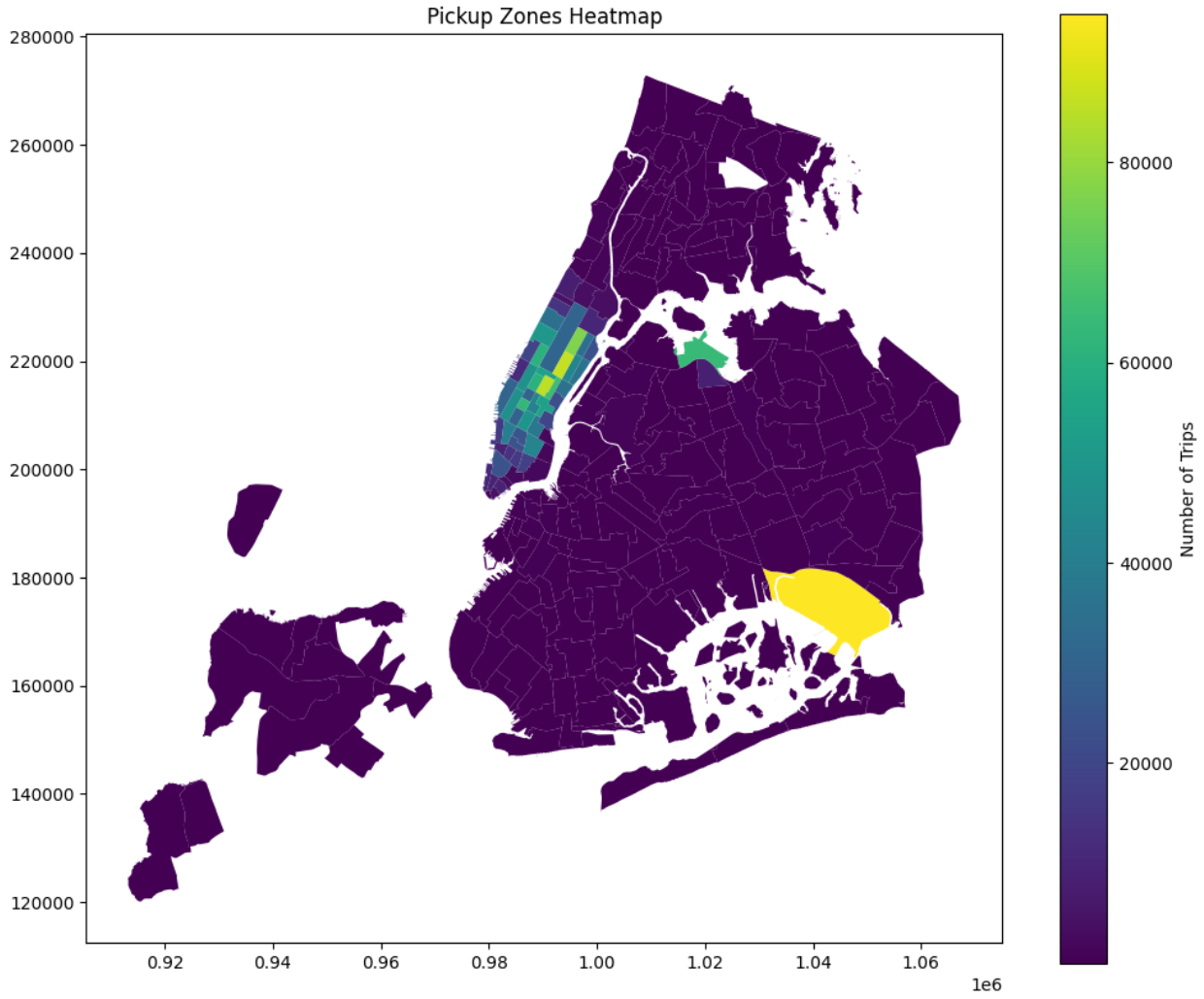
3.1.12. Add the number of trips for each zone to the zones dataframe

```
# can you try displaying the zones DF sorted by the number of trips?
zones_map.sort_values('num_trips', ascending=False).head()
```

✓ 0.0s Python

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry	pickup_zone	num_trips
131	132	0.25	0.00	JFK Airport	132	Queens	MULTIPOLYGON (((1032791.001 181085.006, 103283...	JFK Airport	94859.00
236	237	0.04	0.00	Upper East Side South	237	Manhattan	POLYGON ((993633.442 216961.016, 993507.232 21...	Upper East Side South	86398.00
160	161	0.04	0.00	Midtown Center	161	Manhattan	POLYGON ((991081.026 214453.698, 990952.644 21...	Midtown Center	85375.00
235	236	0.04	0.00	Upper East Side North	236	Manhattan	POLYGON ((995940.048 221122.92, 995812.322 220...	Upper East Side North	77038.00
161	162	0.04	0.00	Midtown East	162	Manhattan	POLYGON ((992224.354 214415.293, 992096.999 21...	Midtown East	65237.00

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



3.1.14. Conclude with results

Busiest hours, days and months –
The demand curve shows a minor peak during morning commute (8:00 AM) and a major peak during the evening rush (18:00 – 19:00). The lowest activity is consistently observed between 3:00 AM and 5:00 AM.
Wednesdays and Thursdays are the busiest days, while Sundays show the lowest trip volume.
Activity dips in January and February and rises to its highest in the fourth quarter.

Trends in revenue collected –
High revenue is observed in the months of May and October. The revenue appears to drop in the third quarter, before rising to the highest in the months of October and November.

Trends in quarterly revenue –
The second and fourth quarters show periods of high revenue. A lower revenue is observed in the third quarter
How fare depends on trip distance, trip duration and passenger counts –
There is a strong positive correlation between trip_distance and fare_amount. However, trip_duration has a weaker correlation with fare, indicating that time spent in traffic does not proportionally increase revenue as effectively as covering distance does.

