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

1.1.1. Importing Libraries

```python
# Import warnings
import warnings
warnings.filterwarnings("ignore")
```

```python
# Import the libraries you will be using for analysis
import numpy as np
import pandas as pd

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

1.1.2. Sample the data and combine the files

```python
import os
import pandas as pd
from datetime import datetime

# Create a list of all the twelve files to read
file_list = os.listdir()

# Initialize an empty dataframe
df = pd.DataFrame()

# 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)

        # Reading the current file
        monthly_data = pd.read_parquet(file_path)  # Assuming the files are in parquet format

        # Convert the pickup datetime to a datetime object if it's not already
        monthly_data['tpep_pickup_datetime'] = pd.to_datetime(monthly_data['tpep_pickup_datetime'])

        # We will store the sampled data for the current month in this df by appending the sampled data from each hour to this
        sampled_data = pd.DataFrame()

        # Loop through dates and then loop through every hour of each date
        for date, day_data in monthly_data.groupby(monthly_data['tpep_pickup_datetime'].dt.date):
            for hour, hour_data in day_data.groupby(day_data['tpep_pickup_datetime'].dt.hour):
                # Sample 5% of the hourly data randomly
                sampled_hour_data = hour_data.sample(frac=0.008, random_state=42)

                # Add data of this hour to the dataframe
                sampled_data = pd.concat([sampled_data, sampled_hour_data])
```

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

    except Exception as e:
        print(f"Error reading file {file_name}: {e}")

# Save the final dataframe to a file if needed
df.to_parquet('sampled_NYC_Taxi_data.parquet', index=False)

print("Sampling completed and data saved to 'sampled_NYC_Taxi_data.parquet'")
```

df = pd.read_parquet('sampled_NYC_Taxi_data.parquet')

2. Data Cleaning

2.1. Fixing Columns

2.1.1. Fix the index

```
# Fix the index and drop any columns that are not needed
# List of columns to drop
columns_to_drop = ['store_and_fwd_flag']

# Drop the columns
df = df.drop(columns=columns_to_drop)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303397 entries, 0 to 303396
Data columns (total 18 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   VendorID               303397 non-null  int64
 1   tpep_pickup_datetime   303397 non-null  datetime64[us]
 2   tpep_dropoff_datetime  303397 non-null  datetime64[us]
 3   passenger_count        293215 non-null  float64
 4   trip_distance          303397 non-null  float64
 5   RatecodeID             293215 non-null  float64
 6   PULocationID           303397 non-null  int64
 7   DOLocationID           303397 non-null  int64
 8   payment_type           303397 non-null  int64
 9   fare_amount            303397 non-null  float64
 10  extra                  303397 non-null  float64
 11  mta_tax                303397 non-null  float64
 12  tip_amount             303397 non-null  float64
 13  tolls_amount           303397 non-null  float64
 14  improvement_surcharge  303397 non-null  float64
 15  total_amount           303397 non-null  float64
 16  congestion_surcharge   293215 non-null  float64
 17  airport_fee_merged     293215 non-null  float64
dtypes: datetime64[us](2), float64(12), int64(4)
memory usage: 41.7 MB
```

2.1.2. Combine the two airport_fee columns

```python
# Combine the two airport fee columns
def merge_columns_manual(df, col1, col2, new_col_name):
    """
    Merges two columns into a single column, prioritizing non-NaN values from col1.

    Parameters:
        df (pd.DataFrame): The DataFrame containing the columns.
        col1 (str): The name of the first column.
        col2 (str): The name of the second column.
        new_col_name (str): The name of the new merged column.

    Returns:
        pd.DataFrame: The DataFrame with the merged column.
    """
    # Create the new merged column
    df[new_col_name] = np.where(
        df[col1].isna(),  # Condition: Check if col1 has NaN
        df[col2],         # If col1 has NaN, use col2
        df[col1]          # Otherwise, use col1
    )

    # Drop the original columns
    df = df.drop(columns=[col1, col2])

    return df

df = merge_columns_manual(df, col1='airport_fee', col2='Airport_fee', new_col_name='airport_fee_merged')
```

