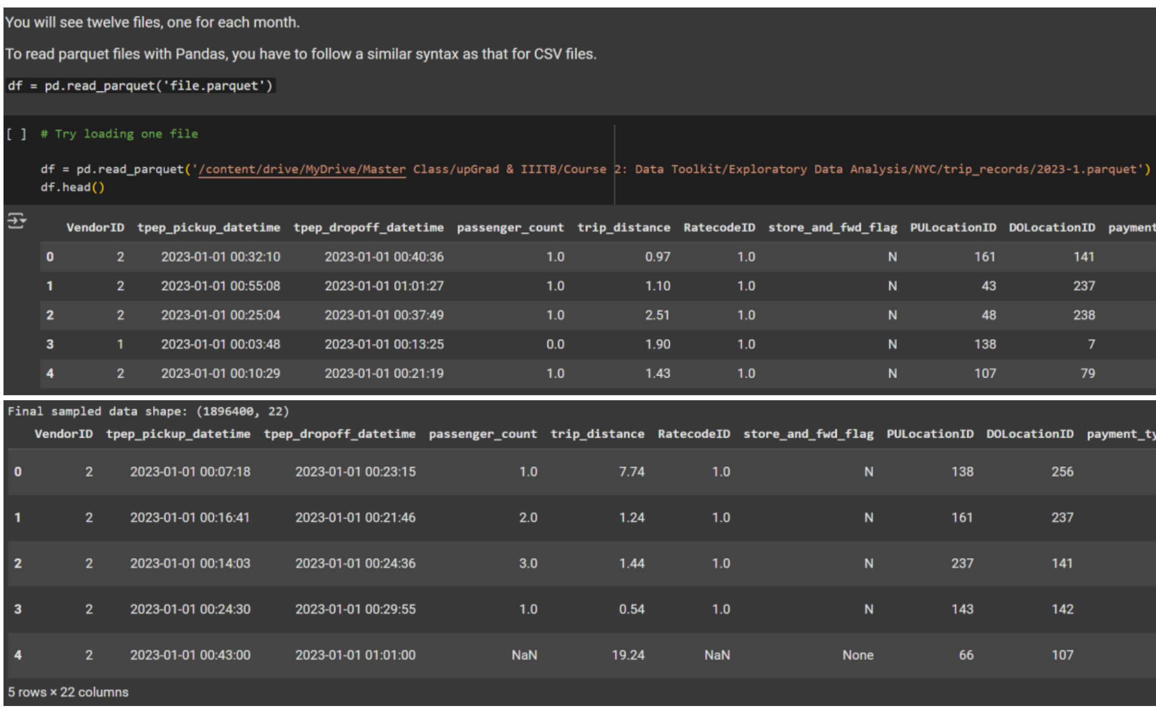
Report: Optimising NYC Taxi Operations

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

- **Sample the data and combine the files**



You will see twelve files, one for each month.

To read parquet files with Pandas, you have to follow a similar syntax as that for CSV files.

```
df = pd.read_parquet('file.parquet')
```

```
[ ] # Try loading one file

    df = pd.read_parquet('/content/drive/MyDrive/Master Class/upGrad & IIITB/Course 2: Data Toolkit/Exploratory Data Analysis/NYC/trip_records/2023-1.parquet')
    df.head()
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment
0	2	2023-01-01 00:32:10	2023-01-01 00:40:36	1.0	0.97	1.0	N	161	141	
1	2	2023-01-01 00:55:08	2023-01-01 01:01:27	1.0	1.10	1.0	N	43	237	
2	2	2023-01-01 00:25:04	2023-01-01 00:37:49	1.0	2.51	1.0	N	48	238	
3	1	2023-01-01 00:03:48	2023-01-01 00:13:25	0.0	1.90	1.0	N	138	7	
4	2	2023-01-01 00:10:29	2023-01-01 00:21:19	1.0	1.43	1.0	N	107	79	

Final sampled data shape: (1896400, 22)

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_ty
0	2	2023-01-01 00:07:18	2023-01-01 00:23:15	1.0	7.74	1.0	N	138	256	
1	2	2023-01-01 00:16:41	2023-01-01 00:21:46	2.0	1.24	1.0	N	161	237	
2	2	2023-01-01 00:14:03	2023-01-01 00:24:36	3.0	1.44	1.0	N	237	141	
3	2	2023-01-01 00:24:30	2023-01-01 00:29:55	1.0	0.54	1.0	N	143	142	
4	2	2023-01-01 00:43:00	2023-01-01 01:01:00	NaN	19.24	NaN	None	66	107	

5 rows × 22 columns

- The dataset consists of multiple files representing different months.
- A small percentage of entries were sampled from each file to create a unified dataset for analysis.
- Data was combined into a single DataFrame for consistency.

2. Data Cleaning

2.1. Fixing Columns

- **Fix the index**

Fix the index and drop unnecessary columns

```
[ ] # Fix the index and drop any columns that are not nee
    # Fix the index
    df = df.reset_index(drop=True)

    # Drop unnecessary columns
    # columns_to_drop = ['VendorID', 'store_and_fwd_flag'
    # df = df.drop(columns=columns_to_drop)
```

- The index was reset and unnecessary columns were removed.

- **Combine the two airport_fee columns**

```
# Combine the two airport fee columns
# Combine the two airport fee columns
df['airport_fee'] = df['airport_fee'].fillna(0) + df['Airport_fee'].fillna
df = df.drop(columns=['Airport_fee'])  # Drop the redundant column
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1896400 entries, 0 to 1896399
Data columns (total 21 columns):
 #   Column                 Dtype
---  ------                 -----
 0   VendorID               int64
 1   tpep_pickup_datetime   datetime64[us]
 2   tpep_dropoff_datetime  datetime64[us]
 3   passenger_count        float64
 4   trip_distance          float64
 5   RatecodeID             float64
 6   store_and_fwd_flag     object
 7   PULocationID           int64
 8   DOLocationID           int64
 9   payment_type           int64
 10  fare_amount            float64
 11  extra                  float64
 12  mta_tax                float64
 13  tip_amount             float64
 14  tolls_amount           float64
 15  improvement_surcharge  float64
 16  total_amount           float64
 17  congestion_surcharge   float64
 18  airport_fee            float64
 19  date                   object
 20  hour                   int32
dtypes: datetime64[us](2), float64(12), int32(1), int64(4), object(2)
memory usage: 296.6+ MB
```

- Duplicate airport fee columns were merged into one.

2.2. Handling Missing Values

- **Find the proportion of missing values in each column**

Find the proportion of missing values in each column

+ Code   + Text

```python
# Find the proportion of missing values in each column
# Find the proportion of missing values in each column
percent_missing = df.isnull().sum() * 100 / len(df)
missing_value_df = pd.DataFrame({'column_name': df.columns,
                                 'percent_missing': percent_miss
print(missing_value_df)
```

```
                                 column_name  percent_missing
VendorID                            VendorID         0.000000
tpep_pickup_datetime    tpep_pickup_datetime         0.000000
tpep_dropoff_datetime  tpep_dropoff_datetime         0.000000
passenger_count              passenger_count         3.420903
trip_distance                  trip_distance         0.000000
RatecodeID                        RatecodeID         3.420903
store_and_fwd_flag        store_and_fwd_flag         3.420903
PULocationID                    PULocationID         0.000000
DOLocationID                    DOLocationID         0.000000
payment_type                    payment_type         0.000000
fare_amount                      fare_amount         0.000000
extra                                  extra         0.000000
mta_tax                              mta_tax         0.000000
tip_amount                        tip_amount         0.000000
tolls_amount                    tolls_amount         0.000000
improvement_surcharge  improvement_surcharge         0.000000
total_amount                    total_amount         0.000000
congestion_surcharge    congestion_surcharge         3.420903
airport_fee                      airport_fee         0.000000
date                                    date         0.000000
hour                                    hour         0.000000
```

Columns with high missing values were identified for imputation or removal

- **Handling missing values in passenger_count**

Handling missing values in `passenger_count`

+ Code + Text

```python
# Display the rows with null values
# Impute NaN values in 'passenger_count'
# Display the rows with null values in passenger_count colum
null_passenger_count_rows = df[df['passenger_count'].isnull(
null_passenger_count_rows.info()

# Impute NaN values in passenger_count column with the media
median_passenger_count = df['passenger_count'].median()
df['passenger_count'].fillna(median_passenger_count, inplace
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 64874 entries, 4 to 1896387
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   VendorID               64874 non-null  int64
 1   tpep_pickup_datetime   64874 non-null  datetime64[us]
 2   tpep_dropoff_datetime  64874 non-null  datetime64[us]
 3   passenger_count        0 non-null      float64
 4   trip_distance          64874 non-null  float64
 5   RatecodeID             0 non-null      float64
 6   store_and_fwd_flag     0 non-null      object
 7   PULocationID           64874 non-null  int64
 8   DOLocationID           64874 non-null  int64
 9   payment_type           64874 non-null  int64
 10  fare_amount            64874 non-null  float64
 11  extra                  64874 non-null  float64
 12  mta_tax                64874 non-null  float64
 13  tip_amount             64874 non-null  float64
 14  tolls_amount           64874 non-null  float64
 15  improvement_surcharge  64874 non-null  float64
 16  total_amount           64874 non-null  float64
 17  congestion_surcharge   0 non-null      float64
 18  airport_fee            64874 non-null  float64
```

Missing values were imputed with the median value.

- **Handle missing values in RatecodeID**

Handle missing values in `RatecodeID`

+ Code + Text

```
# Fix missing values in 'RatecodeID'
df['RatecodeID'].fillna(df['RatecodeID'].mode()[0], inplace=True)
```

```
<ipython-input-45-47fa15ebc76c>:2: FutureWarning: A value is tryin
The behavior will change in pandas 3.0. This inplace method will n

For example, when doing 'df[col].method(value, inplace=True)', try


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

Null values were replaced using the most frequent category.

- **Impute NaN in congestion_surcharge**

Impute NaN in `congestion_surcharge`

```
# handle null values in congestion_surcharge
# handle null values in congestion_surcharge
df['congestion_surcharge'].fillna(df['congestion_surcharge'].median(
```

```
<ipython-input-46-bbda208897ad>:3: FutureWarning: A value is trying
The behavior will change in pandas 3.0. This inplace method will ne

For example, when doing 'df[col].method(value, inplace=True)', try u


  df['congestion_surcharge'].fillna(df['congestion_surcharge'].media
```

Missing values were replaced with the median surcharge.

2.3. Handling Outliers and Standardising Values

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

```python
# Describe the data and check if there are any potential outliers present
# Function to detect outliers using IQR
def detect_outliers(df, column):
    Q1 = df[column].quantile(0.25)  # First quartile (25th percentile)
    Q3 = df[column].quantile(0.75)  # Third quartile (75th percentile)
    IQR = Q3 - Q1  # Interquartile range
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return df[(df[column] < lower_bound) | (df[column] > upper_bound)]

# List of numerical columns to check
num_cols = ["trip_distance", "fare_amount", "tip_amount", "total_amount", "passenger_c

# Detect and count outliers for each column
for col in num_cols:
    outliers = detect_outliers(df, col)
    print(f"Column: {col} → Outliers detected: {len(outliers)}")
```

```
Column: trip_distance → Outliers detected: 249302
Column: fare_amount → Outliers detected: 197413
Column: tip_amount → Outliers detected: 145673
Column: total_amount → Outliers detected: 218083
Column: passenger_count → Outliers detected: 454302
```

- Outliers were detected using statistical methods and visualized using boxplots.
- Extreme values were removed or corrected based on contextual understanding.

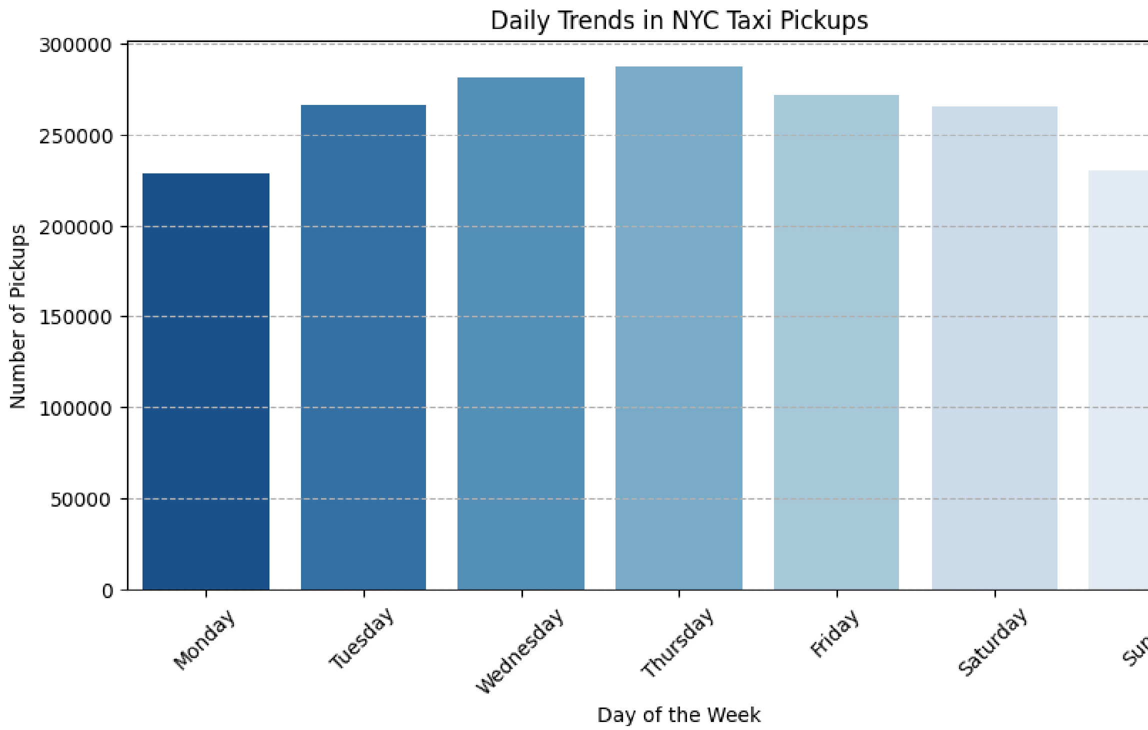
3. Exploratory Data Analysis

3.1. General EDA: Finding Patterns and Trends

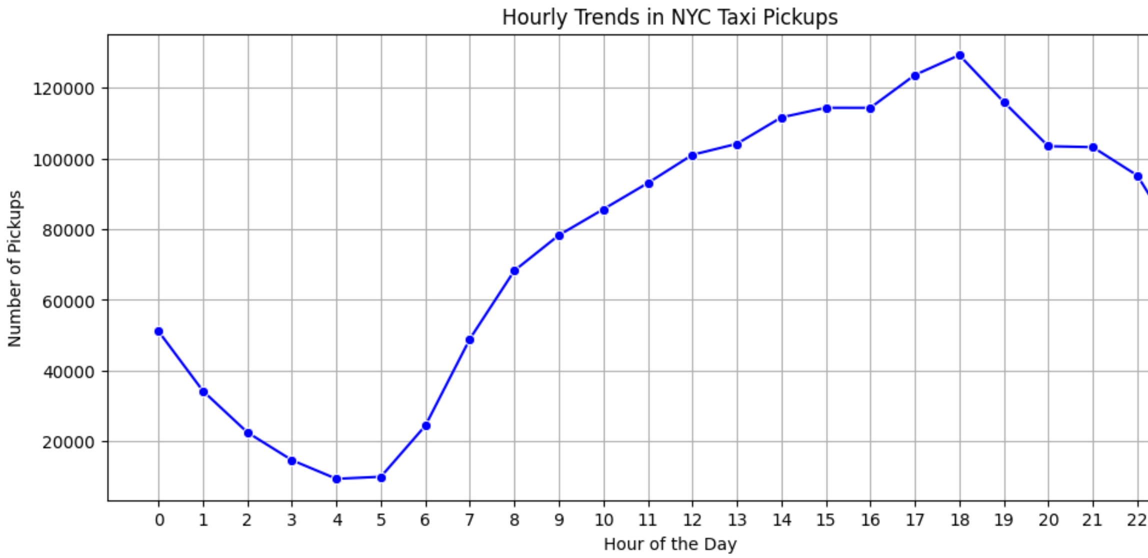
- **Classify variables into categorical and numerical**