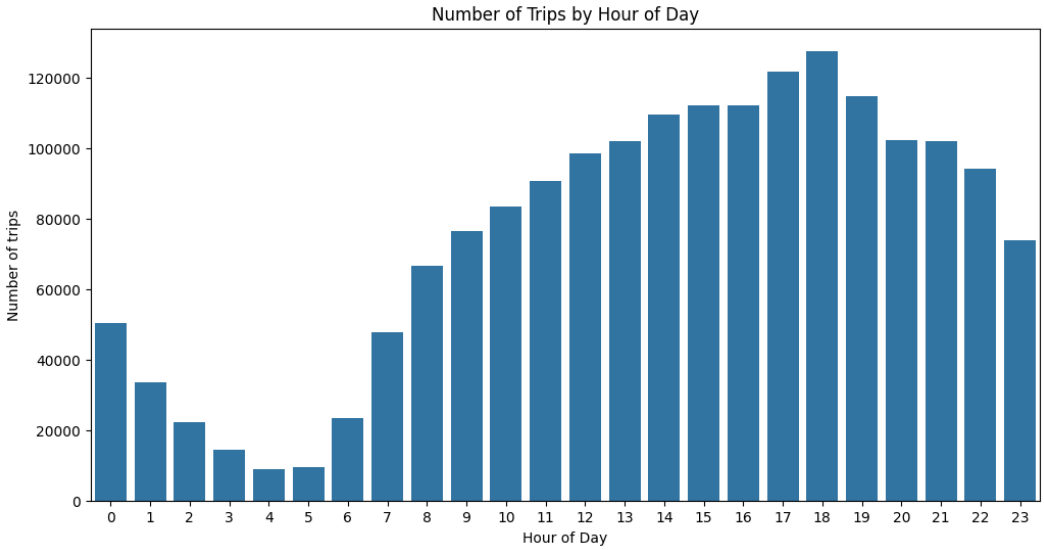
Busiest zones –
JFK Airport, followed by Upper East Side South and Midtown Center appear to have the highest frequency of trips.

3.2. Detailed EDA: Insights and Strategies

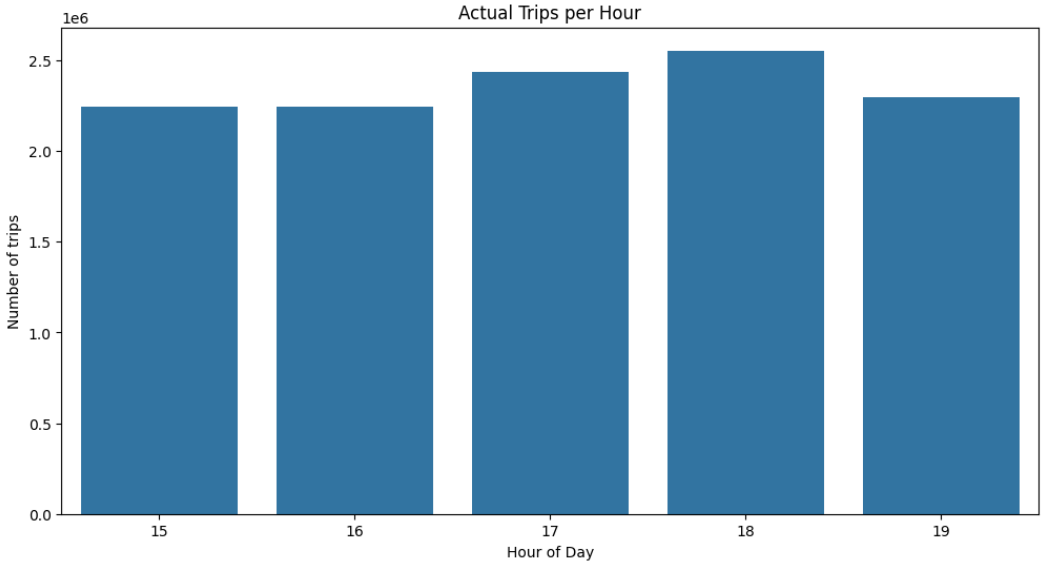
3.2.1. Identify slow routes by comparing average speeds on different routes

	pickup_zone	dropoff_zone	pickup_hour	speed_mph
82440	Randalls Island	Randalls Island	15	0.00
94119	Two Bridges/Seward Park	Downtown Brooklyn/MetroTech	13	0.01
56140	Lincoln Square East	Lincoln Square East	5	0.02
111328	Woodhaven	Woodhaven	1	0.03
29604	Garment District	Astoria	8	0.04