2.2. Handling Missing Values

2.2.1. Find the proportion of missing values in each column

```python
# Find which columns have negative values

# Initialize a dictionary to store the count of negative values for each column
negative_counts = {}

# Loop through each column in the DataFrame
for column in df.columns:
    # Check if the column is numeric (to avoid errors with non-numeric columns)
    if pd.api.types.is_numeric_dtype(df[column]):
        # Count the number of negative values in the column
        negative_count = (df[column] < 0).sum()
        if negative_count > 0:
            negative_counts[column] = negative_count

# Print the results
if negative_counts:
    print("Columns with negative values and their counts:")
    for column, count in negative_counts.items():
        print(f"{column}: {count} negative rows")
else:
    print("No columns have negative values.")

# Analyze negative values by RateCodeID
for column in negative_counts:
    print(f"\nNegative values in {column} by RateCodeID:")
    print(df[df[column] < 0].groupby('RatecodeID').size())
```

```python
# Find the proportion of missing values in each column

# Calculate the proportion of missing values in each column
missing_proportions = df.isna().mean()

# Print the results
print("Proportion of missing values in each column:")
print(missing_proportions)
```

```
Proportion of missing values in each column:
VendorID                 0.000000
tpep_pickup_datetime     0.000000
tpep_dropoff_datetime    0.000000
passenger_count          0.033561
trip_distance            0.000000
RatecodeID               0.033561
PULocationID             0.000000
DOLocationID             0.000000
payment_type             0.000000
fare_amount              0.000000
extra                    0.000000
mta_tax                  0.000000
tip_amount               0.000000
tolls_amount             0.000000
improvement_surcharge    0.000000
total_amount             0.000000
congestion_surcharge     0.033561
airport_fee_merged       0.033561
dtype: float64
```

2.2.2. Handling missing values in passenger_count

```python
median_value = df['passenger_count'].median()
df['passenger_count'] = df['passenger_count'].fillna(median_value)
df['passenger_count'] = df['passenger_count'].replace(0, 1)
```

2.2.3. Handle missing values in RatecodeID

```python
median_value_rate = df['RatecodeID'].median()
df['RatecodeID'] = df['RatecodeID'].fillna(median_value_rate)
```

2.2.4. Impute NaN in congestion_surcharge

```python
df['congestion_surcharge'] = df['congestion_surcharge'].fillna(0)
```

2.3. Handling Outliers and Standardising Values

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

```
df['RatecodeID'] = df['RatecodeID'].replace(99, float(df.RatecodeID.mode()))
df.RatecodeID.value_counts()
```

```
# Select numeric columns for boxplots
numeric_columns = df.select_dtypes(include=['int', 'float']).columns

# Calculate the number of rows and columns for subplots
num_columns = len(numeric_columns)
num_rows = (num_columns // 4) + (1 if num_columns % 4 != 0 else 0)  # Adjust rows
based on number of columns

# Plot boxplots for numeric columns
plt.figure(figsize=(15, 5 * num_rows))  # Adjust figure size based on number of
rows
for i, column in enumerate(numeric_columns, 1):
    plt.subplot(num_rows, 4, i)  # Dynamically adjust subplot layout
    sns.boxplot(y=df[column])
    plt.title(f"Boxplot of {column}")
plt.tight_layout()
plt.show()

# Step 0: Remove entries where passenger_count > 6
df = df[df['passenger_count'] <= 6]
# Continue with outlier handling

# Step 1: Remove entries where trip_distance is nearly 0 and fare_amount is more
than 300
df = df[~((df['trip_distance'] < 0.1) & (df['fare_amount'] > 300))]

# Step 2: Remove entries where trip_distance and fare_amount are 0 but
PULocationID and DOLocationID are different
df = df[~((df['trip_distance'] == 0) & (df['PULocationID'] !=
df['DOLocationID']))]

df = df[~((df['trip_distance'] == 0) & (df['fare_amount'] == 0))]



# Step 3: Remove entries where trip_distance is more than 250 miles
df = df[df['trip_distance'] <= 250]
```

```python
# Step 4: Remove entries where payment_type is 0
df = df[df['payment_type'] != 0]

df = df[~(df.fare_amount>1000)]

# Removing the outliers

# Step 1: Identify the rows to drop
rows_to_drop = df[(df['trip_distance'] == 0) & (df['PULocationID'] ==
df['DOLocationID'])]
print(f"Number of rows to drop: {len(rows_to_drop)}")

# Step 2: Drop the rows
df_cleaned = df.drop(rows_to_drop.index)

# Step 3: Verify the results
print(f"Number of rows before dropping: {len(df)}")
print(f"Number of rows after dropping: {len(df_cleaned)}")
print(f"Rows dropped: {len(df) - len(df_cleaned)}")


df_cleaned = df[(df['fare_amount'] > 0) &
                (df['trip_distance'] > 0) &
                (df['tip_amount'] >= 0)]
```