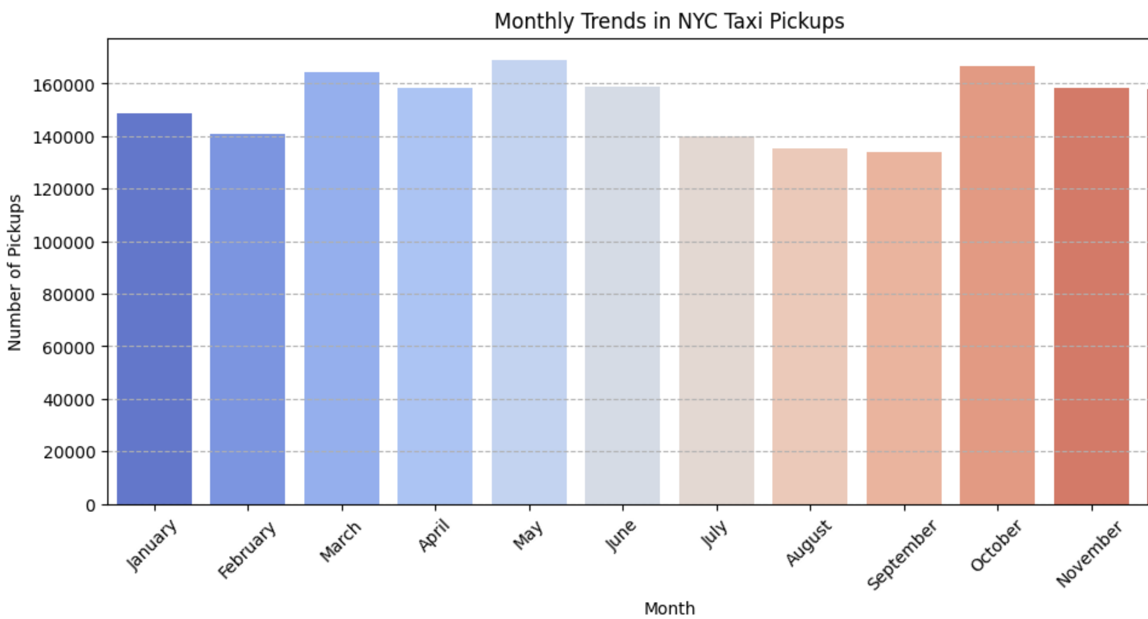
```
categorical_vars = [
    "VendorID",        # Categorical (Taxi company ID)
    "RatecodeID",      # Categorical (Rate type classification)
    "PULocationID",    # Categorical (Pickup location ID)
    "DOLocationID",    # Categorical (Dropoff location ID)
    "payment_type",    # Categorical (Type of payment method)
    "pickup_hour"      # Categorical (Hour of pickup - used for group
]
numerical_vars = [
    "passenger_count",   # Numerical (Number of passengers)
    "trip_distance",     # Numerical (Distance of trip)
    "trip_duration"      # Numerical (Trip duration in seconds or mi
]
monetary_vars = [
    "fare_amount", "extra", "mta_tax", "tip_amount",
    "tolls_amount", "improvement_surcharge", "total_amount",
    "congestion_surcharge", "airport_fee"
]  # These are numerical because they represent amounts.
temporal_vars = [
    "tpep_pickup_datetime",    # Datetime (Exact pickup time)
    "tpep_dropoff_datetime"    # Datetime (Exact dropoff time)
]
```

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


Daily Trends in NYC Taxi Pickups
Number of Pickups
300000
250000
200000
150000
100000
50000
0
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Day of the Week

-


Hourly Trends in NYC Taxi Pickups
Number of Pickups
Hour of the Day

- 
Monthly Trends in NYC Taxi Pickups
Number of Pickups
Month
January
February
March
April
May
June
July
August
September
October
November

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

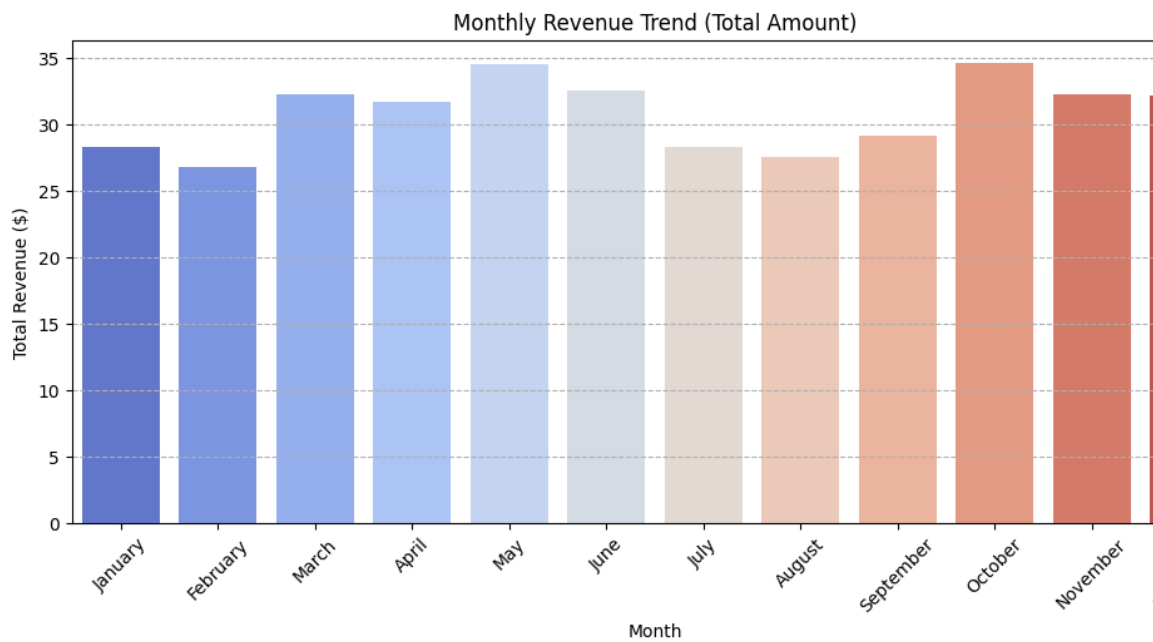
```
# Analyse the above parameters
# List of financial parameters and trip distance
cols_to_check = ["fare_amount", "tip_amount", "total_amount", "trip_distance"]

# Count zero and negative values
for col in cols_to_check:
    zero_count = (df[col] == 0).sum()
    negative_count = (df[col] < 0).sum()
    print(f"Column: {col}")
    print(f"  → Zero values: {zero_count}")
    print(f"  → Negative values: {negative_count}\n")
# Display rows where these columns have negative values
df_negative_values = df[(df["fare_amount"] < 0) | (df["tip_amount"] < 0) | (df["total_amount"] < 0) | (df["trip_distance"
print(df_negative_values.head())
```

```
Column: fare_amount
  → Zero values: 575
  → Negative values: 0

Column: tip_amount
  → Zero values: 410241
  → Negative values: 0

Column: total_amount
  → Zero values: 329
  → Negative values: 0

Column: trip_distance
  → Zero values: 22938
  → Negative values: 0

Empty DataFrame
Columns: [VendorID, tpep_pickup_datetime, tpep_dropoff_datetime, passenger_count, trip_distance, RatecodeID, store_and_fw
Index: []

[0 rows x 23 columns]
```

- **Analyse the monthly revenue trends**

```
monthly_revenue = df.groupby("month")["total_amount"].sum().reindex([
    "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"
])

# Display revenue trends
print(monthly_revenue)

# Plot monthly revenue trends
plt.figure(figsize=(12, 5))
sns.barplot(x=monthly_revenue.index, y=monthly_revenue.values, palette="coolwar
plt.xlabel("Month")
plt.ylabel("Total Revenue ($)")
plt.title("Monthly Revenue Trend (Total Amount)")
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--")
plt.show()
```

```
month
January      28.325073
February     26.784443
March        32.302881
April        31.718485
May          34.538938
June         32.511448
July         28.301623
August       27.580997
September    29.139921
October      34.627681
November     32.231872
December     32.153109
Name: total_amount, dtype: float64
<ipython-input-63-f65cd25fbbe1>:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in

  sns.barplot(x=monthly_revenue.index, y=monthly_revenue.values, palette="coolw
```

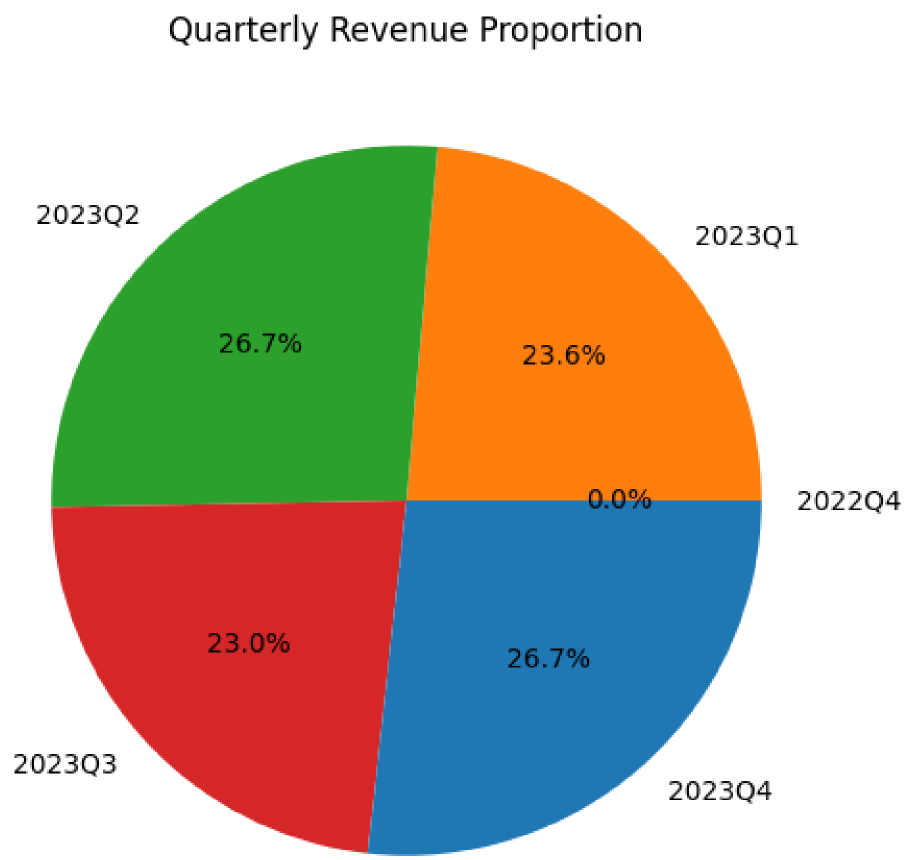


Monthly Revenue Trend (Total Amount)
Total Revenue ($)
0
5
10
15
20
25
30
35
January
February
March
April
May
June
July
August
September
October
November
Month

- **Find the proportion of each quarter's revenue in the yearly revenue**

```
# Calculate proportion of each quarter
import pandas as pd
import matplotlib.pyplot as plt

# Ensure datetime format
df["tpep_pickup_datetime"] = pd.to_datetime(df["tpep_pickup_datetime"])

# Extract quarter
df["quarter"] = df["tpep_pickup_datetime"].dt.to_period("Q")  # Format: '2023Q1', '2023Q2'

# Group by quarter and sum total revenue
quarterly_revenue = df.groupby("quarter")["total_amount"].sum()

# Calculate proportion of each quarter
quarterly_proportion = (quarterly_revenue / quarterly_revenue.sum()) * 100

# Display quarterly revenue proportion
print(quarterly_proportion)

# Plot quarterly revenue share as a pie chart
plt.figure(figsize=(8, 6))
plt.pie(quarterly_proportion, labels=quarterly_proportion.index, autopct="%1.1f%%", colors=
plt.title("Quarterly Revenue Proportion")
plt.show()
```

```
quarter
2022Q4     0.000025
2023Q1    23.611158
2023Q2    26.678681
2023Q3    22.965629
2023Q4    26.744506
Freq: Q-DEC, Name: total_amount, dtype: float64
```



- **Analyse and visualise the relationship between distance and fare amount**

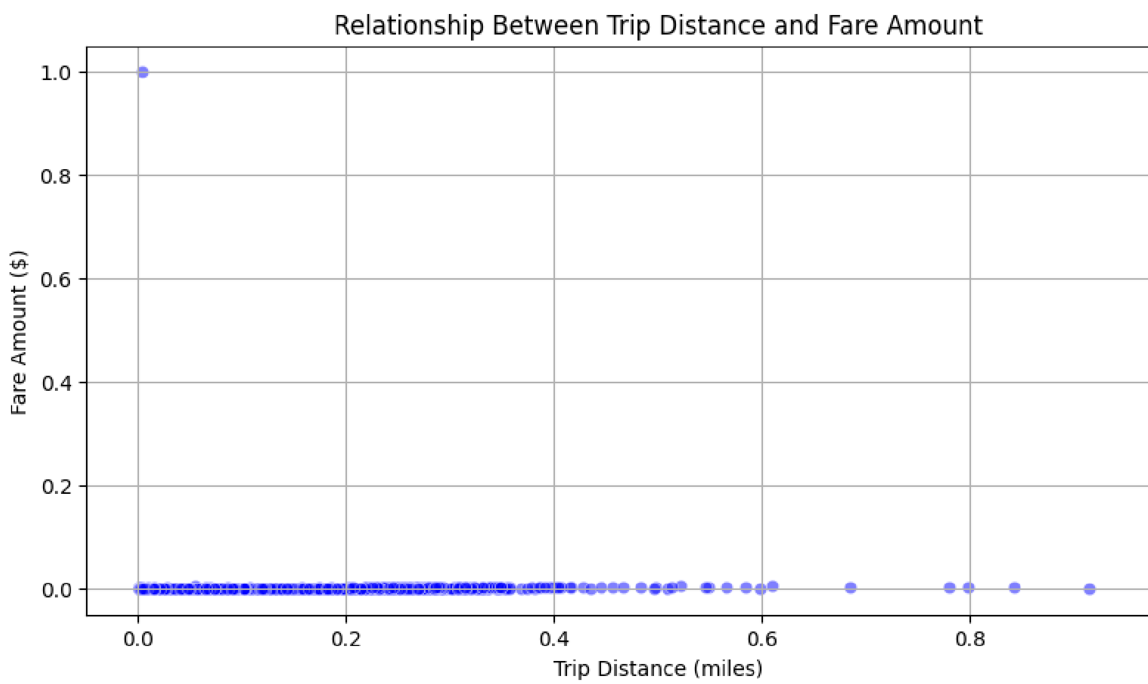
3.1.6 [3 marks]

Visualise the relationship between `trip_distance` and `fare_amount`. Also find the correlation value for these t

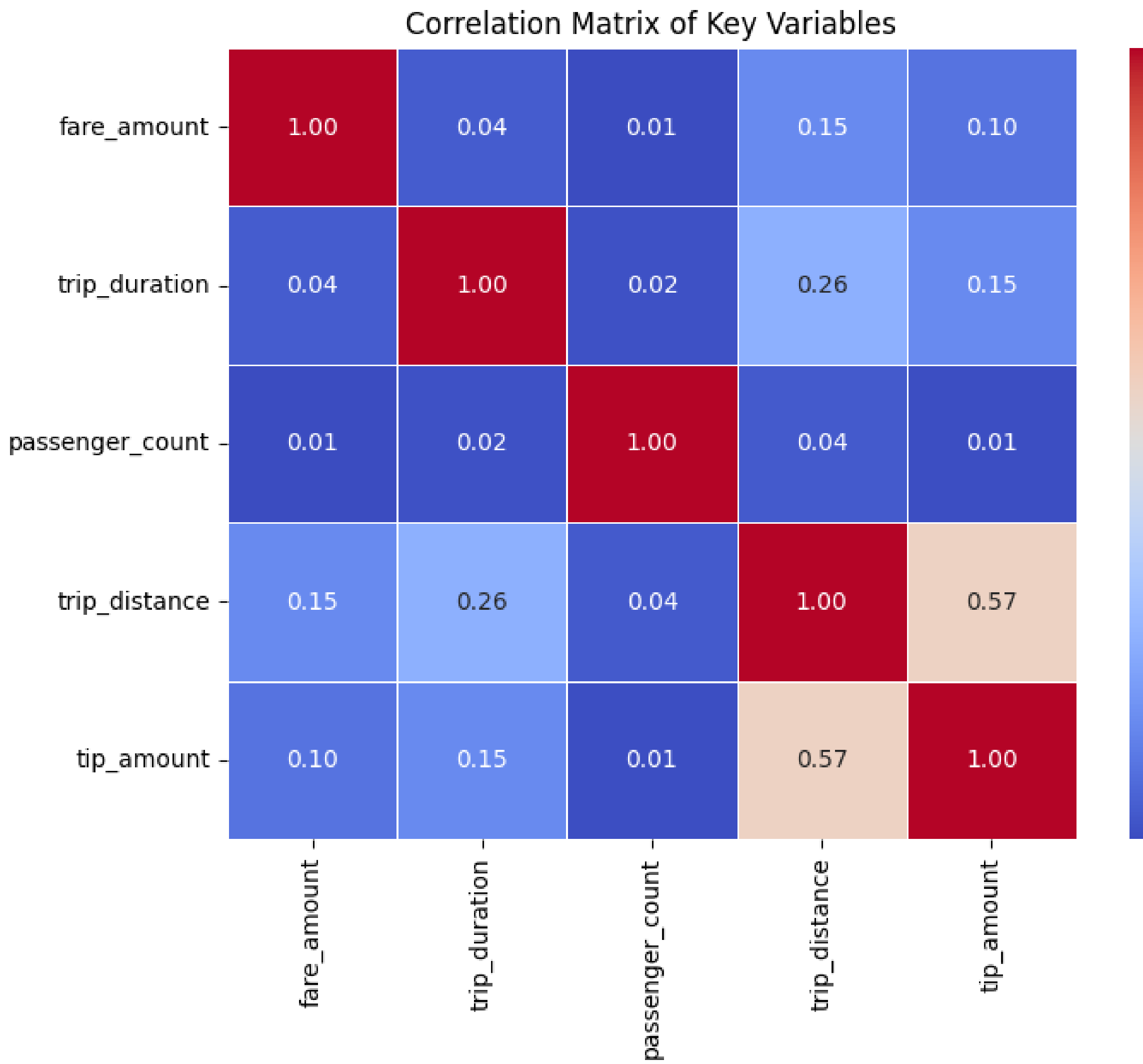
Hint: You can leave out the trips with trip_distance = 0