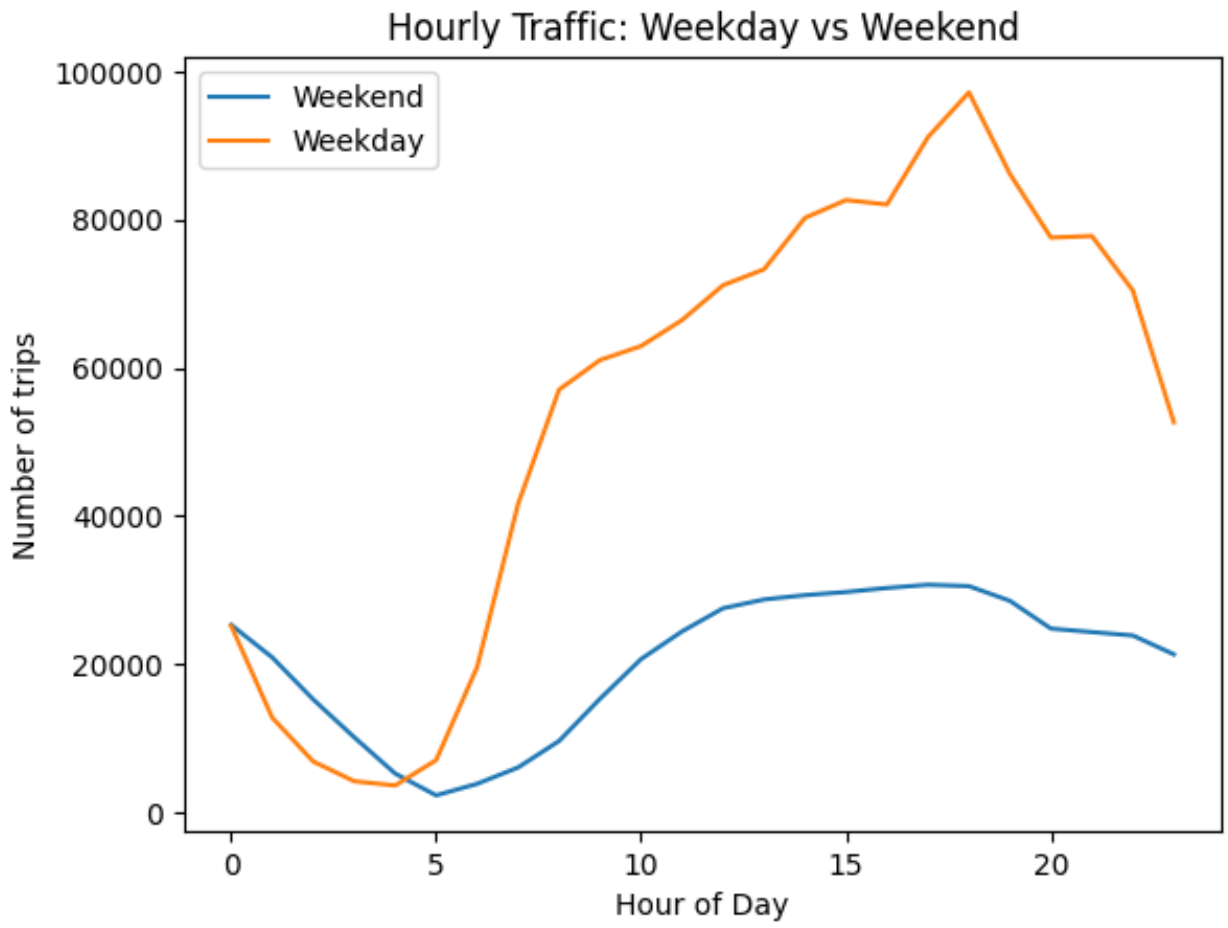
3.2.2. Calculate the hourly number of trips and identify the busy hours



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



3.2.4. Compare hourly traffic on weekdays and weekends



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

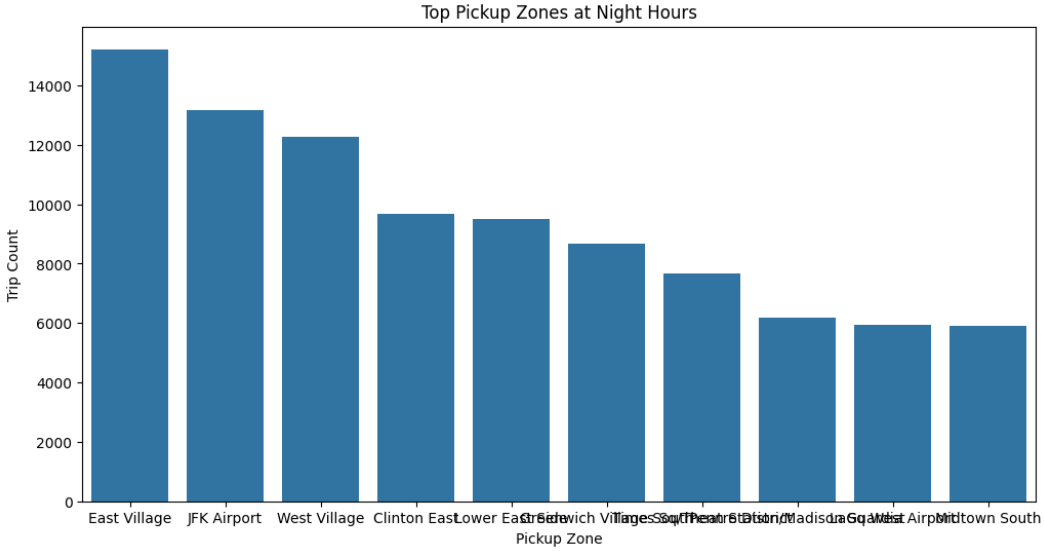
	pickup_hour	pickup_zone	num_trips
2726	18	Midtown Center	7393
2582	17	Midtown Center	7156
2399	16	JFK Airport	6641
3304	22	JFK Airport	6618
2424	16	Midtown Center	6617
2236	15	JFK Airport	6584
2151	14	Upper East Side South	6543
2309	15	Upper East Side South	6541
2775	18	Upper East Side South	6529
2627	17	Upper East Side South	6529

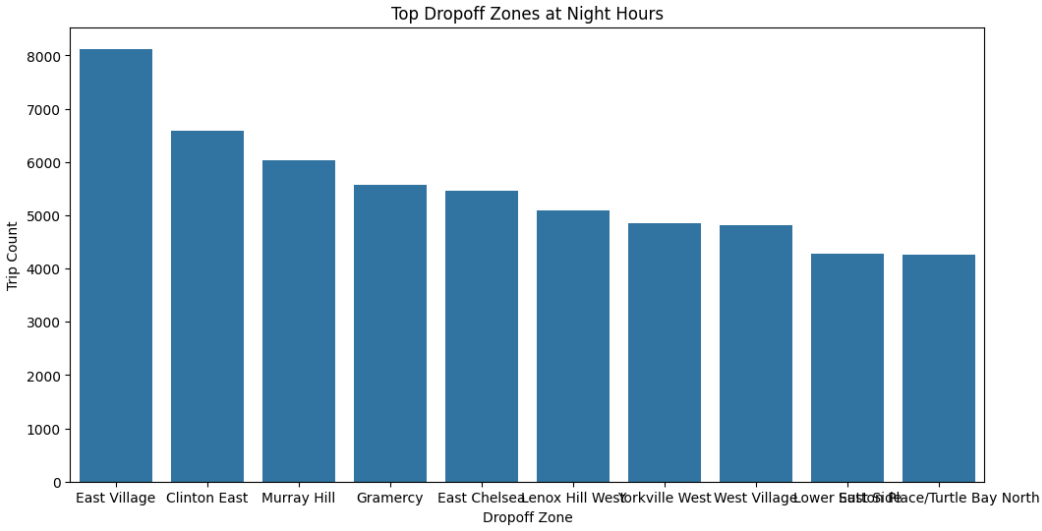
	dropoff_hour	dropoff_zone	num_trips
4454	18	Upper East Side South	6259
3721	15	Upper East Side North	6229
2992	12	Upper East Side South	6074
4453	18	Upper East Side North	6070
3722	15	Upper East Side South	5869
3475	14	Upper East Side North	5837
4208	17	Upper East Side North	5751
3963	16	Upper East Side North	5690
3476	14	Upper East Side South	5626
3237	13	Upper East Side South	5537