3. Exploratory Data Analysis

3.1. General EDA: Finding Patterns and Trends

Classify variables into categorical and numerical

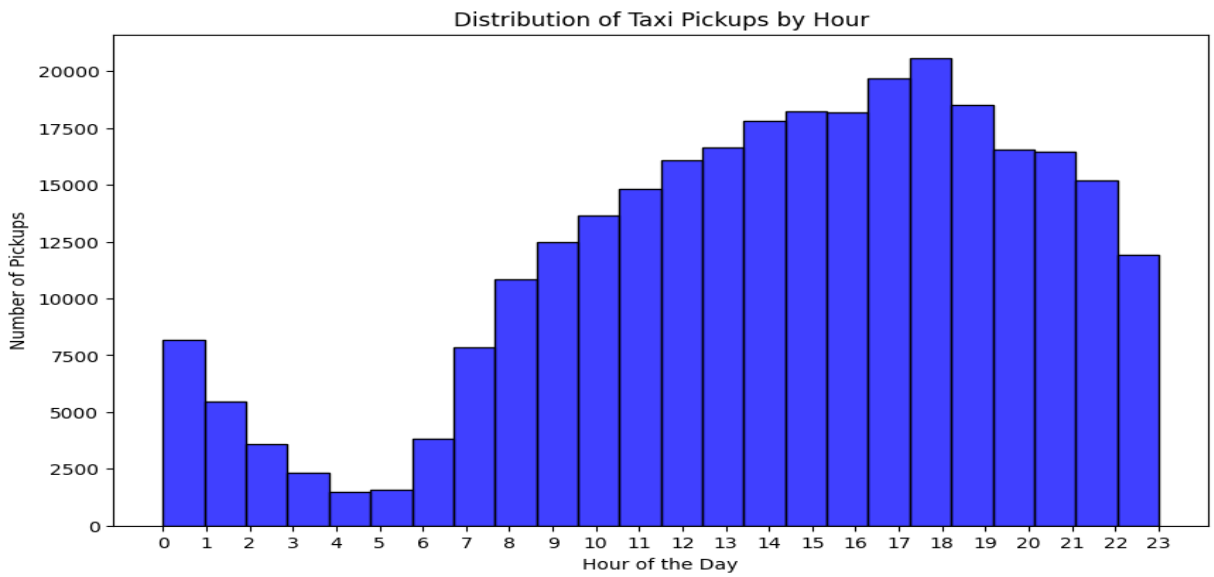
```
| Variable  | Type  | Subtype   | Description |
| --- | --- | --- | --- |
| VendorID  | Categorical   | Nominal   | TPEP provider code (1 or 2). |
| tpep_pickup_datetime  | Date/Time | - | Date and time when the meter was
engaged.|
| tpep_dropoff_datetime | Date/Time | - | Date and time when the meter was
disengaged.|
| passenger_count   | Numerical | Discrete  | Number of passengers in the
vehicle.|
| trip_distance | Numerical | Continuous    | Elapsed trip distance in miles.|
| PULocationID  | Categorical   | Nominal   | TLC Taxi Zone where the taximeter
was engaged.|
```

```
| DOLocationID  | Categorical   | Nominal   | TLC Taxi Zone where the taximeter
was disengaged.|
| RateCodeID    | Categorical   | Ordinal   | Final rate code in effect at the
end of the trip.|
| store_and_fwd_flag    | Categorical | Nominal | Flag indicating whether the
trip record was held in vehicle memory.|
| payment_type  | Categorical | Nominal | Numeric code signifying how the
passenger paid for the trip.|
| fare_amount | Numerical | Continuous |    Time-and-distance fare calculated by
the meter.|
| extra |   Numerical | Continuous |    Miscellaneous extras and surcharges.|
| mta_tax | Numerical | Continuous |    $0.50 MTA tax.|
| tip_amount |  Numerical | Continuous |    Tip amount.|
| tolls_amount |    Numerical | Continuous |    Total amount of all tolls paid.|
| improvement_surcharge |   Numerical | Continuous |    $0.30 improvement
surcharge.|
| total_amount |    Numerical | Continuous |    Total amount charged to
passengers.|
| congestion_surcharge |    Numerical | Continuous |    Total amount collected
for NYS congestion surcharge.|
| airport_fee | Numerical | Continuous |    $1.25 for pick up at LaGuardia and
JFK airports.|
```