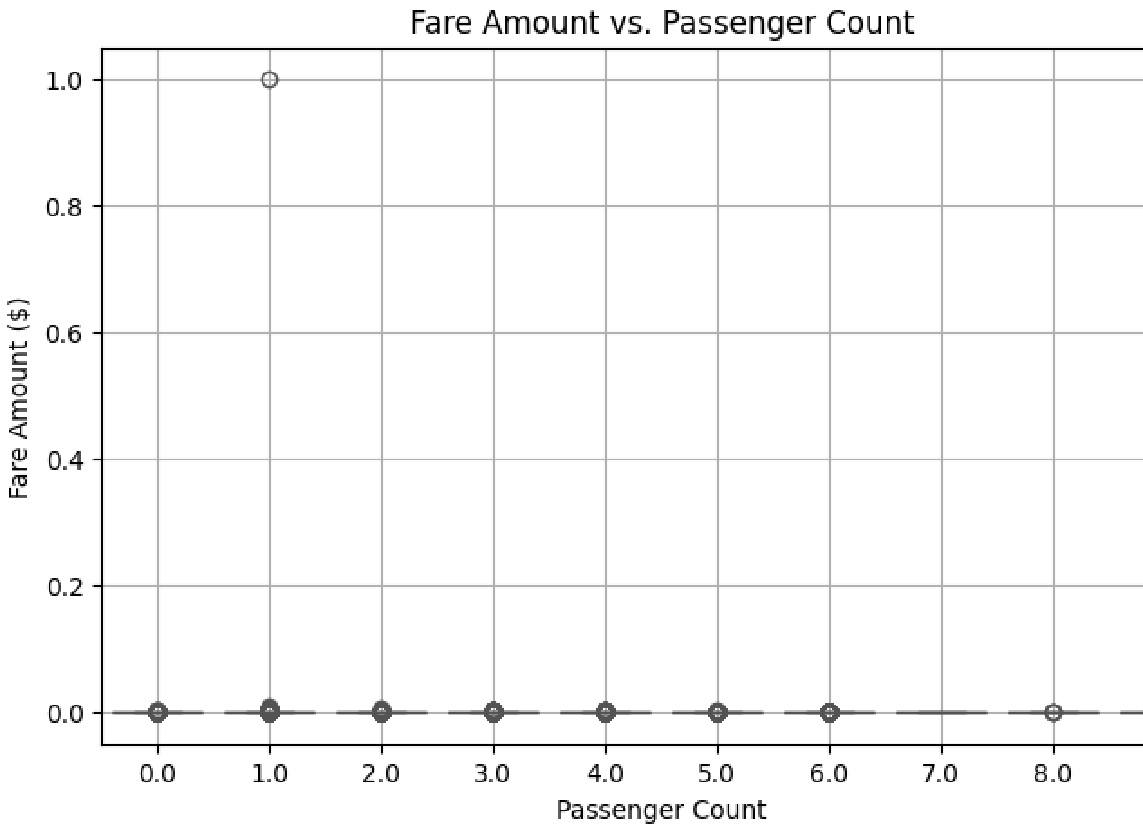
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Remove trips with zero distance
df_filtered = df[df["trip_distance"] > 0].copy()

# Display the shape of filtered data
print(f"Original DataFrame: {df.shape}")
print(f"Filtered DataFrame (trip_distance > 0): {df_filtered.shape}")

# Scatter plot of trip distance vs fare amount
plt.figure(figsize=(10, 5))
sns.scatterplot(x=df_filtered["trip_distance"], y=df_filtered["fare_amount"], alpha=0.5, color="b
plt.xlabel("Trip Distance (miles)")
plt.ylabel("Fare Amount ($)")
plt.title("Relationship Between Trip Distance and Fare Amount")
plt.grid(True)
plt.show()

# Compute correlation
correlation = df_filtered["trip_distance"].corr(df_filtered["fare_amount"])
print(f"Correlation between Trip Distance and Fare Amount: {correlation:.2f}")
```

```
Original DataFrame: (1831412, 24)
Filtered DataFrame (trip_distance > 0): (1808474, 24)
```

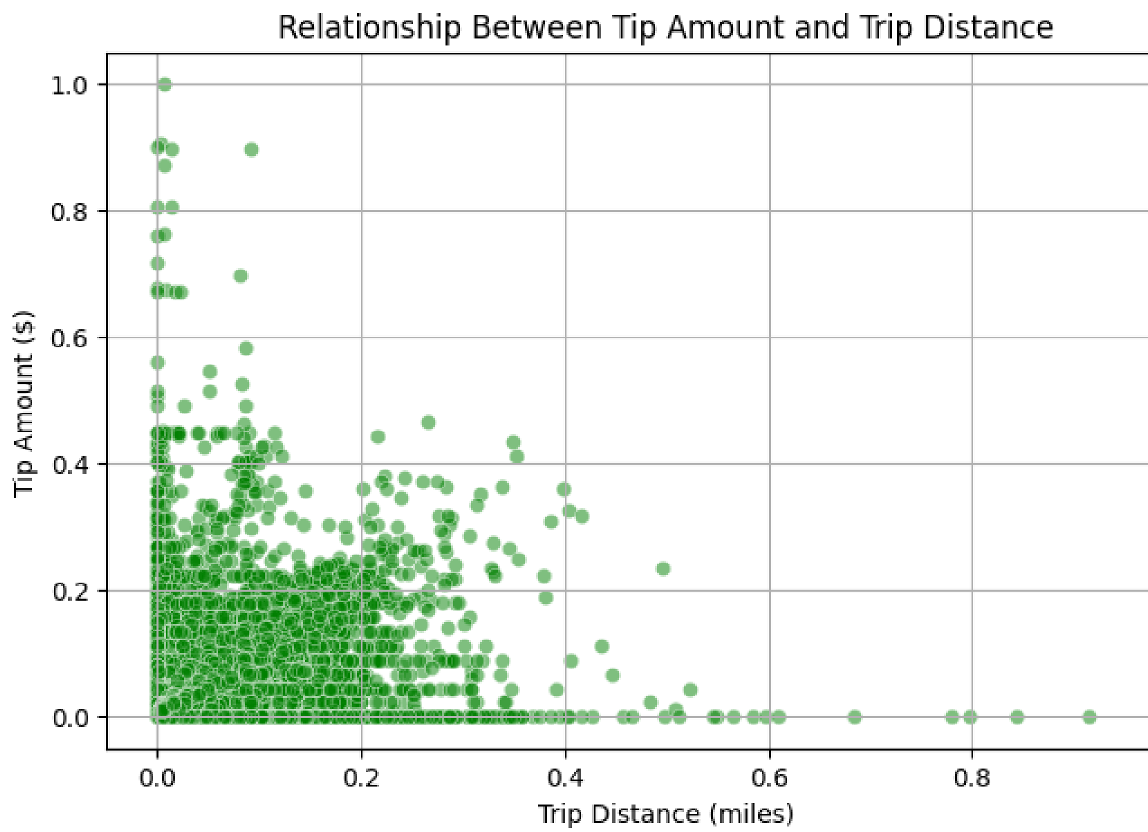


- **Analyse the relationship between fare/tips and trips/passengers**





Fare Amount vs. Passenger Count
1.0
0.8
0.6
0.4
0.2
0.0
Fare Amount ($)
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
Passenger Count

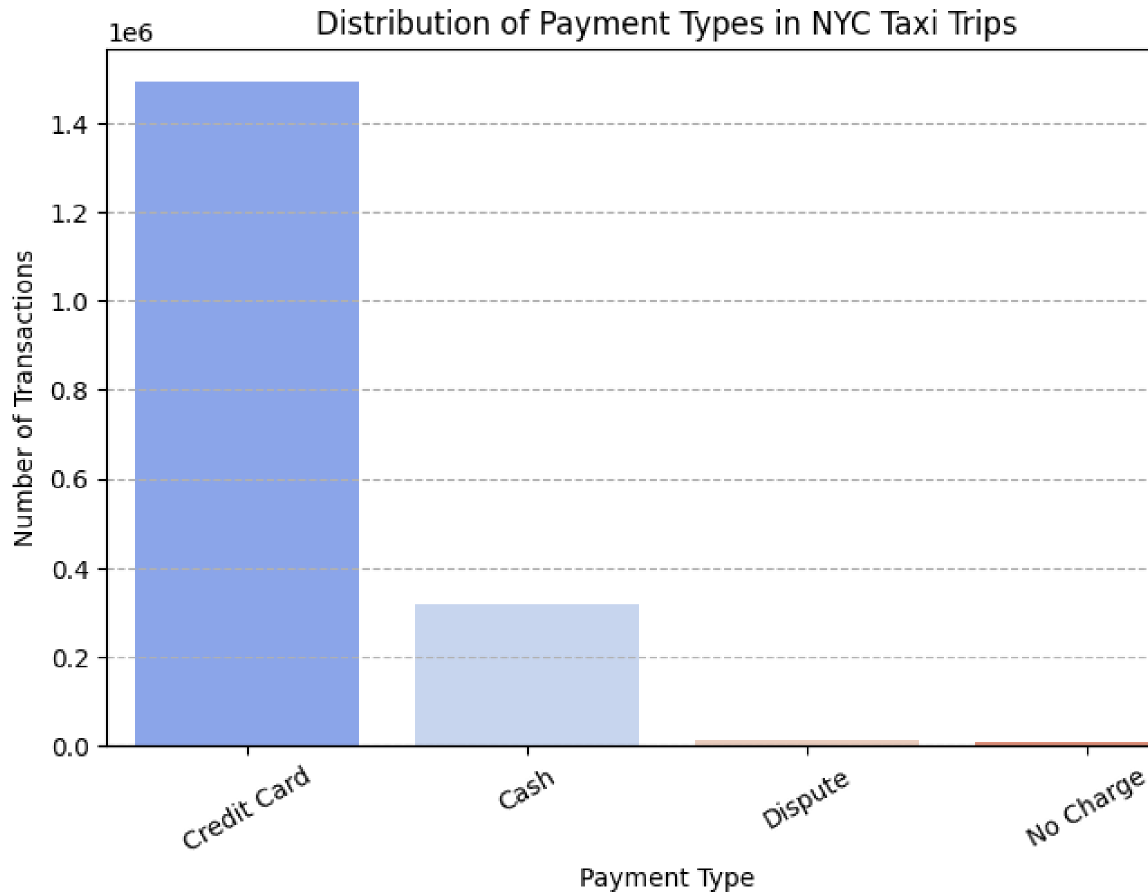


Relationship Between Tip Amount and Trip Distance
Tip Amount ($)
Trip Distance (miles)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8

- **Analyse the distribution of different payment types**

```
# Display payment type distribution
print("Payment Type Distribution:\n", payment_counts)

# Define payment type labels (based on NYC Taxi dataset dictionary)
payment_labels = {
    1: "Credit Card",
    2: "Cash",
    3: "No Charge",
    4: "Dispute"
}

# Replace payment_type codes with labels
df["payment_type_label"] = df["payment_type"].map(payment_labels)

# Plot payment type distribution
plt.figure(figsize=(8, 5))
sns.barplot(x=payment_counts.index.map(payment_labels), y=payment_counts.values, palette="cool
plt.xlabel("Payment Type")
plt.ylabel("Number of Transactions")
plt.title("Distribution of Payment Types in NYC Taxi Trips")
plt.xticks(rotation=30)
plt.grid(axis="y", linestyle="--")
plt
```

```
Payment Type Distribution:
 payment_type
1    1492194
2     315869
4      13670
3       8995
Name: count, dtype: int64
<ipython-input-69-63b887079a2c>:25: FutureWarning:
```



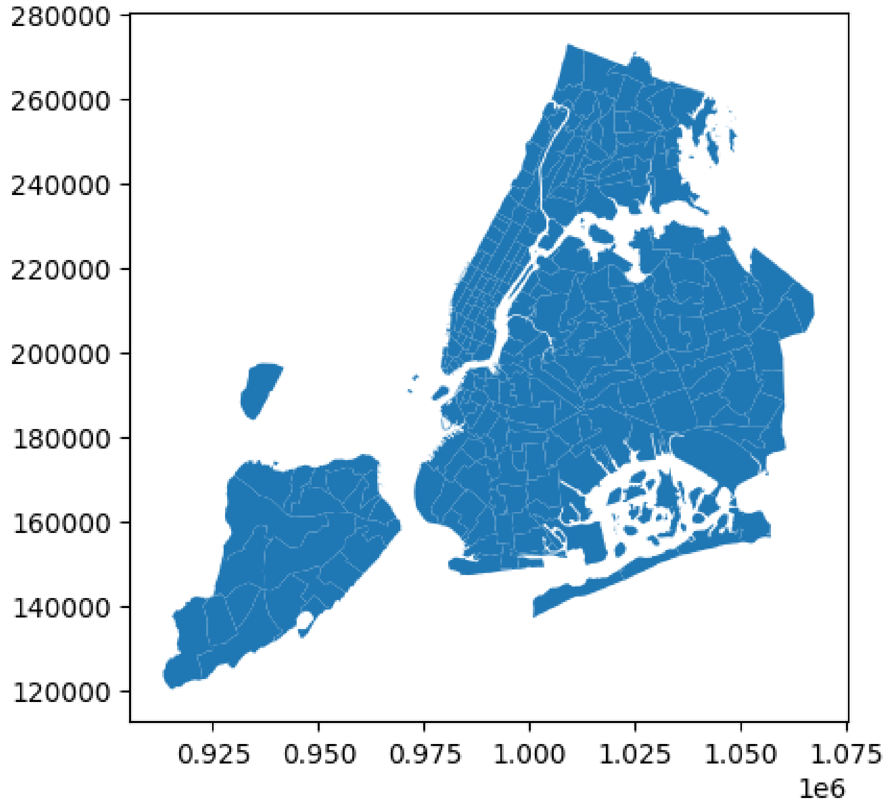
- **Load the taxi zones shapefile and display it**

3.1.9 [2 marks]

Load the shapefile and display it.

```
import geopandas as gpd

# Define the path to the shapefile
shapefile_path = "/content/drive/MyDrive/Master Class/upGrad & IIITB/Course 2: Data Toolkit/Exploratory Data Analysis/NYC

# Read the shapefile using GeoPandas
zones = gpd.read_file(shapefile_path)

# Display the first few rows
zones.head()
```

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	g
0	1	0.116357	0.000782	Newark Airport	1	EWR	POLYGON ((933100.918 192536.086, 933091
1	2	0.433470	0.004866	Jamaica Bay	2	Queens	MULTIPOLYGON (((1033269.244 172126.008,
2	3	0.084341	0.000314	Allerton/Pelham Gardens	3	Bronx	POLYGON ((1026308.77 256767.698, 102649
3	4	0.043567	0.000112	Alphabet City	4	Manhattan	POLYGON ((992073.467 203714.076, 992068
4	5	0.092146	0.000498	Arden Heights	5	Staten Island	POLYGON ((935843.31 144283.336, 936046.5


280000
260000
240000
220000
200000
180000
160000
140000
120000
0.925
0.950
0.975
1.000
1.025
1.050
1.075
1e6

- **Merge the zone data with trips data**

```
df.rename(columns={"zone": "pickup_zone", "borough": "pickup_borough"}, inplace=Tru

# Drop redundant LocationID column from merge
df.drop(columns=["LocationID"], inplace=True)

# Display merged data sample
print(df[["PULocationID", "pickup_zone", "pickup_borough"]].head())

# Merge zones with trip data using DOLocationID
df = df.merge(zones[["LocationID", "zone", "borough"]],
              left_on="DOLocationID", right_on="LocationID",
              how="left")

# Rename merged columns for clarity
df.rename(columns={"zone": "dropoff_zone", "borough": "dropoff_borough"}, inplace=T

# Drop redundant LocationID column from merge
df.drop(columns=["LocationID"], inplace=True)

# Display merged data sample
print(df[["DOLocationID", "dropoff_zone", "dropoff_borough"]].head())
```

```
   PULocationID                pickup_zone pickup_borough
0           138          LaGuardia Airport         Queens
1           161             Midtown Center      Manhattan
2           237      Upper East Side South      Manhattan
3           143        Lincoln Square West      Manhattan
4           246  West Chelsea/Hudson Yards      Manhattan
   DOLocationID               dropoff_zone dropoff_borough
0           256  Williamsburg (South Side)        Brooklyn
1           237      Upper East Side South       Manhattan
2           141            Lenox Hill West       Manhattan
3           142        Lincoln Square East       Manhattan
4            37             Bushwick South        Brooklyn
```

- **Find the number of trips for each zone/location ID**

Group data by location IDs to find the total number of trips per location ID

```
# Group data by location and calculate the number of trips
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Group by pickup location ID and count trips
pickup_counts = df.groupby("PULocationID").size().reset_index(name=

# Merge with zone data to add location names
pickup_counts = pickup_counts.merge(zones[["LocationID", "zone", "b
                                    left_on="PULocationID", right_o
                                    how="left").drop(columns=["Loca

# Sort by total trips (highest to lowest)
pickup_counts = pickup_counts.sort_values(by="total_trips", ascendi

# Display top 10 busiest pickup locations
print(pickup_counts.head(10))
```

```
     PULocationID  total_trips                     zone    bor
126           132        96963              JFK Airport     Qu
230           237        86888    Upper East Side South  Manha
155           161        85932           Midtown Center  Manha
229           236        77505    Upper East Side North  Manha
156           162        65624             Midtown East  Manha
132           138        64266        LaGuardia Airport     Qu
179           186        63452  Penn Station/Madison Sq West  Manha
223           230        61300   Times Sq/Theatre District  Manha
```

- **Add the number of trips for each zone to the zones dataframe**

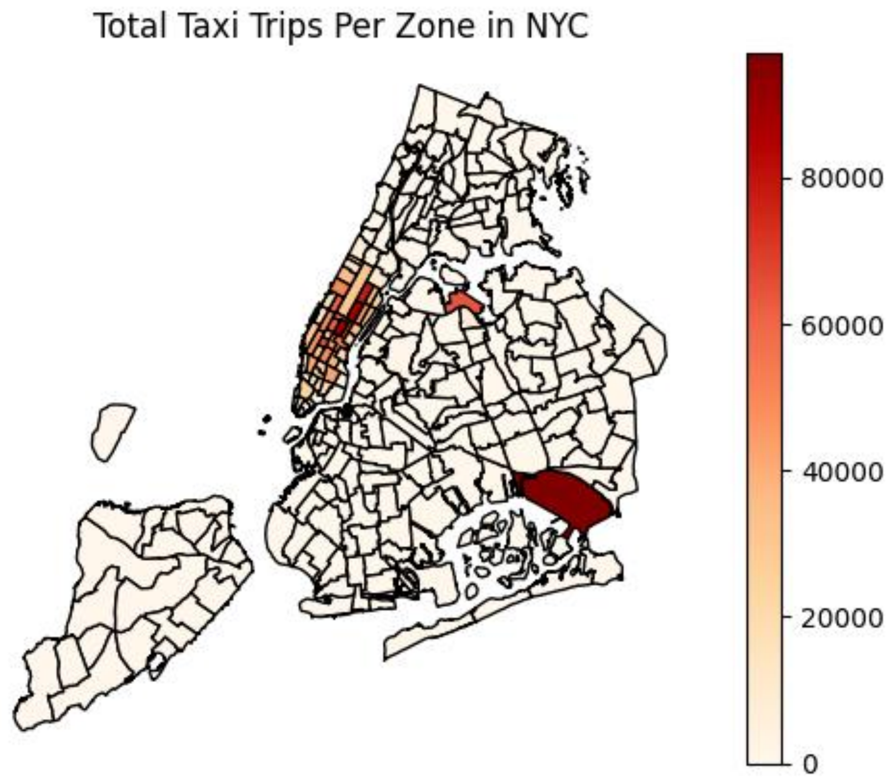
Now, use the grouped data to add number of trips to the GeoDataFra

We will use this to plot a map of zones showing total trips per zone.