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

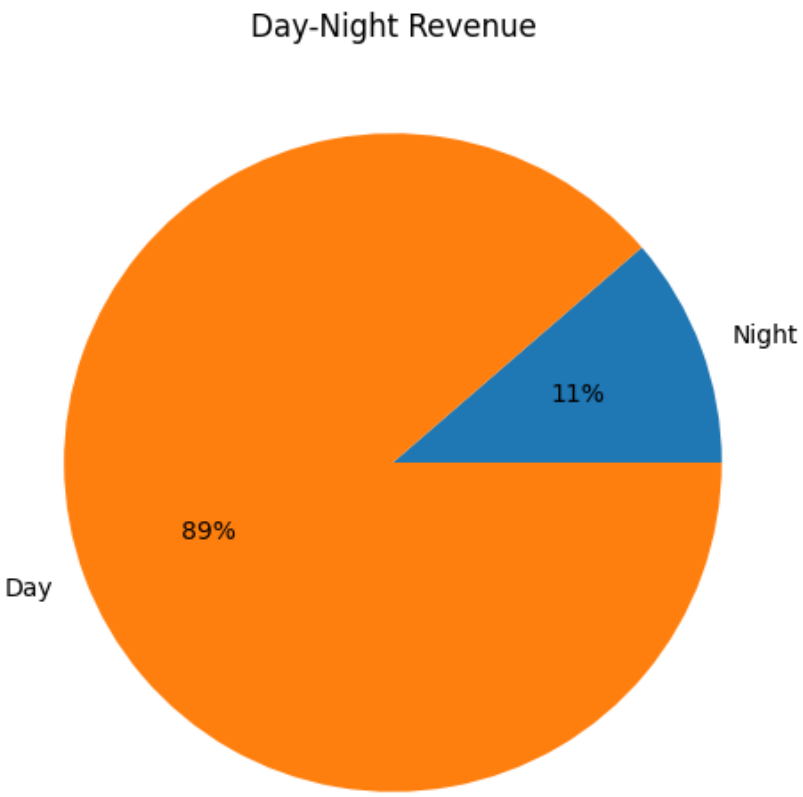
	zone	pickup_trips	dropoff_trips	ratio
63	East Elmhurst	8194	852	9.62
115	JFK Airport	95044	19344	4.91
125	LaGuardia Airport	63866	21816	2.93
173	Penn Station/Madison Sq West	62937	39788	1.58
37	Central Park	30585	22206	1.38
100	Greenwich Village South	23971	17413	1.38
232	West Village	40143	30245	1.33
149	Midtown East	65241	51833	1.26
148	Midtown Center	85383	71015	1.20
91	Garment District	29906	25079	1.19