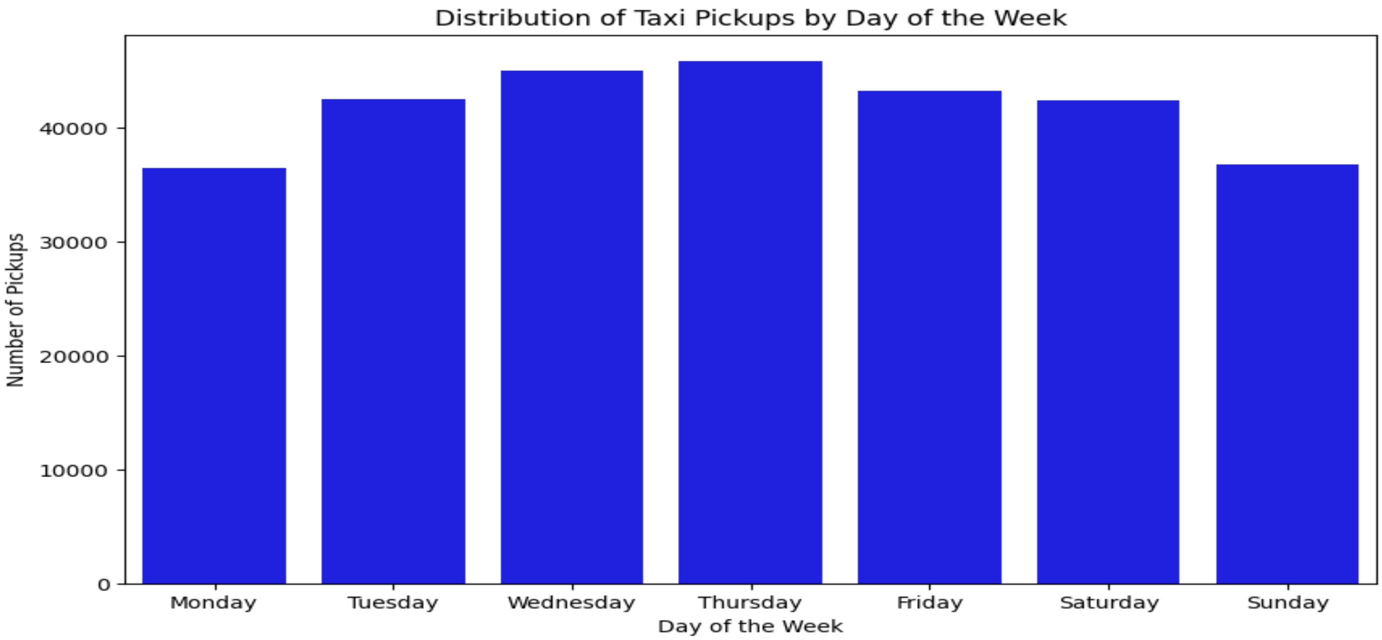
3.1.1. Analyse the distribution of taxi pickups by hours, days of the week, and months

```
# Find and show the hourly trends in taxi pickups

# Convert tpep_pickup_datetime to datetime format
df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])

# Extract hour, day of the week, and month
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour
df['pickup_day'] = df['tpep_pickup_datetime'].dt.day_name()  # Full day name
(e.g., Monday)
df['pickup_month'] = df['tpep_pickup_datetime'].dt.month_name()  # Full month
name (e.g., January)

# Plot distribution by hour
plt.figure(figsize=(10, 6))
sns.histplot(df['pickup_hour'], bins=24, kde=False, color='blue')
plt.title('Distribution of Taxi Pickups by Hour')
plt.xlabel('Hour of the Day')
```