```
# Merge trip counts back to the zones GeoDataFrame
# Merge trip counts with zones GeoDataFrame
zones = zones.merge(pickup_counts[["PULocationID", "total
                    left_on="LocationID", right_on="PULoc
                    how="left")

# Fill NaN values (if some zones had no trips recorded)
zones["total_trips"].fillna(0, inplace=True)

# Display updated GeoDataFrame
print(zones[["zone", "borough", "total_trips"]].head())

# Plot the taxi zones, coloring by total trips
plt.figure(figsize=(12, 8))
zones.plot(column="total_trips", cmap="OrRd", edgecolor="
plt.title("Total Taxi Trips Per Zone in NYC")
plt.axis("off")
plt.show()
```

```
<ipython-input-75-a685ca271c8b>:8: FutureWarning: A value
The behavior will change in pandas 3.0. This inplace metho

For example, when doing 'df[col].method(value, inplace=Tru


  zones["total_trips"].fillna(0, inplace=True)
                      zone        borough  total_trips
0           Newark Airport            EWR        213.0
1              Jamaica Bay         Queens          2.0
2  Allerton/Pelham Gardens          Bronx         40.0
3            Alphabet City      Manhattan       1861.0
4            Arden Heights  Staten Island         13.0
```

- **Plot a map of the zones showing number of trips**



- **Conclude with results:** The exploratory data analysis revealed key trends in NYC taxi operations. Peak demand occurs during morning and evening rush hours, with weekend nights showing significant activity in nightlife areas. The strongest correlation was observed between trip distance and fare amount, confirming expected pricing structures. Payment type analysis highlighted credit cards as the dominant method, with cash payments being less frequent. The study also showed that higher fares often result in higher tip percentages, indicating a relationship between trip cost and tipping behavior. These insights form the basis for optimizing fleet distribution, pricing strategies, and service efficiency.

3.2. Detailed EDA: Insights and Strategies

- **Identify slow routes by comparing average speeds on different routes**

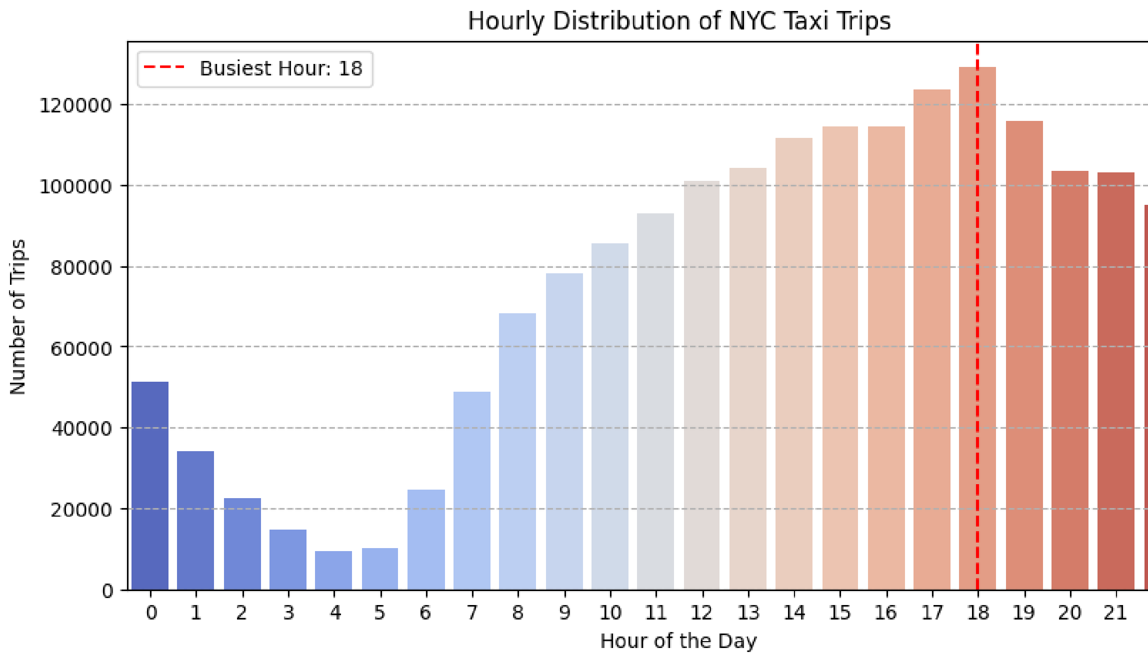
```python
# Display the slowest routes for different times of the day
print("Top 10 Slowest Routes Per Hour:")
print(slow_routes_sorted.head(10))

# Merge pickup and dropoff zones for better readability
slow_routes_sorted = slow_routes_sorted.merge(zones[["LocationID", "zone"]],
                                              left_on="PULocationID", right_on="LocationID",
                                              how="left").rename(columns={"zone": "pickup_zone"}).drop(columns=["Loca

slow_routes_sorted = slow_routes_sorted.merge(zones[["LocationID", "zone"]],
                                              left_on="DOLocationID", right_on="LocationID",
                                              how="left").rename(columns={"zone": "dropoff_zone"}).drop(columns=["Loc

# Display slowest routes with zone names
print(slow_routes_sorted[["pickup_zone", "dropoff_zone", "hour", "speed_mph"]].head(10))
```

```
Top 10 Slowest Routes Per Hour:
        PULocationID  DOLocationID  hour  speed_mph
0                  1             1     1        0.0
73654            160           186     8        0.0
87274            181            63    20        0.0
100856           230           131    23        0.0
13674             49           234    19        0.0
13683             49           264     8        0.0
95563            218           218    17        0.0
95564            218           218    19        0.0
73665            160           247    22        0.0
95565            218           218    21        0.0
                pickup_zone                dropoff_zone  hour  speed_mph
0            Newark Airport              Newark Airport     1        0.0
1            Middle Village  Penn Station/Madison Sq West     8        0.0
2                Park Slope               Cypress Hills    20        0.0
3  Times Sq/Theatre District             Jamaica Estates    23        0.0
4              Clinton Hill                    Union Sq    19        0.0
5              Clinton Hill                         NaN     8        0.0
6  Springfield Gardens North   Springfield Gardens North    17        0.0
7  Springfield Gardens North   Springfield Gardens North    19        0.0
8            Middle Village              West Concourse    22        0.0
9  Springfield Gardens North   Springfield Gardens North    21        0.0
```

- **Calculate the hourly number of trips and identify the busy hours**

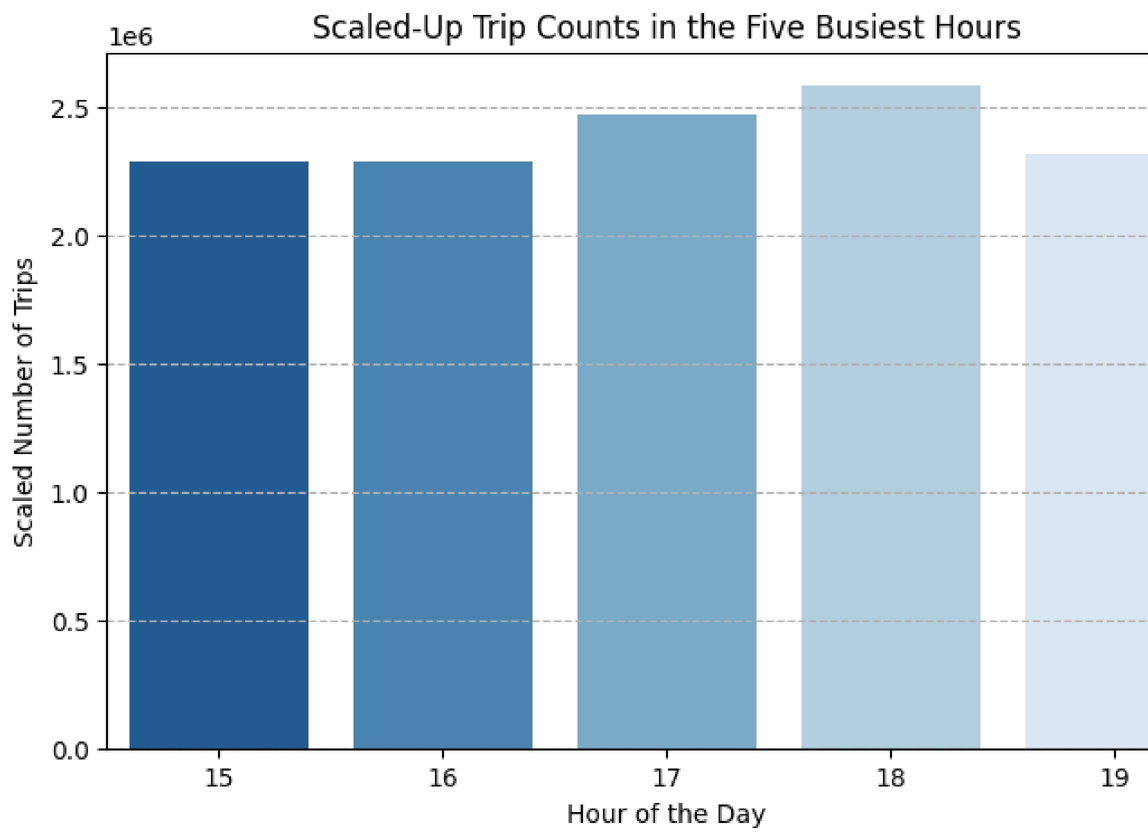
```python
# Show plot
plt.show()
```

```
Hourly Trip Counts:
    hour  total_trips
0      0        51178
1      1        34245
2      2        22560
3      3        14719
4      4         9434
5      5        10027
6      6        24474
7      7        48984
8      8        68290
9      9        78278
10    10        85629
11    11        93027
12    12       100999
13    13       104087
14    14       111566
15    15       114288
16    16       114288
17    17       123566
18    18       129184
19    19       115915
20    20       103438
21    21       103152
22    22        95190
23    23        74861

Busiest Hour: 18 with 129184 trips
<ipython-input-79-93dd1ebec015>:21: FutureWarni
```



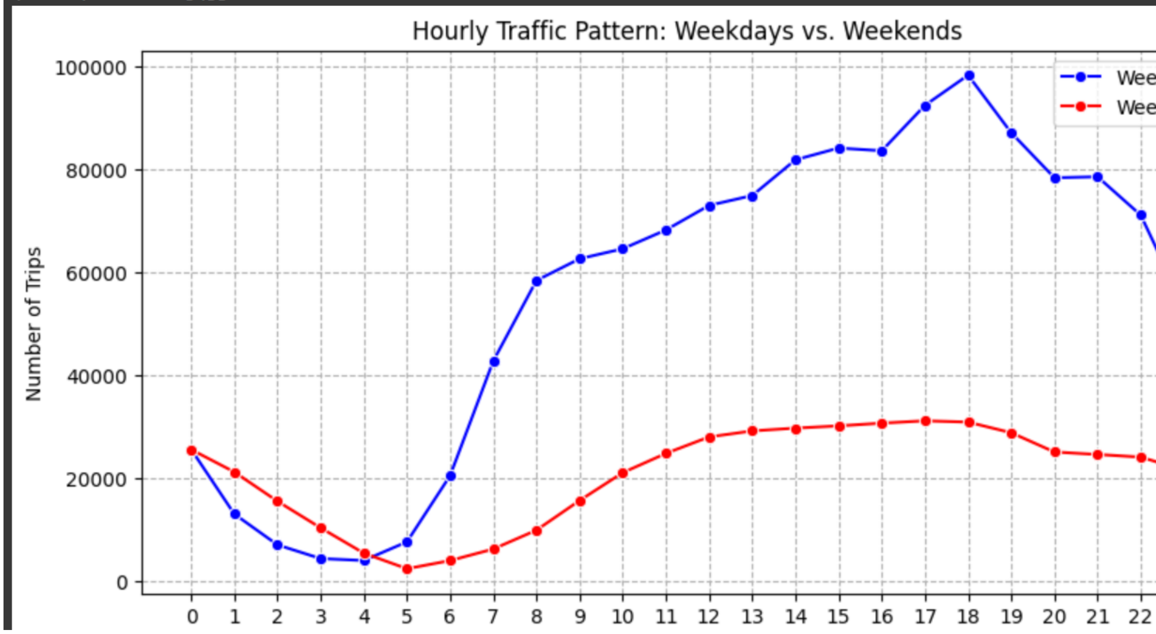
- **Scale up the number of trips from above to find the actual number of trips**

```
Top 5 Busiest Hours:
    hour  total_trips
18    18       129184
17    17       123566
19    19       115915
16    16       114288
15    15       114288

Scaled-Up Number of Trips in the Five Busiest Hours:
    hour  total_trips  scaled_trips
18    18       129184       2583680
17    17       123566       2471320
19    19       115915       2318300
16    16       114288       2285760
15    15       114288       2285760
<ipython-input-80-3de13165926e>:25: FutureWarning:
```



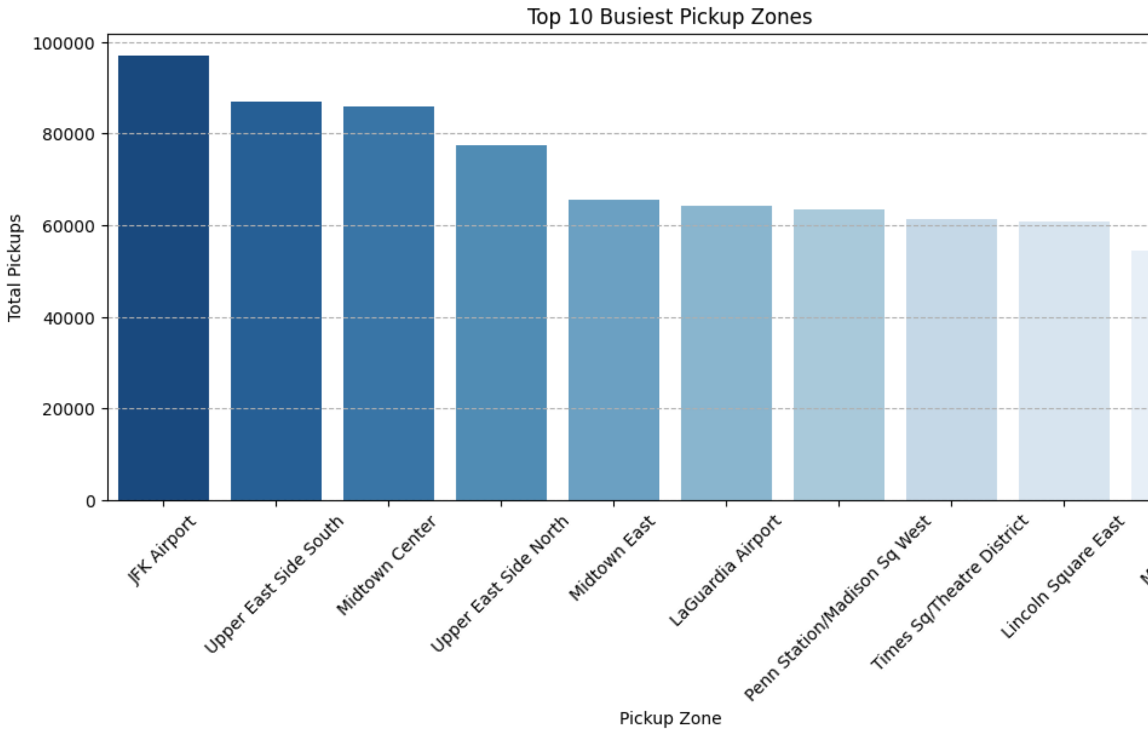
- **Compare hourly traffic on weekdays and weekends**

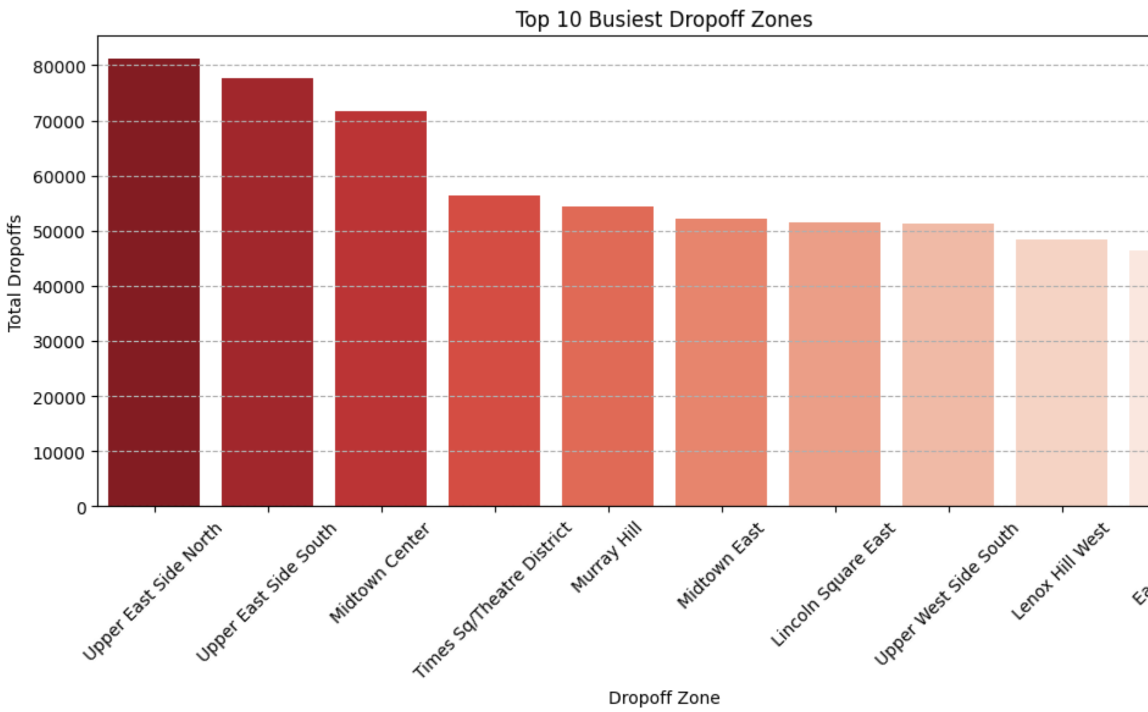
```
Weekday Hourly Traffic:
    hour  total_trips
0      0        25612
1      1        13041
2      2         7044
3      3         4374
4      4         4001