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

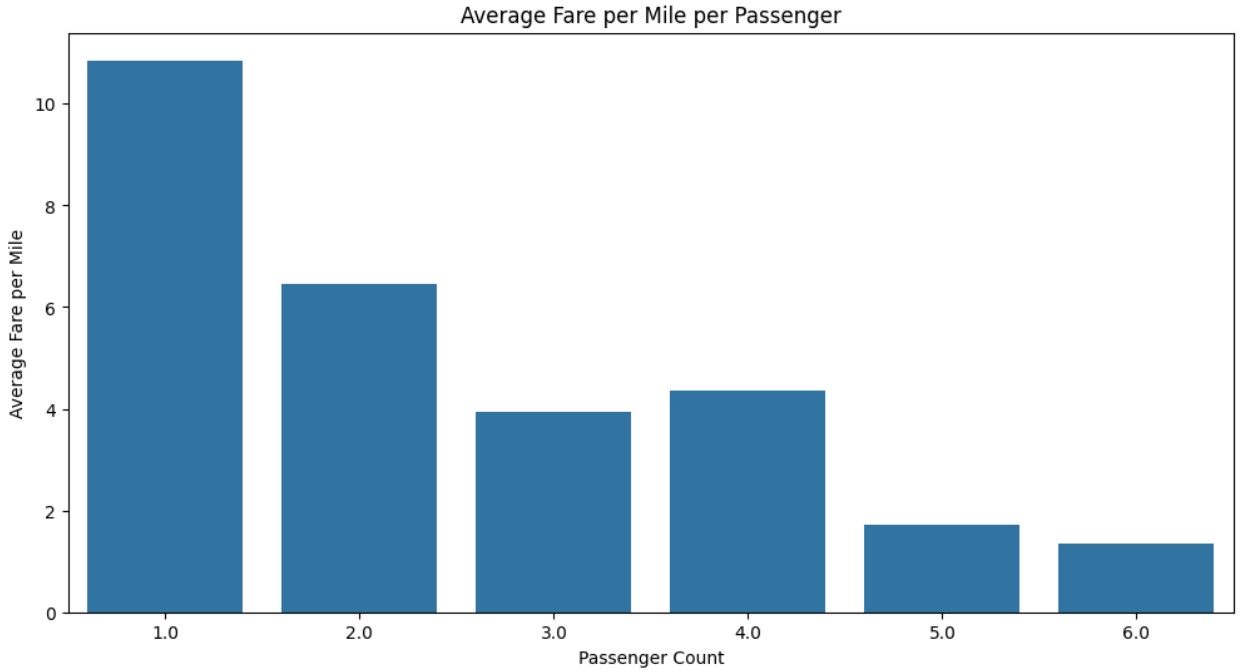




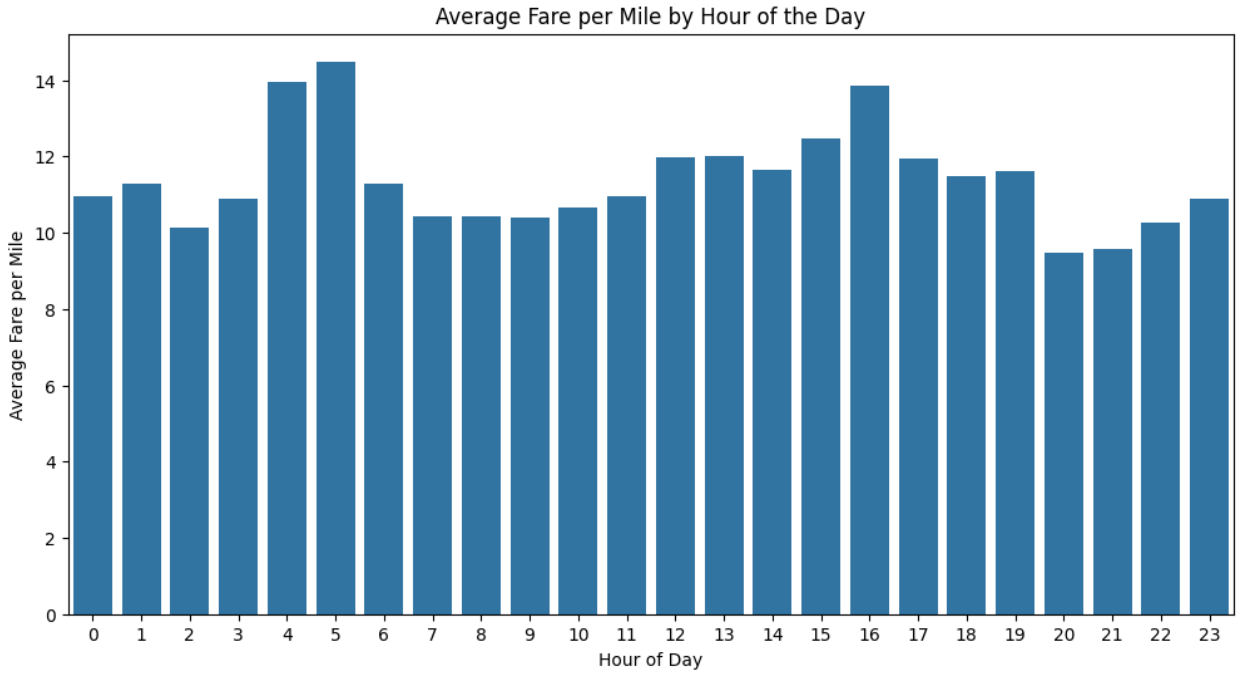
3.2.8. Find the revenue share for nighttime and daytime hours