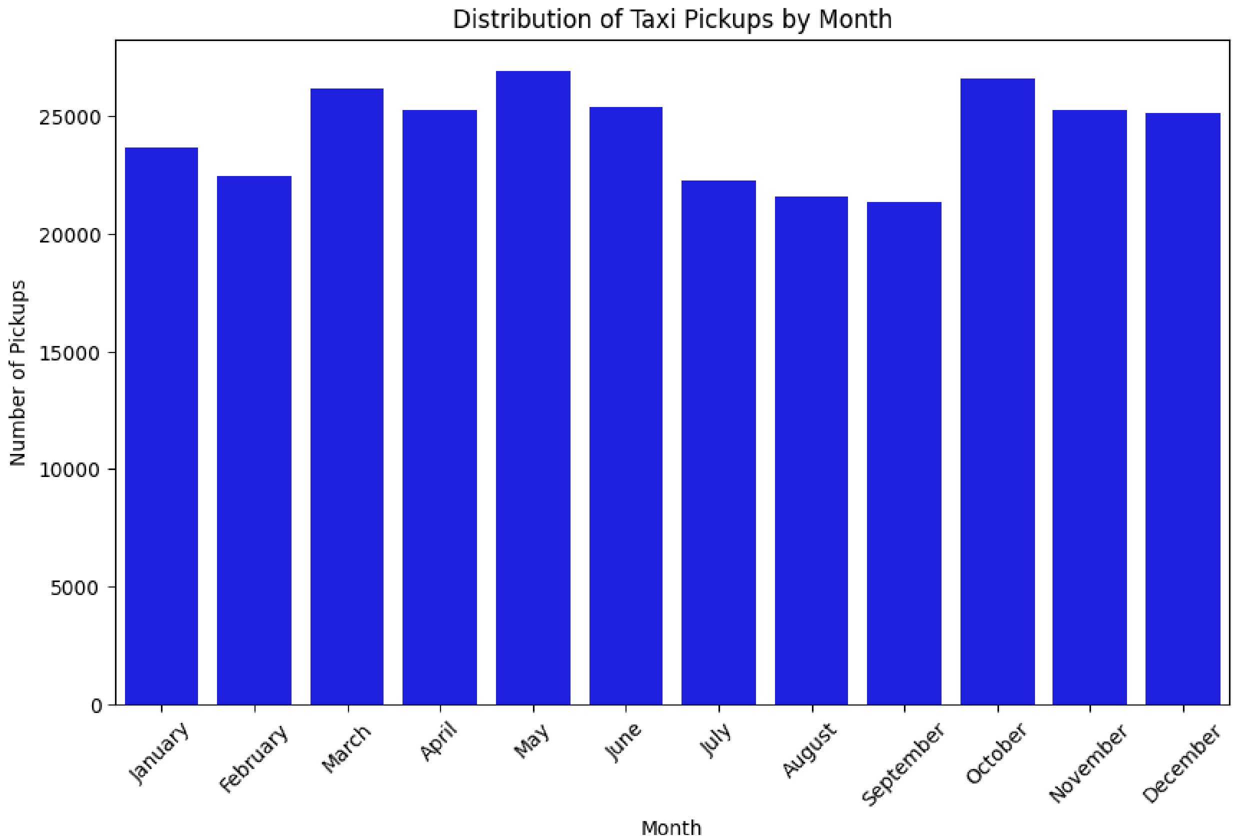
```
plt.ylabel('Number of Pickups')
plt.xticks(range(0, 24))  # Ensure all hours are displayed
plt.show()
```




```
# Find and show the daily trends in taxi pickups (days of the week)

# Order days of the week for proper plotting
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'Sunday']

# Plot distribution by day of the week
plt.figure(figsize=(10, 6))
sns.countplot(x=df['pickup_day'], order=day_order, color='blue')
plt.title('Distribution of Taxi Pickups by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Pickups')
plt.show()
```




```python
# Show the monthly trends in pickups

# Order months for proper plotting
month_order = ['January', 'February', 'March', 'April', 'May', 'June',
               'July', 'August', 'September', 'October', 'November', 'December']

# Plot distribution by month
plt.figure(figsize=(10, 6))
sns.countplot(x=df['pickup_month'], order=month_order, color='blue')
plt.title('Distribution of Taxi Pickups by Month')
plt.xlabel('Month')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.show()
```