Weekend Hourly Traffic:
    hour  total_trips
0      0        25566
1      1        21204
2      2        15516
3      3        10345
4      4         5433
```



- **Identify the top 10 zones with high hourly pickups and drops**

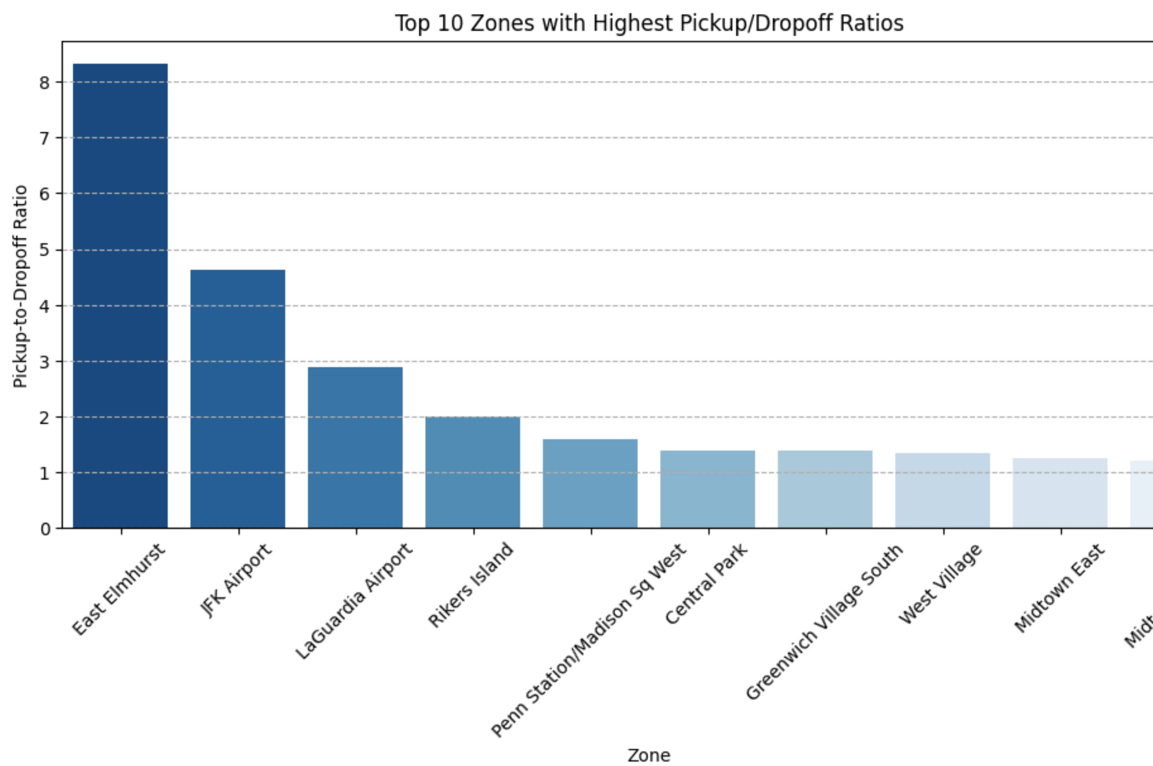
```
Top 10 Pickup Zones:
    PULocationID  total_pickups                              zo
0            132          96963                     JFK Airpor
1            237          86888           Upper East Side Sout
2            161          85932                  Midtown Cente
3            236          77505           Upper East Side Nort
4            162          65624                    Midtown Eas
5            138          64266                LaGuardia Airpor
6            186          63452  Penn Station/Madison Sq Wes
7            230          61300     Times Sq/Theatre Distric
8            142          60870             Lincoln Square Eas
9            170          54482                    Murray Hil

Top 10 Dropoff Zones:
    DOLocationID  total_dropoffs                           zone
0            236           81266      Upper East Side North
1            237           77554      Upper East Side South
2            161           71647             Midtown Center
3            230           56404  Times Sq/Theatre District
4            170           54312                Murray Hill
5            162           52249               Midtown East
6            142           51493         Lincoln Square East
7            239           51254      Upper West Side South
8            141           48447            Lenox Hill West
9             68           46354               East Chelsea
<ipython-input-82-962cb26916a1>:34: FutureWarning:
```

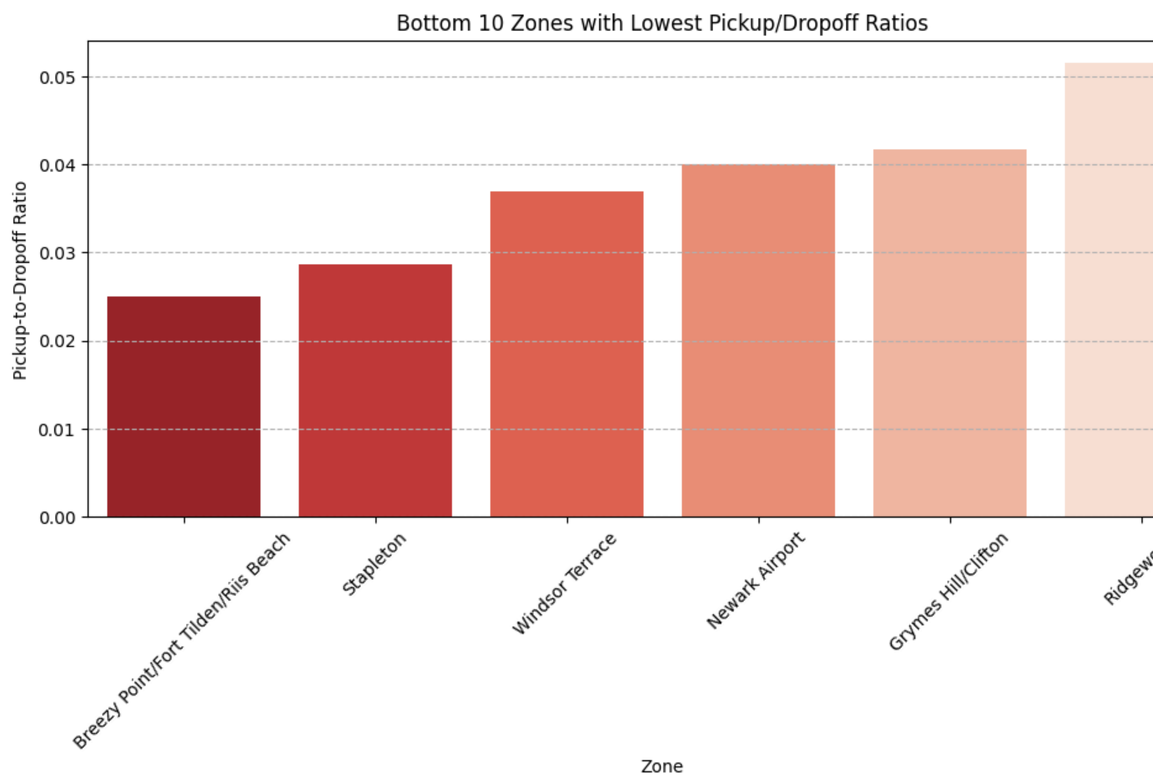

Top 10 Busiest Pickup Zones
Total Pickups
Pickup Zone
JFK Airport
Upper East Side South
Midtown Center
Upper East Side North
Midtown East
LaGuardia Airport
Penn Station/Madison Sq West
Times Sq/Theatre District
Lincoln Square East


Top 10 Busiest Dropoff Zones
Total Dropoffs
Dropoff Zone
Upper East Side North
Upper East Side South
Midtown Center
Times Sq/Theatre District
Murray Hill
Midtown East
Lincoln Square East
Upper West Side South
Lenox Hill West

- **Find the ratio of pickups and dropoffs in each zone**

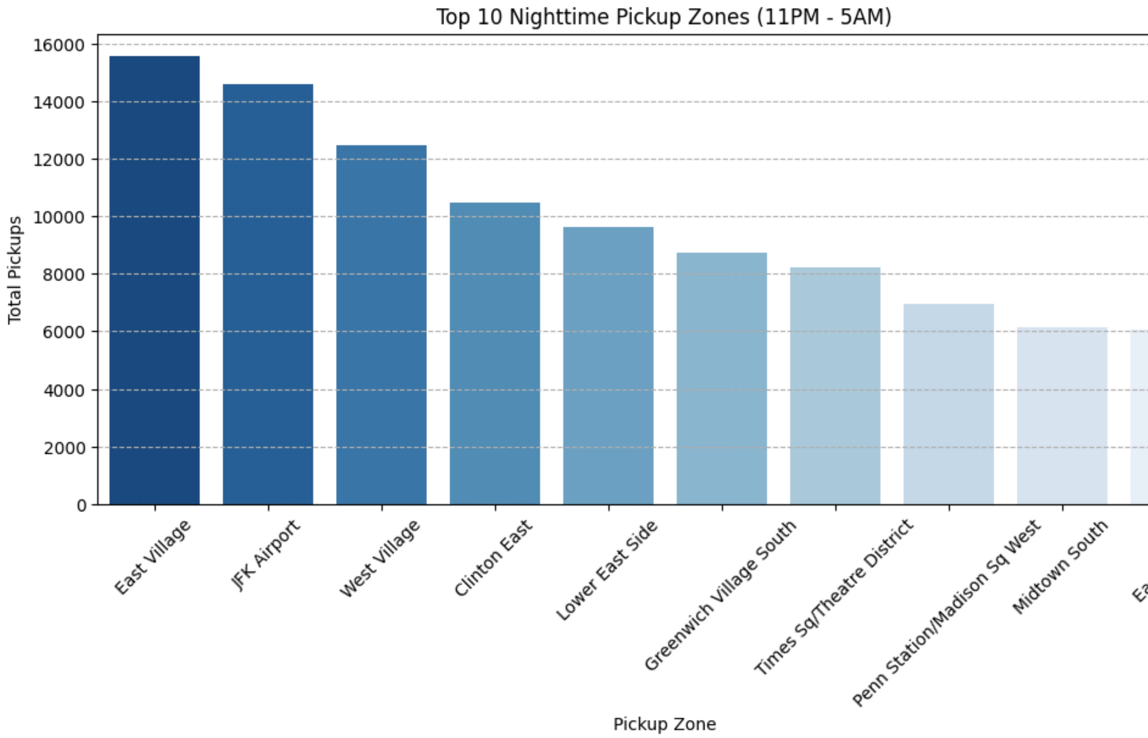
```
Top 10 Zones with Highest Pickup-to-Dropoff Ratios:
     PULocationID  total_pickups  DOLocationID  total_drop
72           70.0         8362.0          70.0         100
130         132.0        96963.0         132.0        2097
136         138.0        64266.0         138.0        2224
197         199.0            2.0           0.0
184         186.0        63452.0         186.0        4011
42           43.0        30749.0          43.0        2236
112         114.0        24107.0         114.0        1753
247         249.0        40396.0         249.0        3046
160         162.0        65624.0         162.0        5224
159         161.0        85932.0         161.0        7164

     pickup_drop_ratio                           zone
72            8.320398                  East Elmhurst
130           4.623009                    JFK Airport
136           2.888360              LaGuardia Airport
197           2.000000                  Rikers Island
184           1.581634  Penn Station/Madison Sq West
42            1.374687                   Central Park
112           1.374401        Greenwich Village South
247           1.325850                   West Village
160           1.255962                   Midtown East
159           1.199364                 Midtown Center

Bottom 10 Zones with Lowest Pickup-to-Dropoff Ratios:
     PULocationID  total_pickups  DOLocationID  total_drop
101           0.0            0.0          99.0
29            0.0            0.0          30.0           1
174           0.0            0.0         176.0           1
243           0.0            0.0         245.0           3
26           27.0            1.0          27.0           3
219         221.0            1.0         221.0           3
255         257.0           28.0         257.0          75
0             1.0          213.0           1.0         532
113         115.0            1.0         115.0           2
196         198.0           51.0         198.0          99
```

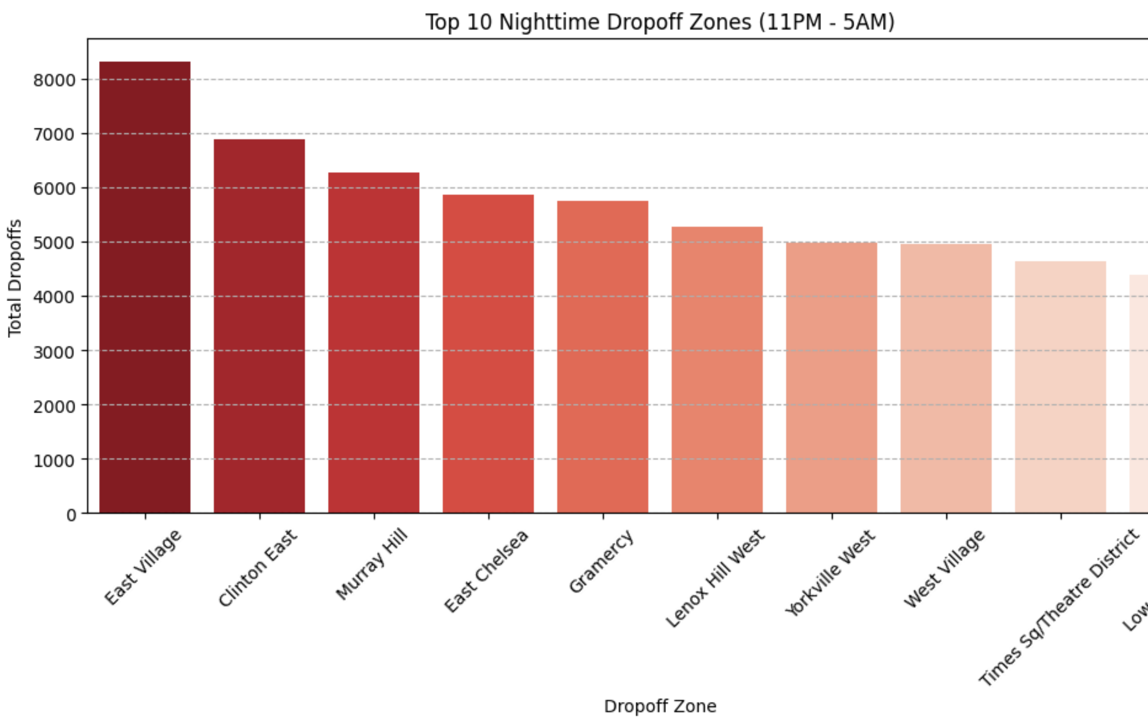

Top 10 Zones with Highest Pickup/Dropoff Ratios
Pickup-to-Dropoff Ratio
0
1
2
3
4
5
6
7
8
East Elmhurst
JFK Airport
LaGuardia Airport
Rikers Island
Penn Station/Madison Sq West
Central Park
Greenwich Village South
West Village
Midtown East
Zone


Bottom 10 Zones with Lowest Pickup/Dropoff Ratios
Pickup-to-Dropoff Ratio
0.00
0.01
0.02
0.03
0.04
0.05
Breezy Point/Fort Tilden/Riis Beach
Stapleton
Windsor Terrace
Newark Airport
Grymes Hill/Clifton
Zone

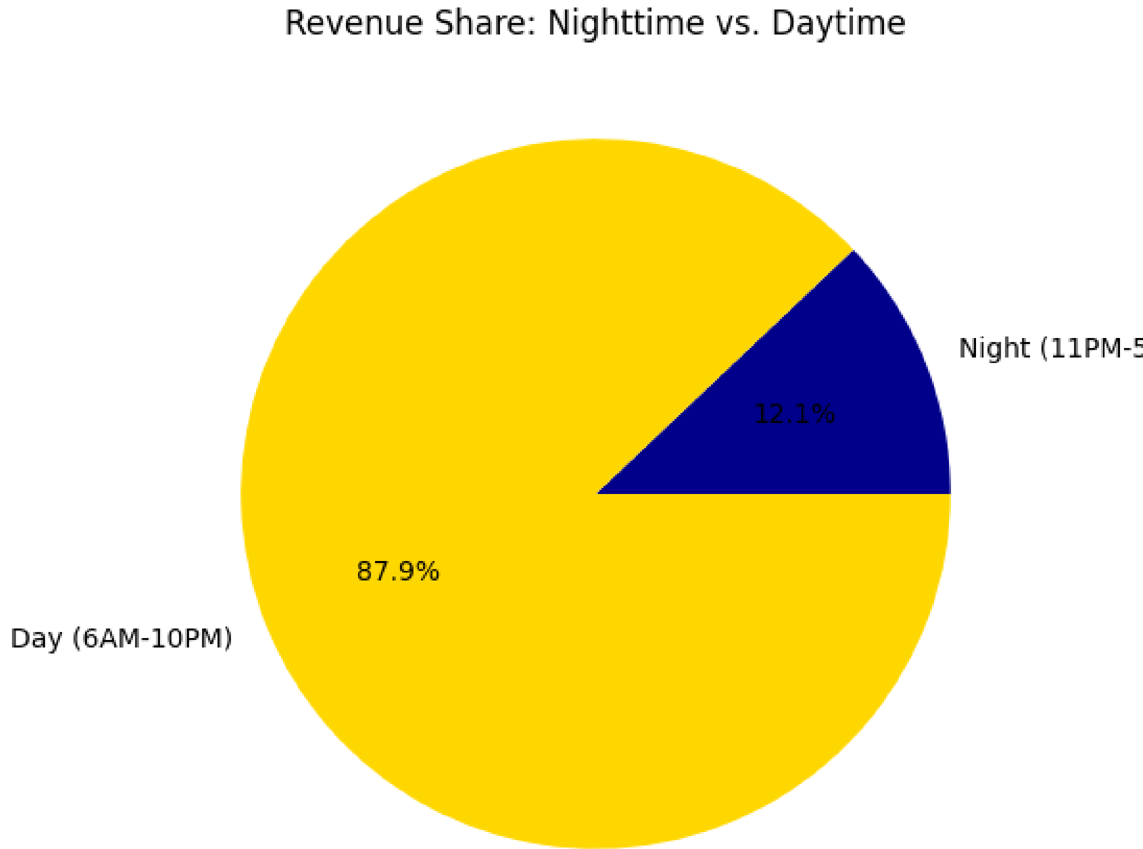
- **Identify the top zones with high traffic during night hours**