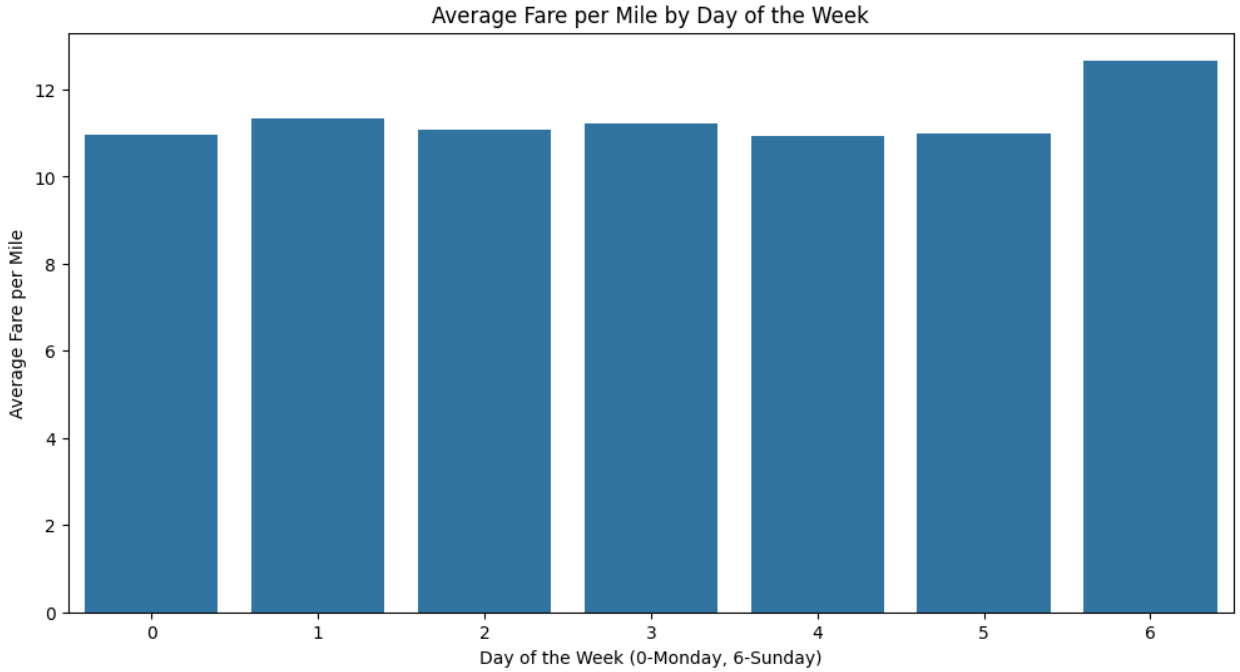


3.2.9. For the different passenger counts, find the average fare per mile per passenger

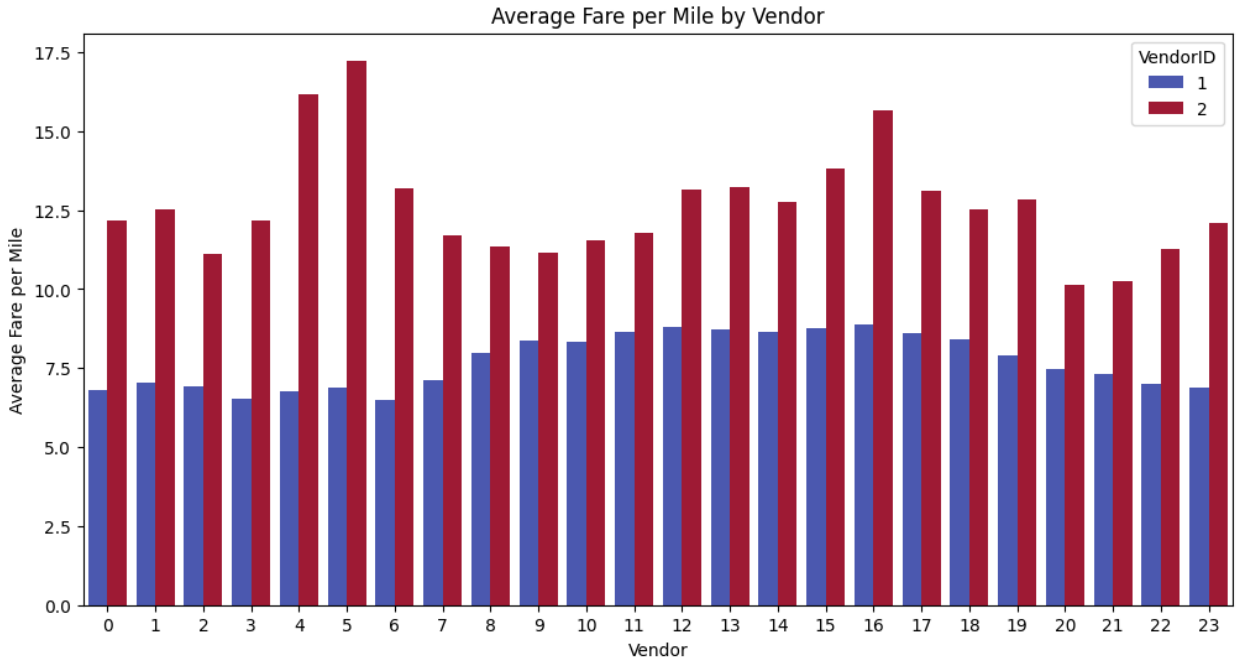


3.2.10. Find the average fare per mile by hours of the day and by days of the week