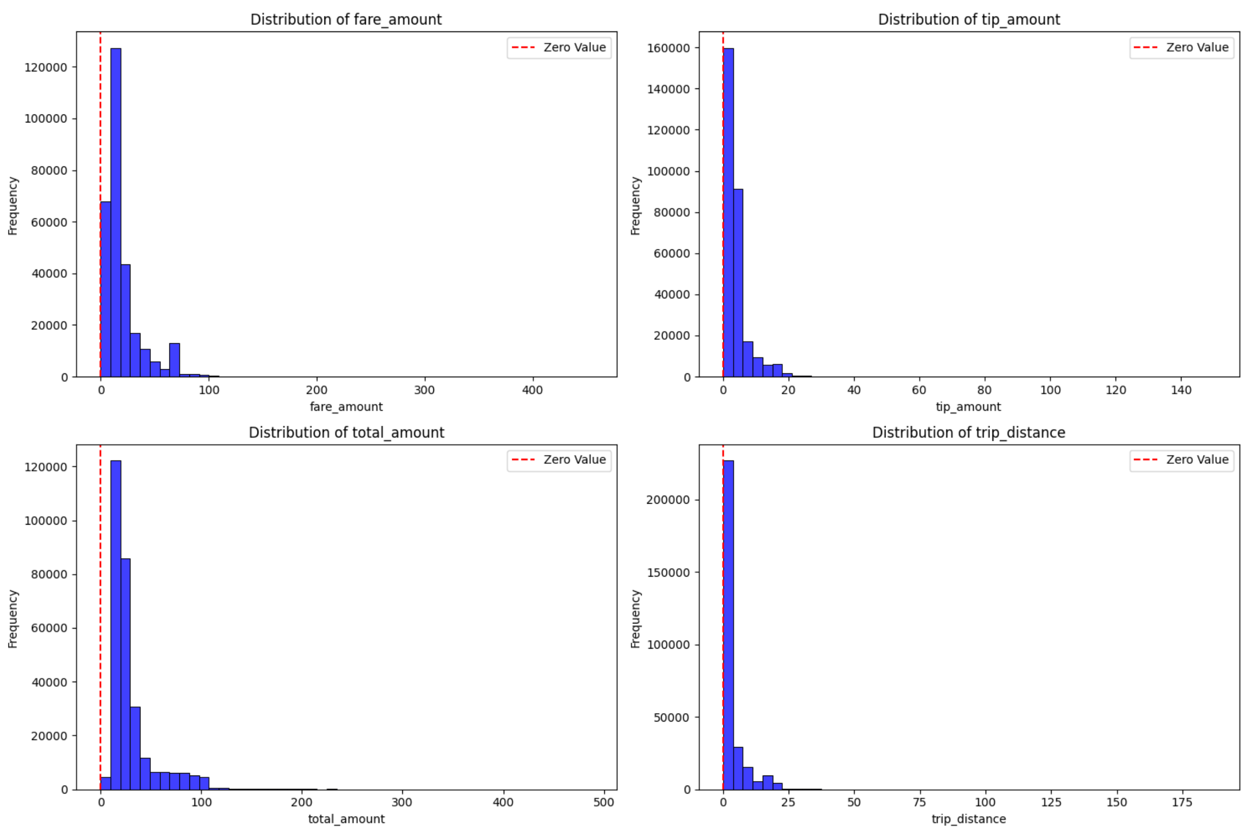


3.1.2. Filter out the zero/negative values in fares, distance and tips

```
# Analyse the above parameters

# List of columns to analyze
columns_to_check = ['fare_amount', 'tip_amount', 'total_amount',
'trip_distance']

# Step 1: Check for zero and negative values
for column in columns_to_check:
    zero_count = (df[column] == 0).sum()
    negative_count = (df[column] < 0).sum()
    total_count = len(df[column])

    print(f"Column: {column}")
    print(f"Zero values: {zero_count} ({(zero_count / total_count) *
100:.2f}%)")
    print(f"Negative values: {negative_count} ({(negative_count /
total_count) * 100:.2f}%)")
    print("\n")
```

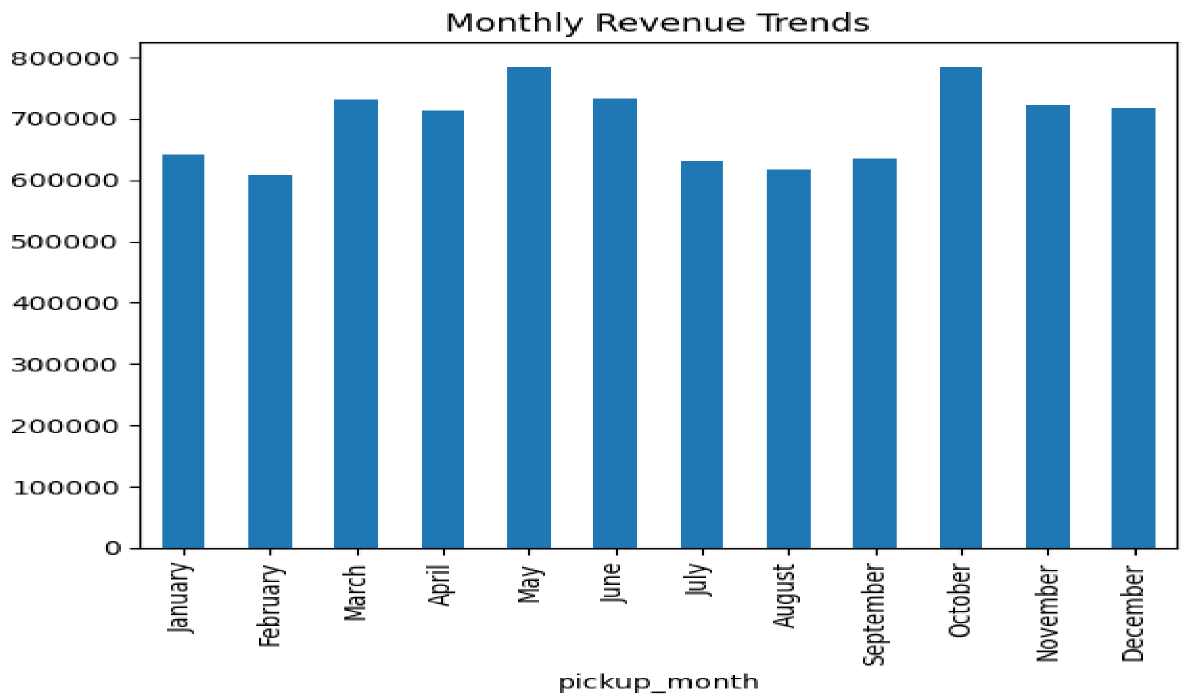
```python
columns_to_check = [
    'extra',
    'mta_tax',
    'tolls_amount',
    'improvement_surcharge',
    'total_amount',
    'congestion_surcharge',
    'airport_fee_merged'
]

# Remove rows with negative values in any of the specified columns
df = df[~((df[columns_to_check] < 0).any(axis=1))]
```