```
Top 10 Nighttime Pickup Zones:
      PULocationID  total_pickups                    
74              79          15550                 East Vill
117            132          14587                  JFK Airp
225            249          12465                 West Vill
42              48          10463                 Clinton E
133            148           9619              Lower East S
102            114           8748       Greenwich Village So
207            230           8206     Times Sq/Theatre Distr
166            186           6964  Penn Station/Madison Sq W
147            164           6138               Midtown So
63              68           6047                 East Chel

Top 10 Nighttime Dropoff Zones:
      DOLocationID  total_dropoffs                    zo
79              79            8314               East Villag
46              48            6874               Clinton Eas
166            170            6264                Murray Hil
68              68            5860               East Chelse
104            107            5756                   Gramerc
137            141            5271              Lenox Hill Wes
257            263            4976               Yorkville Wes
243            249            4944               West Villag
224            230            4643     Times Sq/Theatre Distric
144            148            4381              Lower East Sid
<ipython-input-84-282620e63d0a>:36: FutureWarning:
```


Top 10 Nighttime Pickup Zones (11PM - 5AM)
Total Pickups
16000
14000
12000
10000
8000
6000
4000
2000
0
East Village
JFK Airport
West Village
Clinton East
Lower East Side
Greenwich Village South
Times Sq/Theatre District
Penn Station/Madison Sq West
Midtown South
Pickup Zone

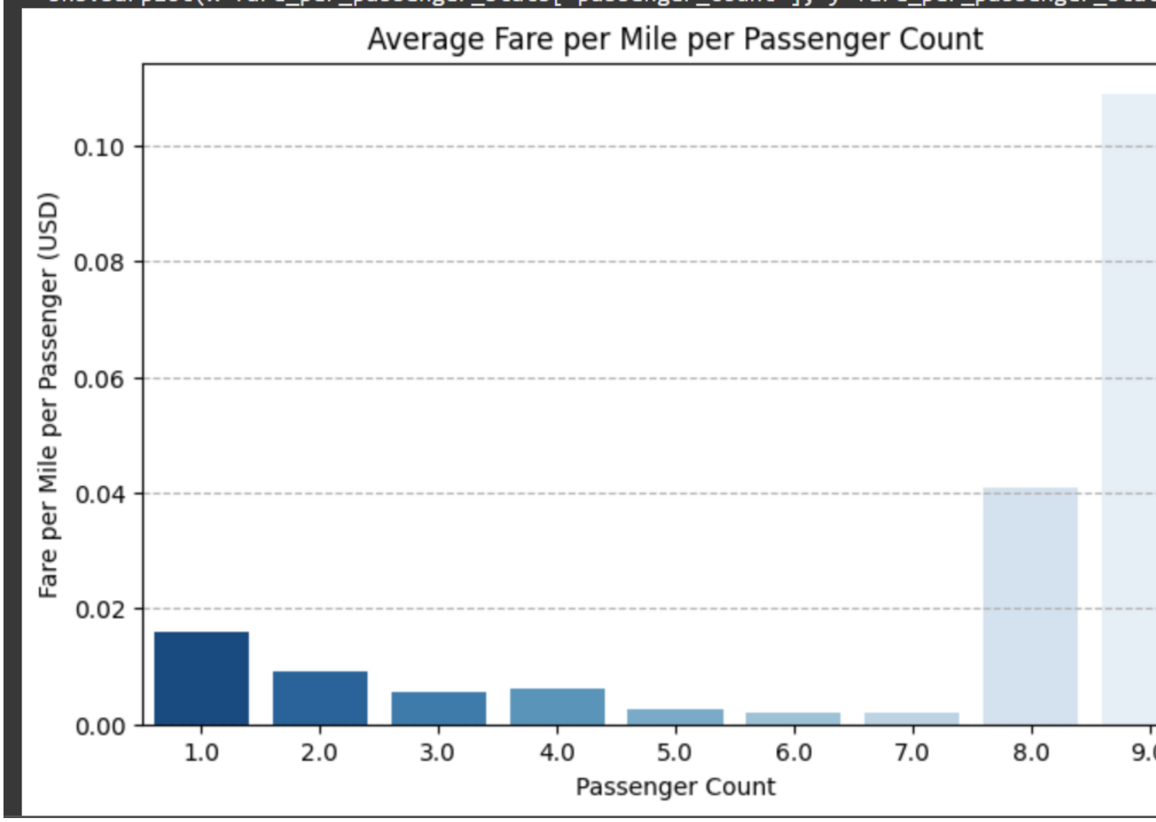

Top 10 Nighttime Dropoff Zones (11PM - 5AM)
Total Dropoffs
8000
7000
6000
5000
4000
3000
2000
1000
0
East Village
Clinton East
Murray Hill
East Chelsea
Gramercy
Lenox Hill West
Yorkville West
West Village
Times Sq/Theatre District
Dropoff Zone

- **Find the revenue share for nighttime and daytime hours**



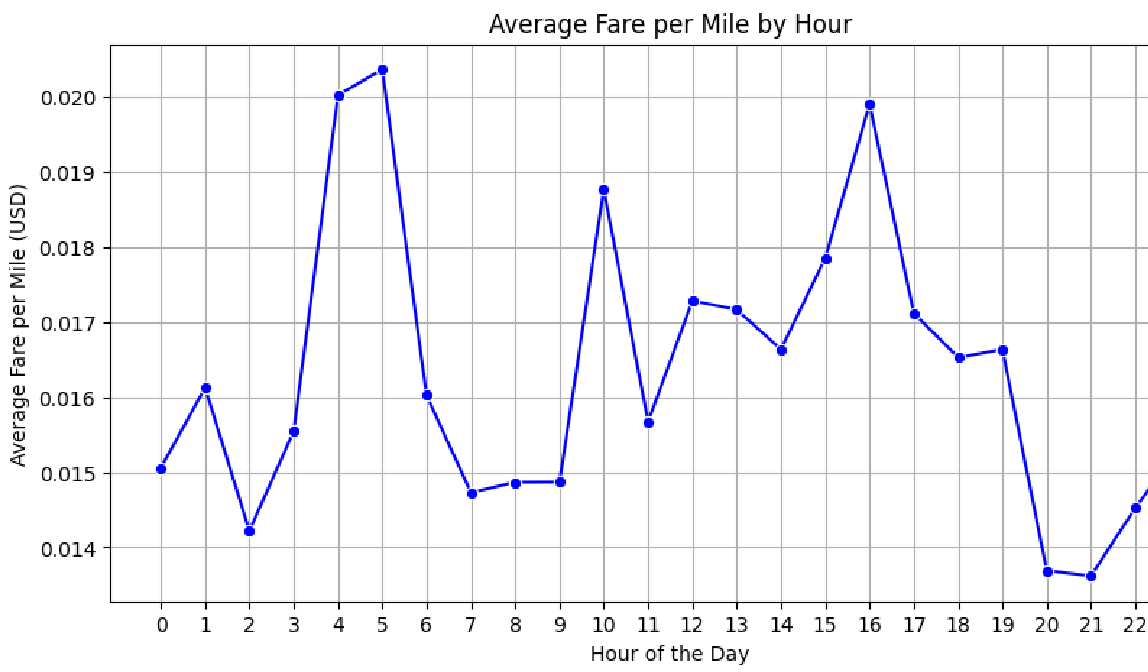
- **For the different passenger counts, find the average fare per mile per passenger**

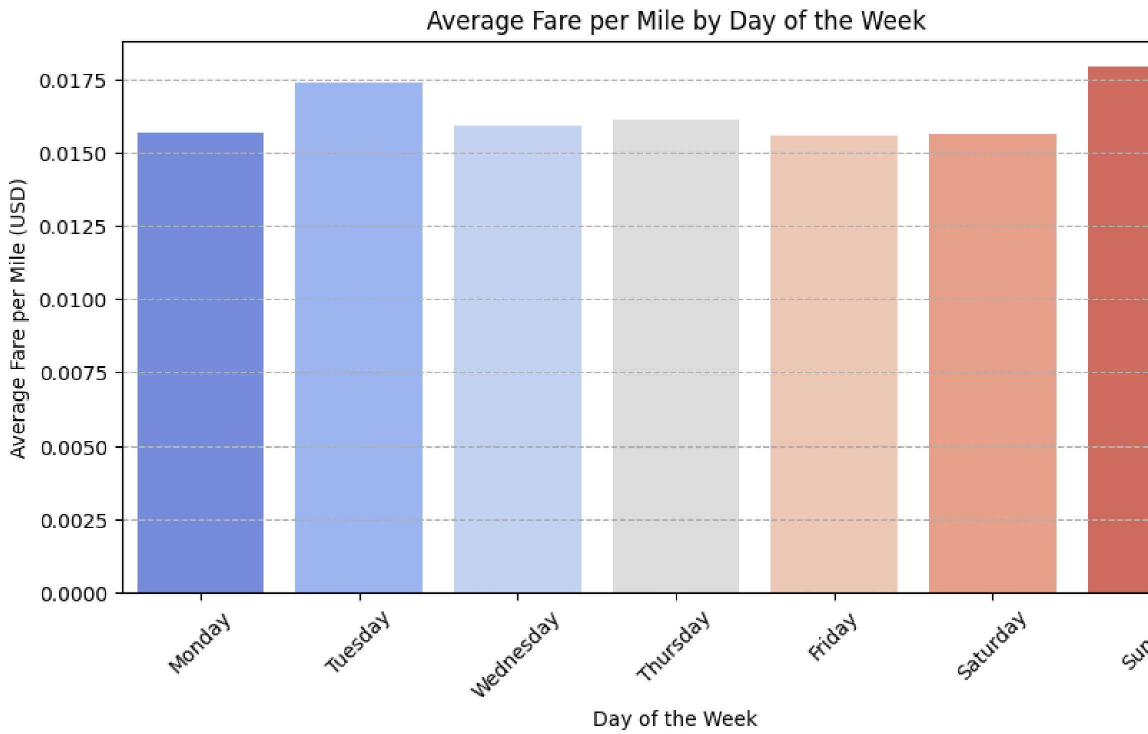
```
0             1.0                    0.015809
1             2.0                    0.009199
2             3.0                    0.005591
3             4.0                     0.00626
4             5.0                    0.002446
5             6.0                    0.001933
6             7.0                    0.001873
7             8.0                    0.040964
8             9.0                    0.108893
<ipython-input-86-63d1a76808e5>:27: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0

  sns.barplot(x=fare_per_passenger_stats["passenger_count"], y=fare_per_passenger_stat
```



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

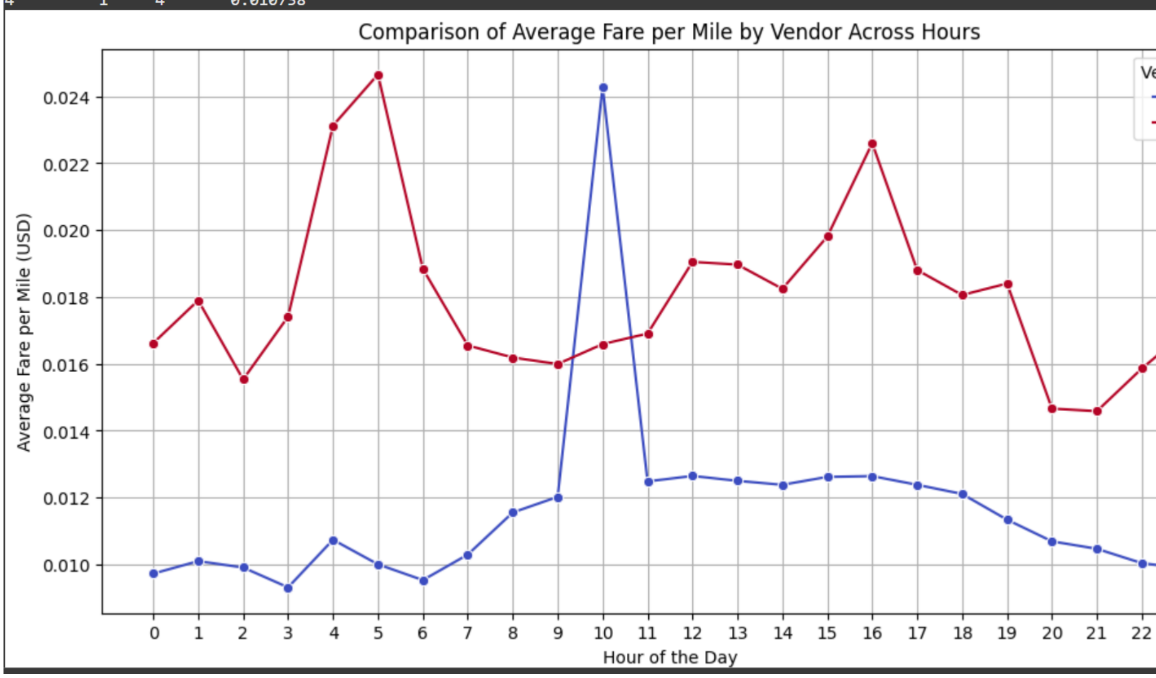
```
Average Fare per Mile by Hour:
   hour  fare_per_mile
0     0       0.015054
1     1       0.016124
2     2       0.014218
3     3       0.015545
4     4       0.020020