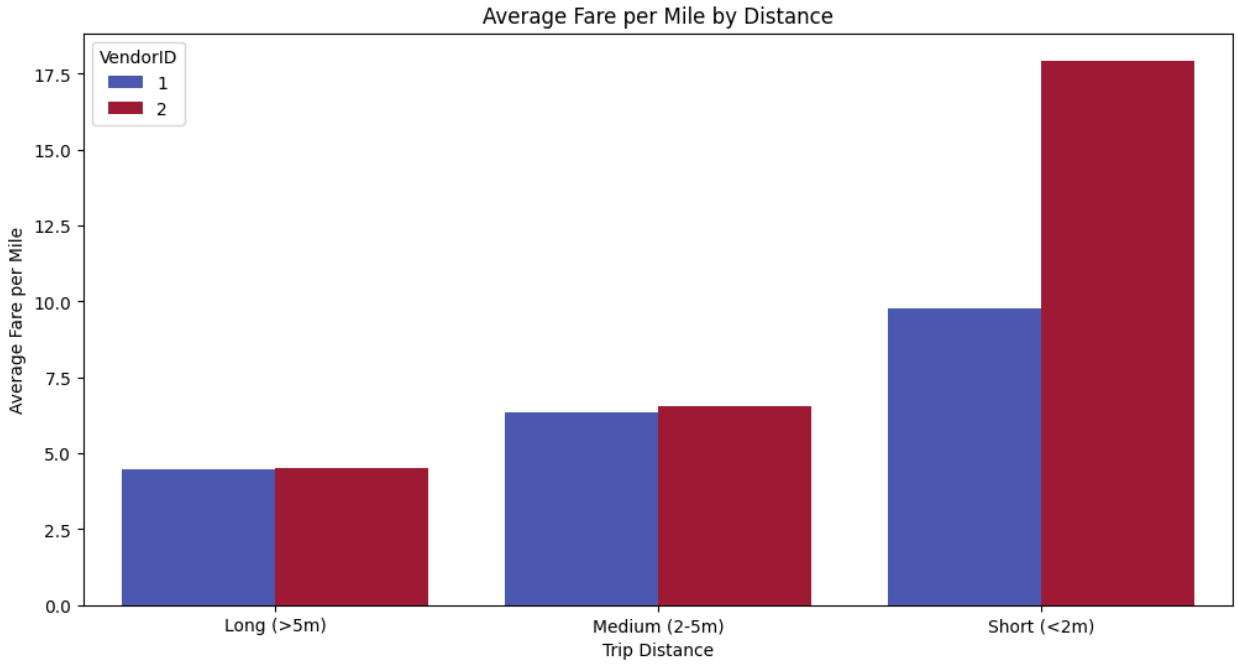




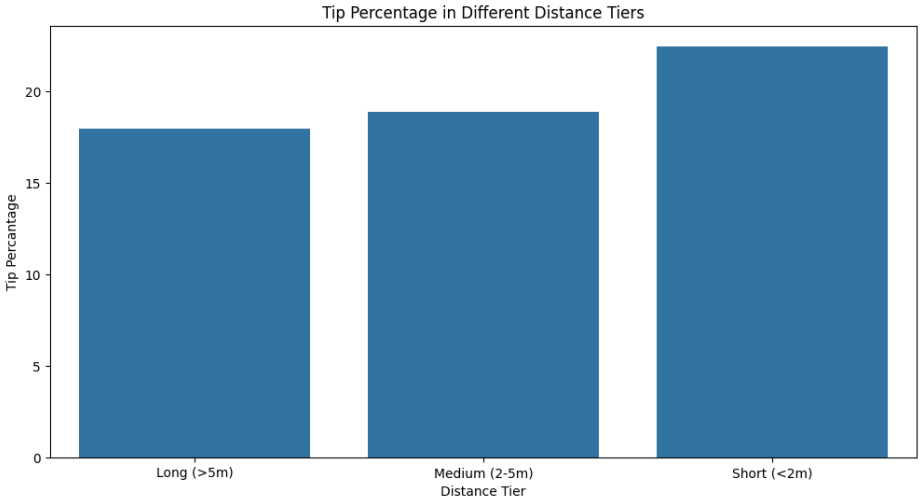
3.2.11. Analyse the average fare per mile for the different vendors



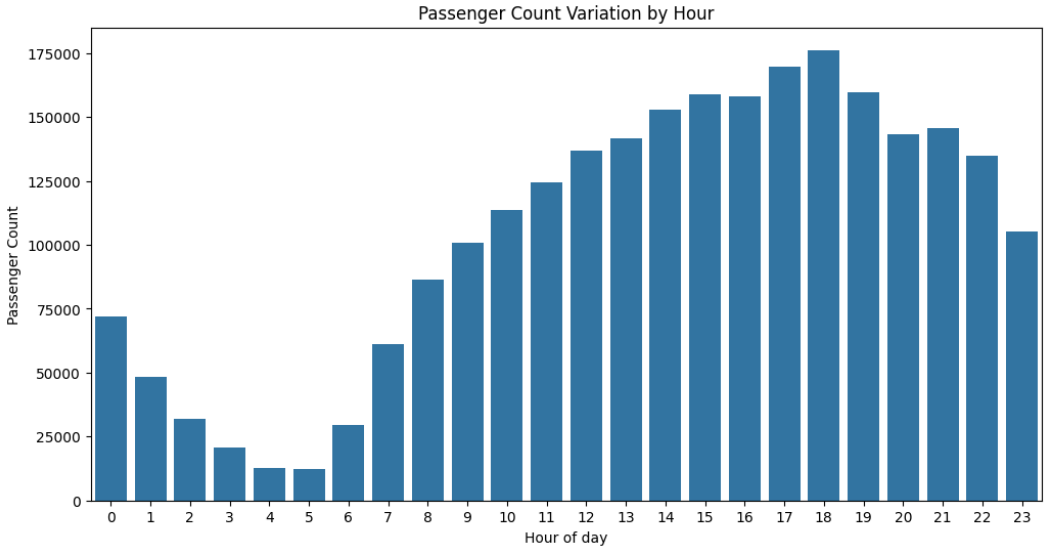
3.2.12. Compare the fare rates of different vendors in a distance-tiered fashion