3.1.3. Analyse the monthly revenue trends




```python
monthly_revenue =
df_cleaned.groupby('pickup_month')['total_amount'].sum().reindex(month_order)
monthly_revenue.plot(kind='bar')
```

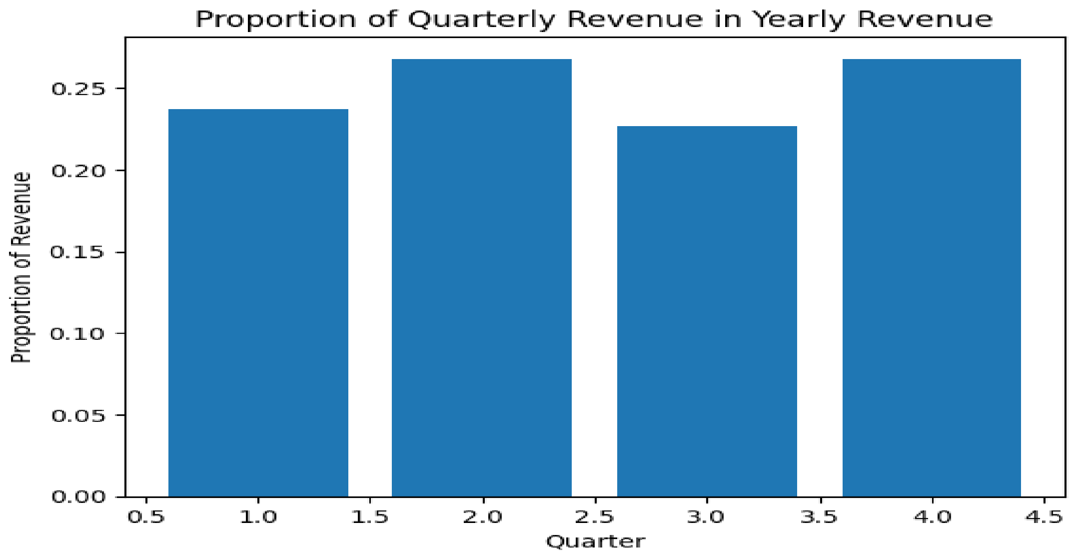
```
plt.title('Monthly Revenue Trends')
plt.show()
```



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

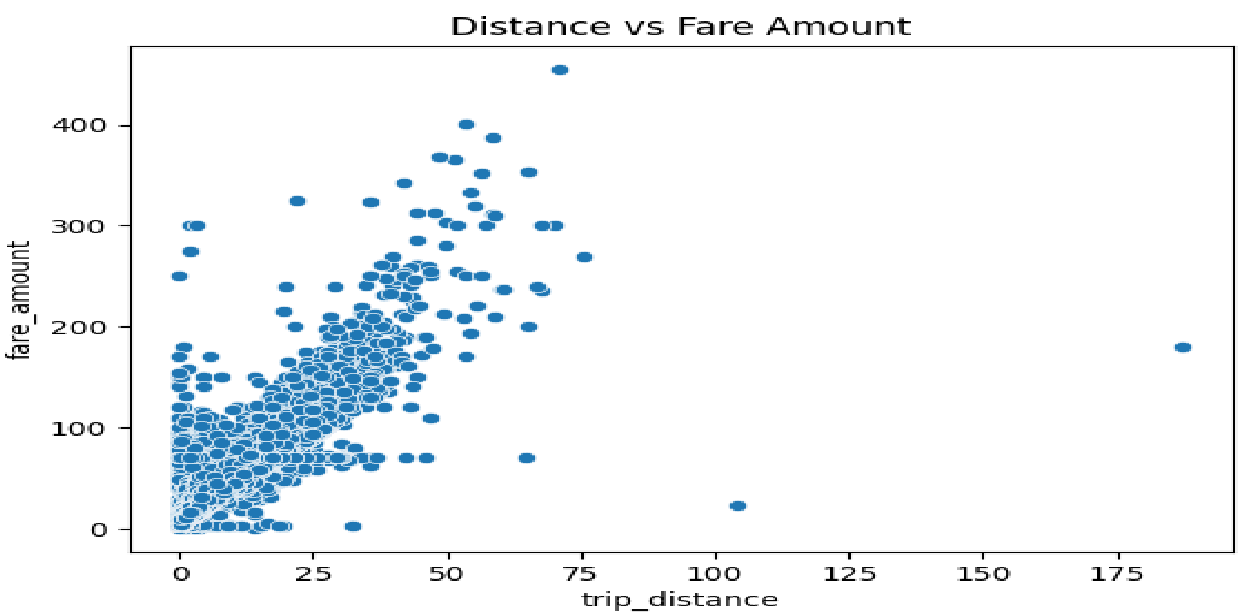
```
# Calculate proportion of each quarter

df_cleaned['pickup_quarter'] = df_cleaned['tpep_pickup_datetime'].dt.quarter
quarterly_revenue =
df_cleaned.groupby('pickup_quarter')['total_amount'].sum()
quarterly_revenue_proportion = quarterly_revenue / quarterly_revenue.sum()
print(quarterly_revenue_proportion)
```

```
pickup_quarter
1    0.237985
2    0.268170
3    0.226337
4    0.267507

Name: total_amount, dtype: float64
```