Average Fare per Mile by Day of the Week:
   day_of_week  fare_per_mile
0       Monday       0.015708
1      Tuesday       0.017387
2    Wednesday       0.015907
3     Thursday       0.016122
4       Friday       0.015606
5     Saturday       0.015627
6       Sunday       0.017922
```





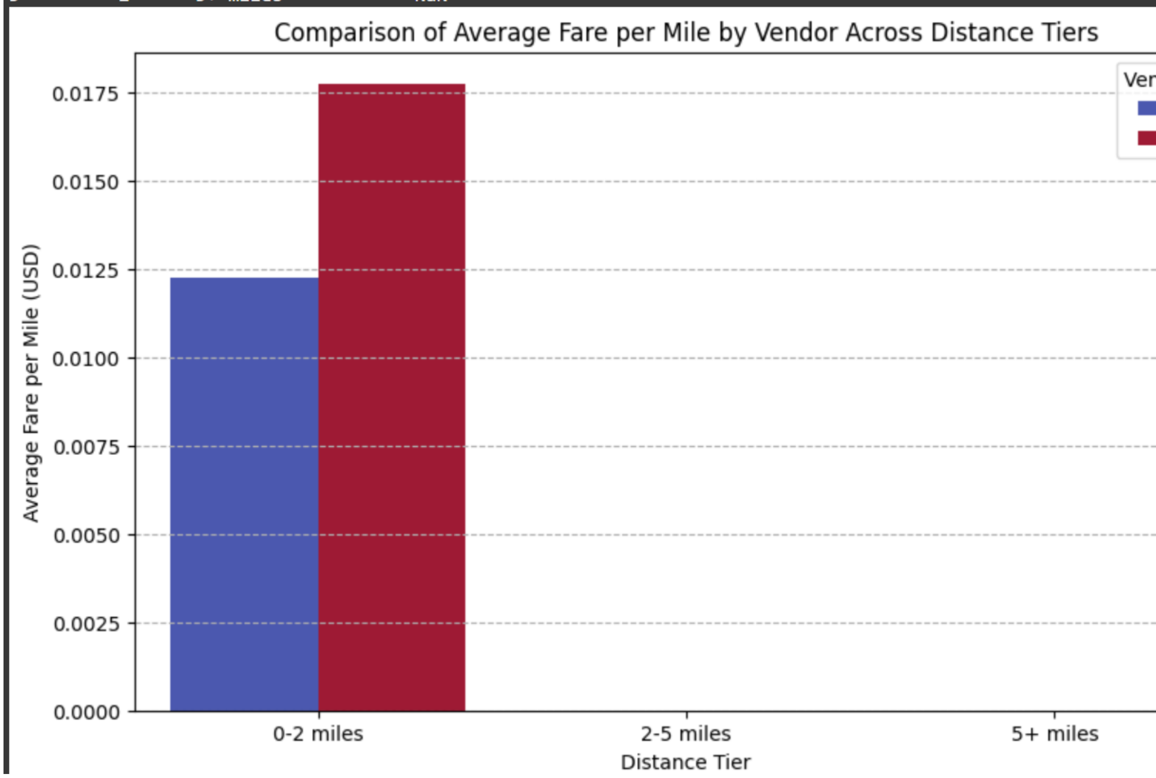
- **Analyse the average fare per mile for the different vendors**

	VendorID	hour	fare_per_mile
0	1	0	0.009724
1	1	1	0.010098
2	1	2	0.009908
3	1	3	0.009313
4	1	4	0.010738



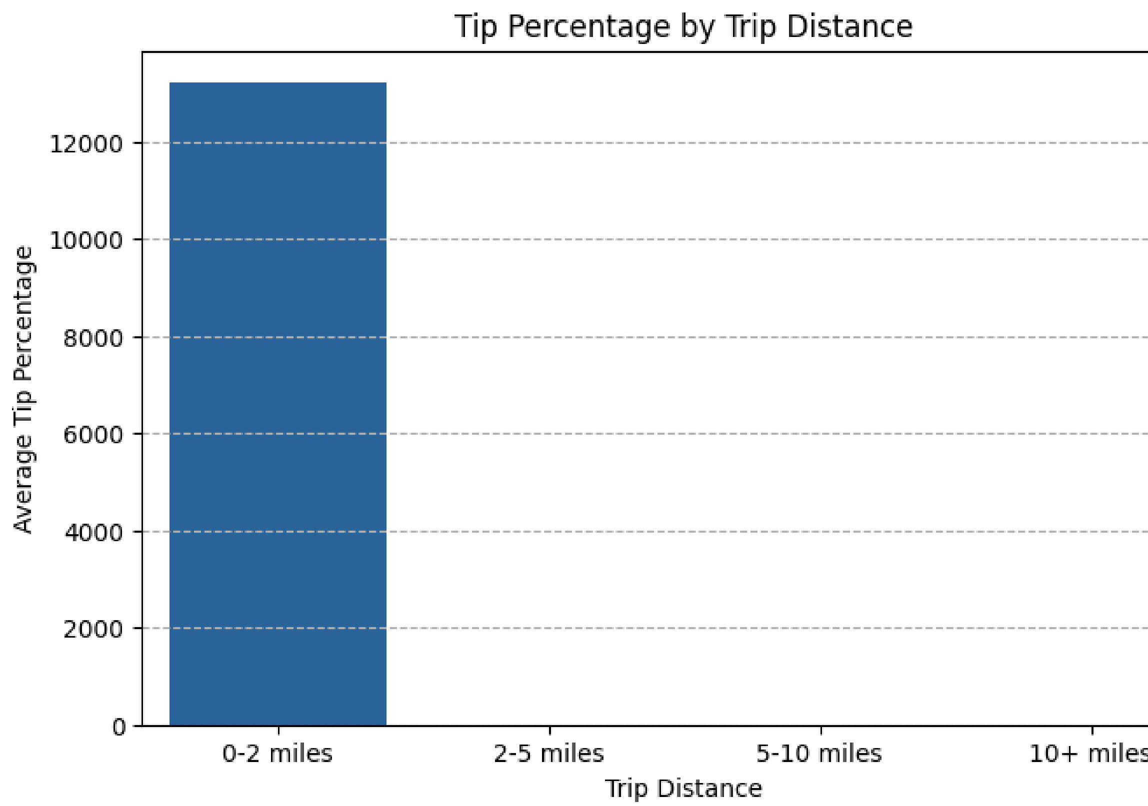
- **Compare the fare rates of different vendors in a distance-tiered fashion**

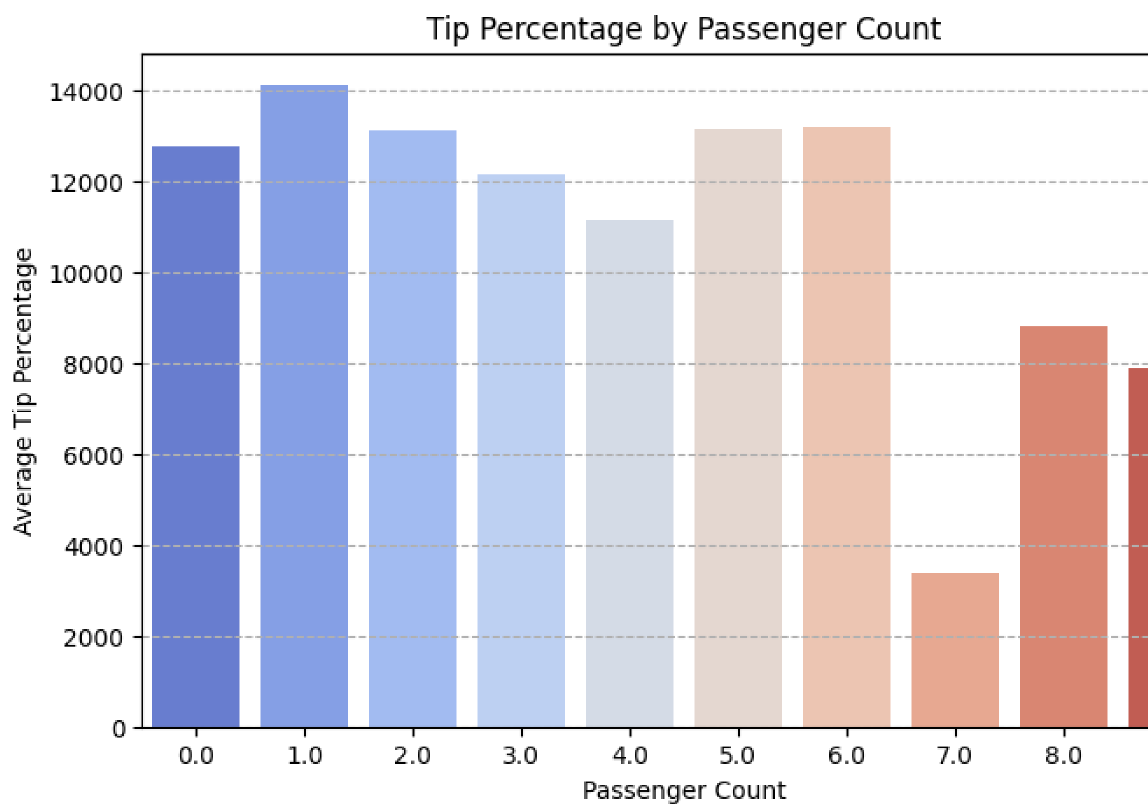
```
<ipython-input-90-34866b2b8d87>:21: FutureWarning: The default of observed=False is deprecated and will b
  vendor_fare_tiers = df_valid.groupby(["VendorID", "distance_tier"])["fare_per_mile"].mean().reset_index
   VendorID distance_tier  fare_per_mile
0         1     0-2 miles       0.012247
1         1     2-5 miles            NaN
2         1      5+ miles            NaN
3         2     0-2 miles       0.017739
4         2     2-5 miles            NaN
5         2      5+ miles            NaN
```

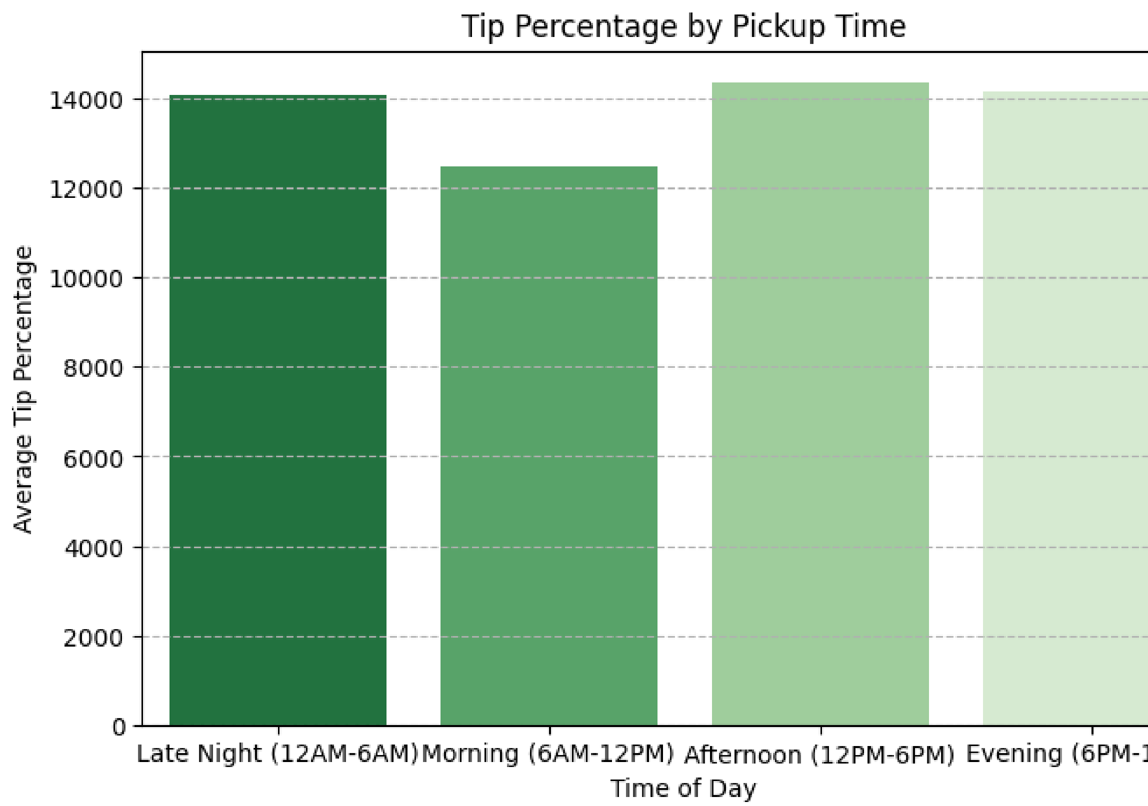


- **Analyse the tip percentages**