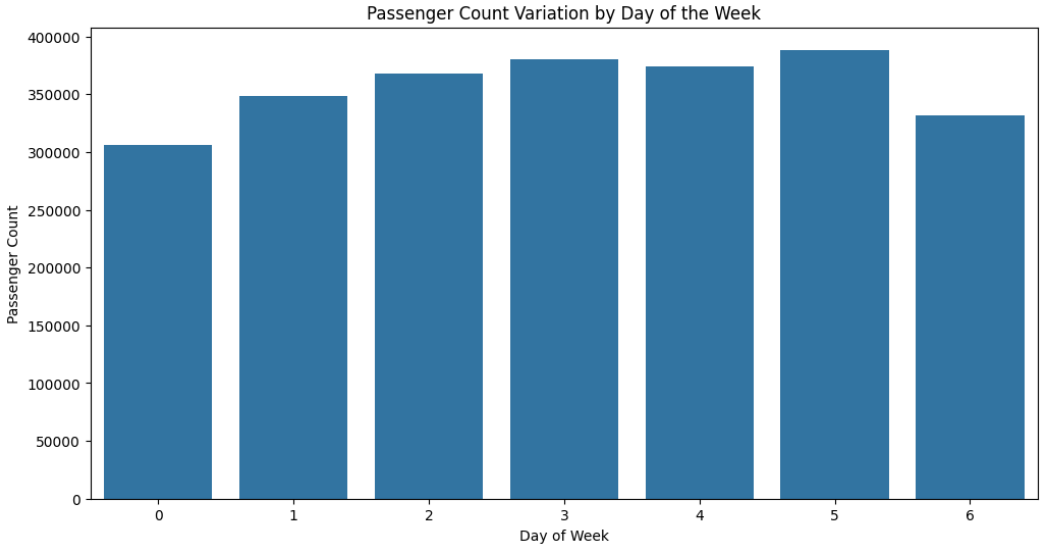


3.2.13. Analyse the tip percentages

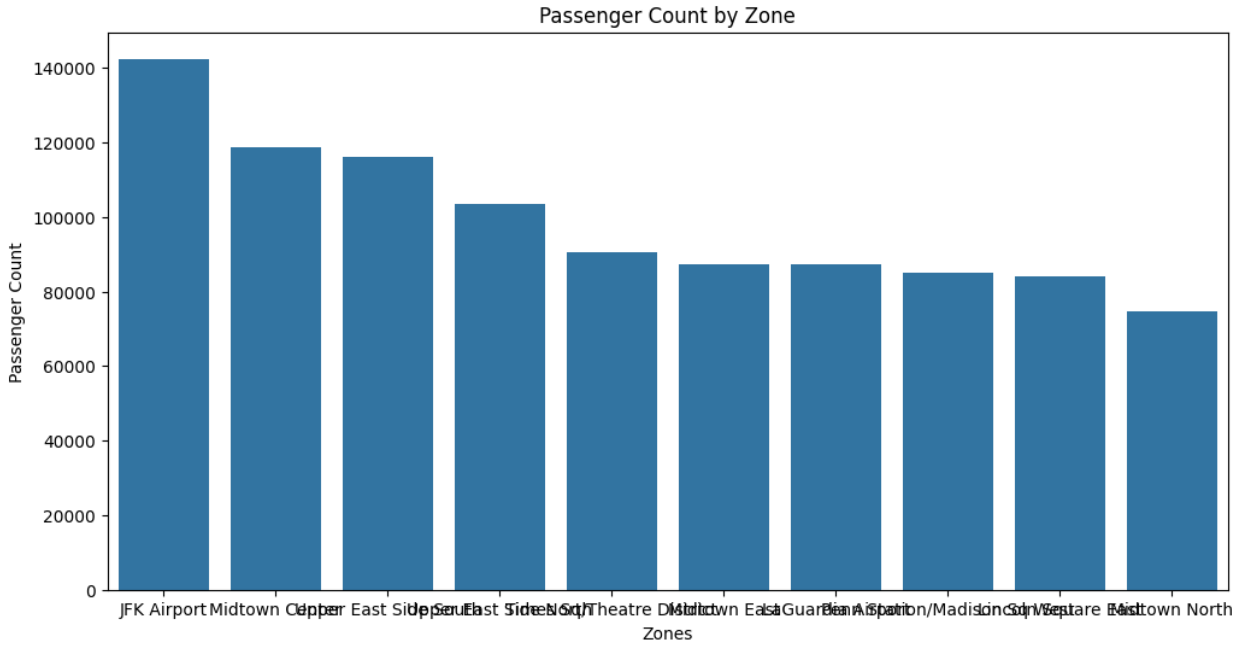


3.2.14. Analyse the trends in passenger count

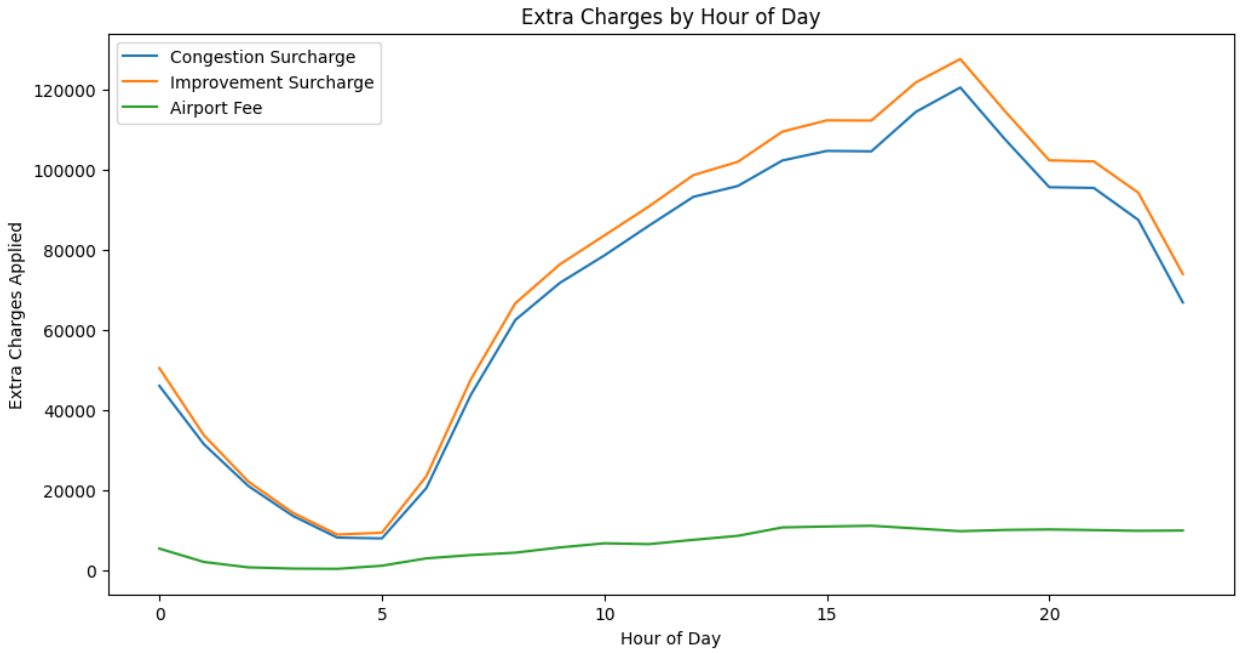




3.2.15. Analyse the variation of passenger counts across zones



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



4. Conclusions

4.1. Final Insights and Recommendations

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

Analysis shows that the hours between 3pm and 7pm, with the number of trips peaking at 6pm. We see high number of pickups from Midtown Center and JFK Airport in the same time frame.

Deploying a higher number of taxis in these areas 30 minutes prior to the evening rush could meet the demand surge. Also, suggesting alternative routes could help mitigate traffic congestions during this period.

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.

JFK Airport and Midtown Center are high-revenue zones and likely have a high wait-time. Maintaining a dedicated fleet at airports on during Thursday evenings and Sunday evenings could mitigate this.

During off-peak weekday mornings, drivers/taxi fleets could be deployed in residential zones, instead of airport queues.

Pickup hotspots shift to late-night on weekends. Fleet concentration can be shifted to entertainment districts like East Village and Clinton East starting 9 PM on Fridays and Saturdays to capture the "night out" crowd.

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

Short trips can be prioritized over long distance during low-demand hours (11 AM to 3PM) as short trips often have higher fare-per-mile yields. Deploying fleets in high-density residential areas like Upper East Side to perform volume-based short trips, accumulate revenue faster than a single long trip.

Cabs can be equipped with a system that suggests default tips when passengers pay with credit cards.