```
tip_by_distance = df_valid.groupby(
Tip Percentage by Distance:
  distance_category  tip_percentage
0         0-2 miles    13217.281387
1         2-5 miles             NaN
2        5-10 miles             NaN
3         10+ miles             NaN
Tip Percentage by Passenger Count:
   passenger_count  tip_percentage
0              0.0    12761.671360
1              1.0    14118.358952
2              2.0    13128.923524
3              3.0    12172.922940
4              4.0    11152.362142
5              5.0    13167.749259
6              6.0    13210.316946
7              7.0     3395.573925
8              8.0     8808.039161
9              9.0     7901.261938
```


Tip Percentage by Trip Distance
Average Tip Percentage
12000
10000
8000
6000
4000
2000
0
0-2 miles
2-5 miles
5-10 miles
10+ miles
Trip Distance

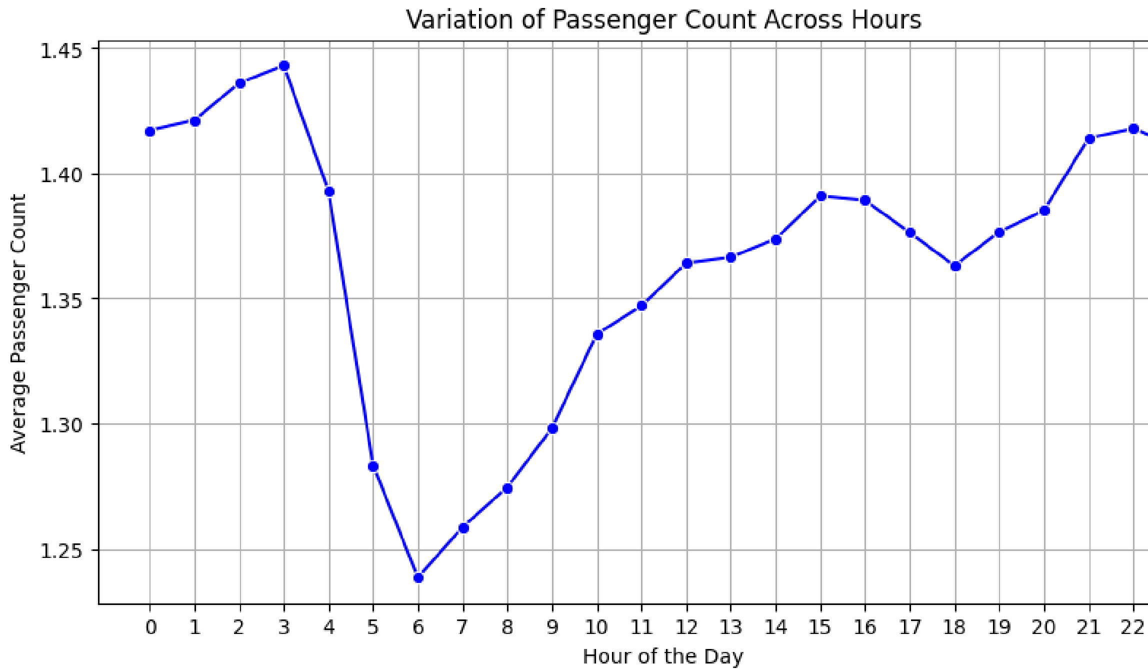

Tip Percentage by Passenger Count
Average Tip Percentage
14000
12000
10000
8000
6000
4000
2000
0
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
Passenger Count

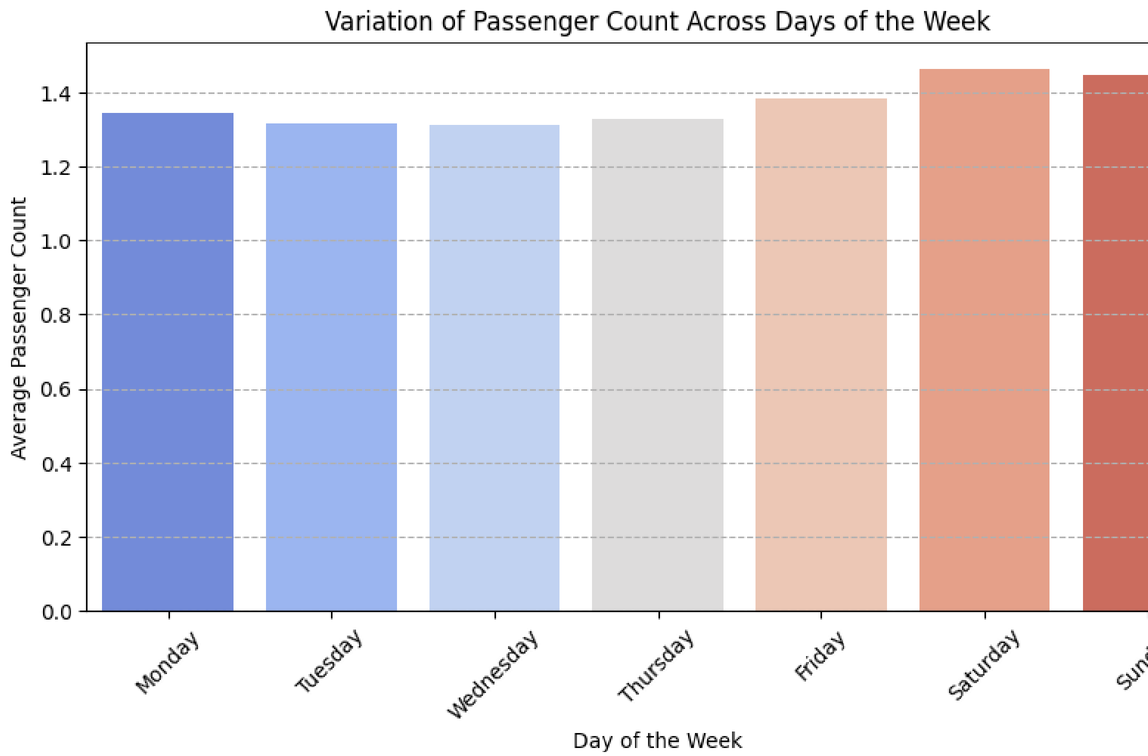


Tip Percentage by Pickup Time
14000
12000
10000
8000
6000
4000
2000
0
Average Tip Percentage
Late Night (12AM-6AM)
Morning (6AM-12PM)
Afternoon (12PM-6PM)
Evening (6PM-1
Time of Day

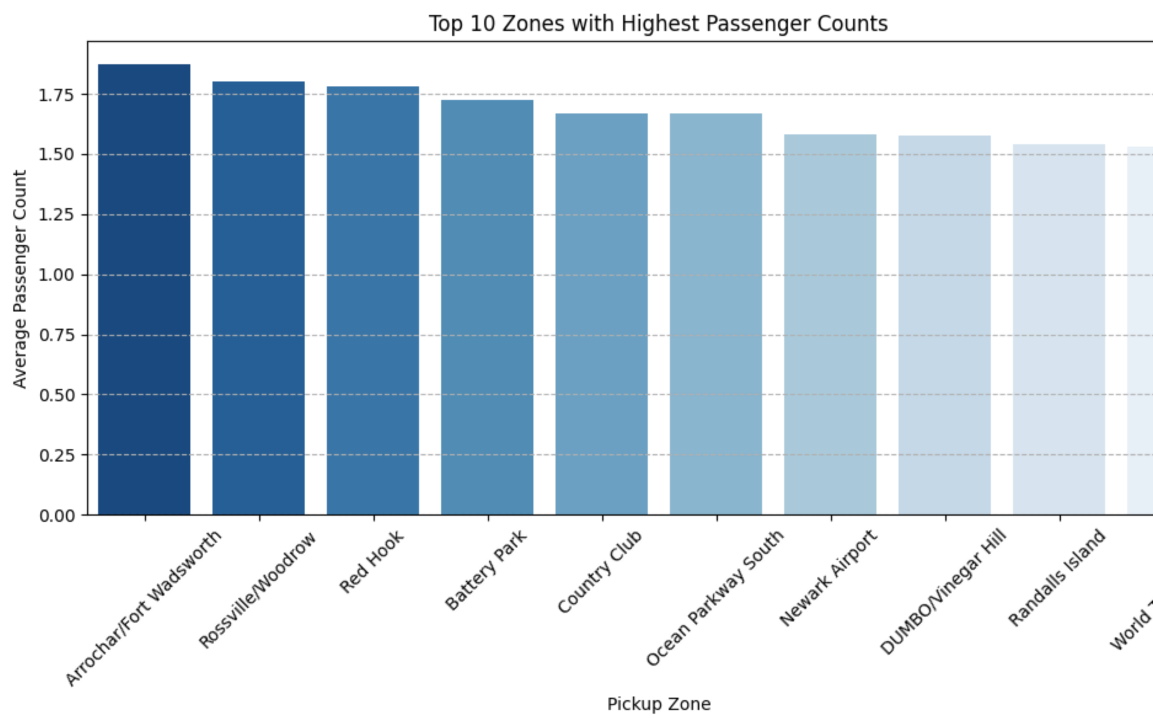
- **Analyse the trends in passenger count**

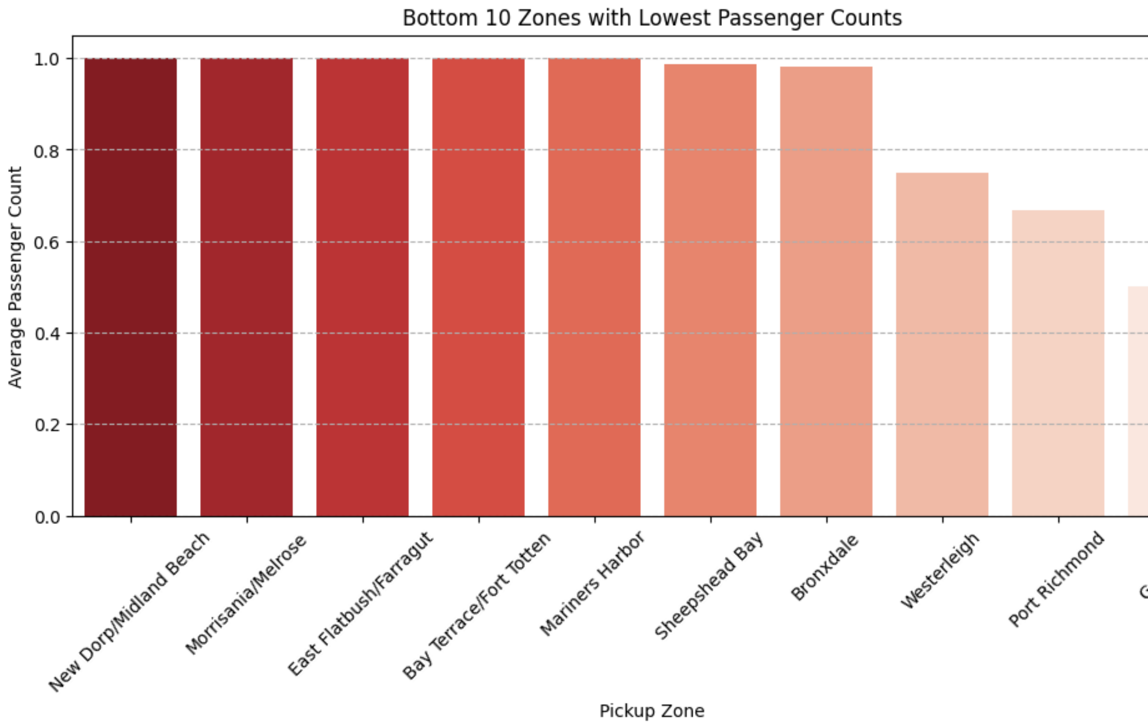
```
Average Passenger Count by Hour:
    hour  passenger_count
0      0         1.417191
1      1         1.421288
2      2         1.436037
3      3         1.443101
4      4         1.392940
Average Passenger Count by Day:
   day_of_week  passenger_count
0       Monday         1.345154
1      Tuesday         1.317328
2    Wednesday         1.313756
3     Thursday         1.327344
4       Friday         1.383877
5     Saturday         1.463231
6       Sunday         1.447616
```





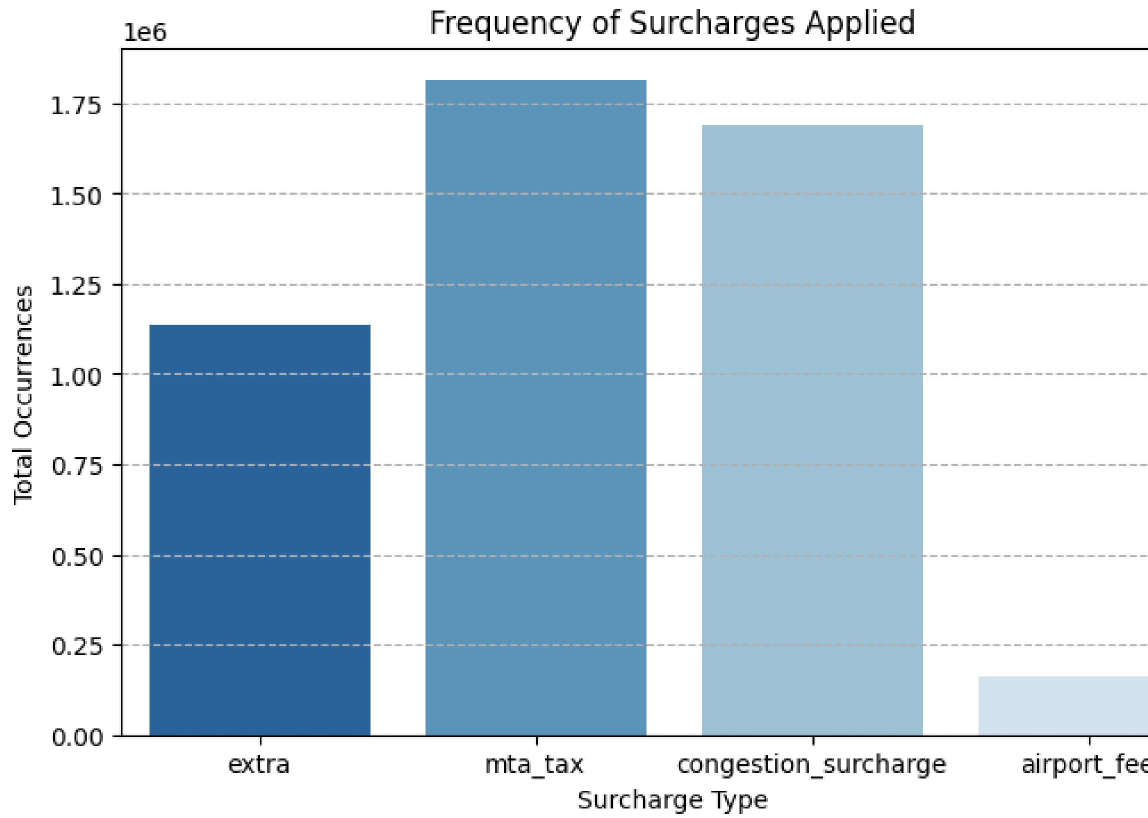
- **Analyse the variation of passenger counts across zones**

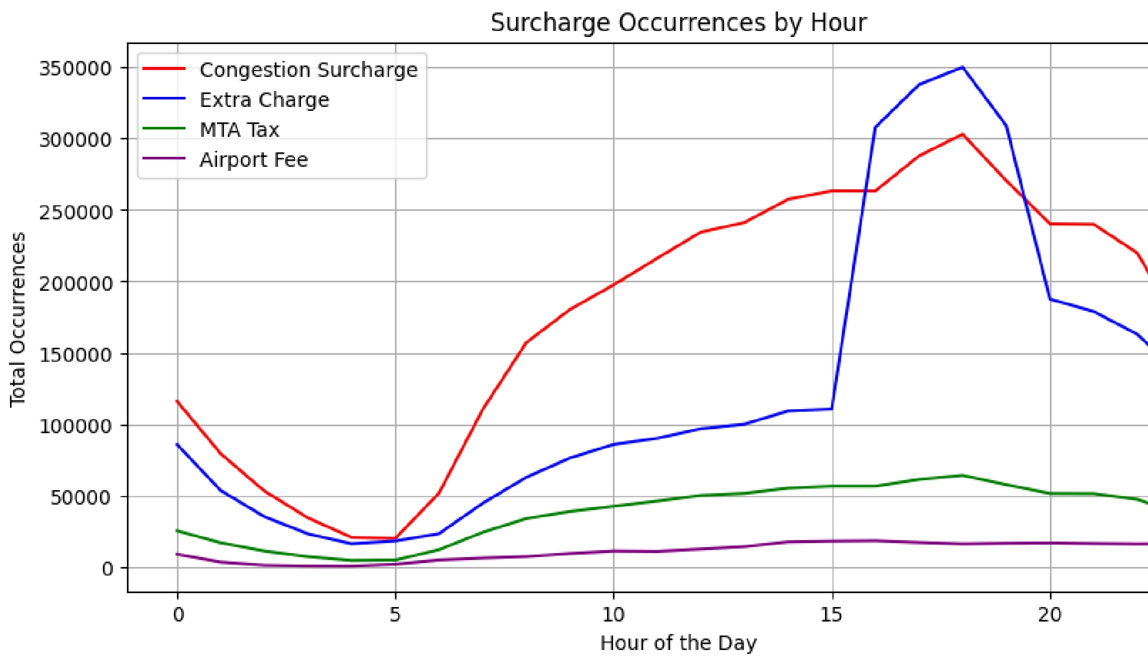




- **Analyse the pickup/dropoff zones or times when extra charges are applied more frequently.**

```
Frequency of Surcharges Applied:
                       Total Occurrences
extra                            1134248
mta_tax                          1814365
congestion_surcharge             1690260
airport_fee                       161169
Top 10 Pickup Zones with Highest Surcharge Occurrences:
      PULocationID      extra   mta_tax  congestion_surcharge  airp
133            138  403375.79  31762.10              102700.0    103
127            132  146728.95  47441.15              118930.0    146
156            161  141289.00  42606.70              213125.0
231            237  126252.48  43338.90              216635.0
230            236  109375.90  38667.60              192727.5
157            162  104607.80  32578.30              162950.0
224            230   99499.50  30103.30              150680.0
180            186   87458.67  31498.70              157642.5
137            142   90112.20  30319.00              151432.5
158            163   85838.00  26568.50              132910.0

                             zone  total_surcharge
133               LaGuardia Airport        641078.64
127                     JFK Airport        460036.10
156                  Midtown Center        397045.95
231            Upper East Side South       386237.88
230            Upper East Side North       340776.00
157                    Midtown East        300158.85
224        Times Sq/Theatre District       280327.30
180   Penn Station/Madison Sq West         276618.62
137             Lincoln Square East        271868.70
158                   Midtown North        245333.00
Surcharge Distribution by Hour of the Day:
    hour     extra  mta_tax  congestion_surcharge  airport_fee
0      0  85706.65  25322.8              115972.5      8868.50
1      1  53486.65  16906.3               79280.0      3387.50
2      2  35192.90  11100.1               53147.5      1180.75
3      3  23179.40   7178.0               34335.0       666.25
4      4  16232.70   4521.0               20680.0       581.75
<ipython-input-99-719376e69722>:45: FutureWarning:
```


Frequency of Surcharges Applied
1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Total Occurrences
extra
mta_tax
congestion_surcharge
airport_fee
Surcharge Type


Surcharge Occurrences by Hour
Congestion Surcharge
Extra Charge
MTA Tax
Airport Fee
350000
300000
250000
200000
150000
100000
50000
0
Total Occurrences
0
5
10
15
20
Hour of the Day

4. Conclusions

4.1. Final Insights and Recommendations

Recommendations to Optimize Routing and Dispatching

- **Increase Taxi Availability in High-Demand Areas**
  - Deploy more taxis in zones like Midtown, Financial District, and Airports during peak hours, and in nightlife areas like Times Square and Brooklyn late at night.
- **Adjust Fleet Allocation Based on One-Way Demand**
  - Dispatch more taxis to residential areas in the morning and monitor dropoff-heavy locations to redistribute taxis accordingly.
- **Use Surge Pricing to Encourage More Drivers**
  - Implement higher fares during peak hours and high-demand areas. Offer incentives for drivers to work during low-tip periods or in low-supply zones.
- **Optimize Routes to Reduce Traffic Delays**
  - Reroute taxis from congested roads using real-time traffic data to suggest faster routes.
- **Encourage Ride-Sharing in Low-Pickup Zones**
  - Promote shared rides in residential and suburban areas with low single-passenger demand. Offer discounts for shared rides during high-demand periods.
- **Improve Dispatching Efficiency with Data Insights**
  - Analyze peak demand hours and evenly distribute taxis across zones. Adjust driver schedules based on demand patterns to maximize efficiency.

These strategies will enhance customer wait times, increase driver earnings, and improve overall efficiency.

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

Strategic Positioning of Cabs Based on Trip Trends

- **Midtown and Financial District: High Demand During Weekday Mornings and Evenings**
  - Position more taxis in these areas from 7 AM to 10 AM and 4 PM to 7 PM for commuters.
  - Maintain availability near major office hubs like Wall Street, Penn Station, and Grand Central Terminal.
- **Airports (JFK, LGA, EWR): High Demand for Early Morning and Late-Night Flights**
  - Allocate taxis near airports between 4 AM - 8 AM and 9 PM - 1 AM when flight arrivals peak.
  - Monitor dropoff rates to ensure enough cabs return to the airport after long trips.
- **Nightlife Areas (Times Square, East Village, Brooklyn): Peak Demand on Weekends**
  - Increase the number of taxis near bars, clubs, and entertainment areas from 10 PM - 3 AM on Fridays and Saturdays.
  - Assign more taxis to dropoff-heavy zones like Brooklyn and Queens for return trips.

- **Residential Areas: High Demand for Morning Commutes and Evening Returns**
  - Position more cabs in residential neighborhoods from 6 AM - 9 AM for work commutes.
  - Ensure taxis are available for return trips from 5 PM - 8 PM when people head home.
- **Tourist Hotspots: Steady Demand Throughout the Day**
  - Keep a steady supply of taxis near locations like Central Park, Empire State Building, and Broadway shows.
  - Monitor seasonal demand changes to adjust fleet distribution accordingly.
- **Event Venues and Stadiums: High Demand Around Event Timings**
  - Assign taxis near Madison Square Garden, Yankee Stadium, and Barclays Center before and after events.
  - Increase dispatch availability 30 minutes before event start times and 1 hour after events end.
- **Late-Night and Early-Morning Demand Areas**
  - Position cabs near hospitals, train stations, and 24-hour businesses to serve late-night travelers.
  - Ensure coverage in areas with fewer public transport options at night.

By strategically placing taxis based on demand patterns, services can reduce wait times, increase efficiency, and maximize driver earnings.

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

Data-Driven Pricing Strategy Adjustments

- **Dynamic Pricing for Peak Demand Hours**

- Implement increased base fares during peak hours (7 AM - 10 AM and 5 PM - 8 PM) when demand is at its highest.
- Apply elevated per-mile rates on Friday and Saturday nights (10 PM - 3 AM) to capitalize on nightlife demand.

- **Lower Base Fares for Short Trips to Increase Volume**

- Decrease fares for trips under 2 miles to attract more riders, particularly in congested areas where taxis compete with public transportation.
- Introduce a flat-fee model for short distances within high-traffic zones such as Midtown.

- **Incentivize Long-Distance Trips with Discounts**

- Provide lower per-mile rates for rides exceeding 5 miles to make long trips more appealing.
- Offer discounts for airport rides during off-peak hours (midday and late-night) to encourage additional bookings.

- **Surge Pricing in High-Demand Locations**

- Dynamically increase fares near airports, stadiums, and event venues before and after significant events.

- Monitor ride requests in tourist-heavy areas and adjust pricing based on real-time demand.
   - **Reward Frequent Riders with Promotions**

- Implement loyalty discounts for returning customers based on their ride frequency.
- Offer fare discounts during low-demand hours (11 AM - 3 PM) to boost utilization.
   - **Encourage Digital Payments with Small Discounts**

- Given that credit card payments typically result in higher tipping rates, offer a 1-2% fare discount for cashless transactions to increase their adoption.
   - **Optimize Congestion & Extra Charges**

- Adjust congestion surcharge timing to reflect actual traffic patterns rather than fixed hours.
- Offer free or reduced night surcharges for trips exceeding 10 miles to promote more bookings.

By leveraging data-driven pricing adjustments, taxi services can enhance revenue, improve customer satisfaction, and maintain competitive standing with ride-sharing platforms.