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

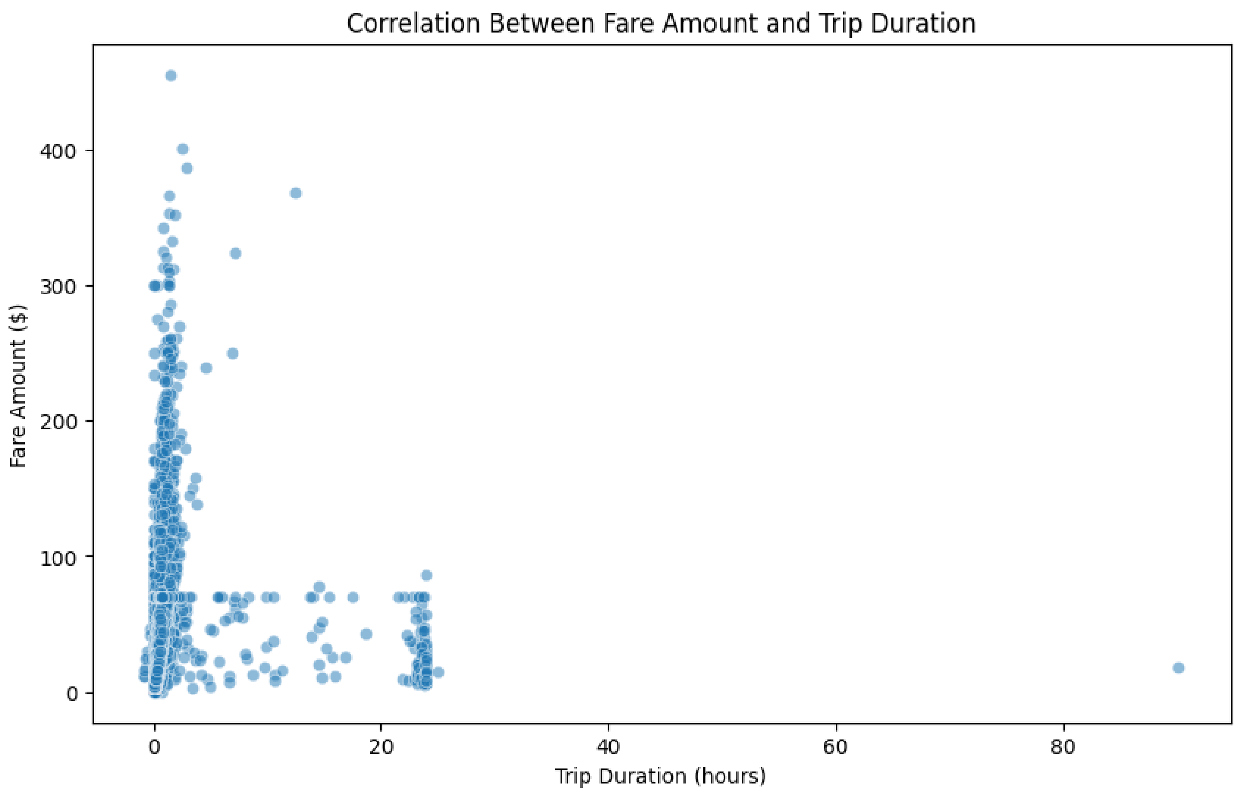
```
# Show how trip fare is affected by distance

sns.scatterplot(x=df_cleaned['trip_distance'], y=df_cleaned['fare_amount'])
plt.title('Distance vs Fare Amount')
plt.show()
```



```
# Show relationship between fare and number of passengers


# 2. Correlation between fare_amount and passenger_count
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df_cleaned['passenger_count'], y=df_cleaned['fare_amount'],
alpha=0.5)
plt.title('Correlation Between Fare Amount and Passenger Count')
plt.xlabel('Passenger Count')
plt.ylabel('Fare Amount ($)')
plt.show()

correlation = df_cleaned['fare_amount'].corr(df_cleaned['passenger_count'])
print(f"Correlation between fare_amount and passenger_count: {correlation:.2f}")
```

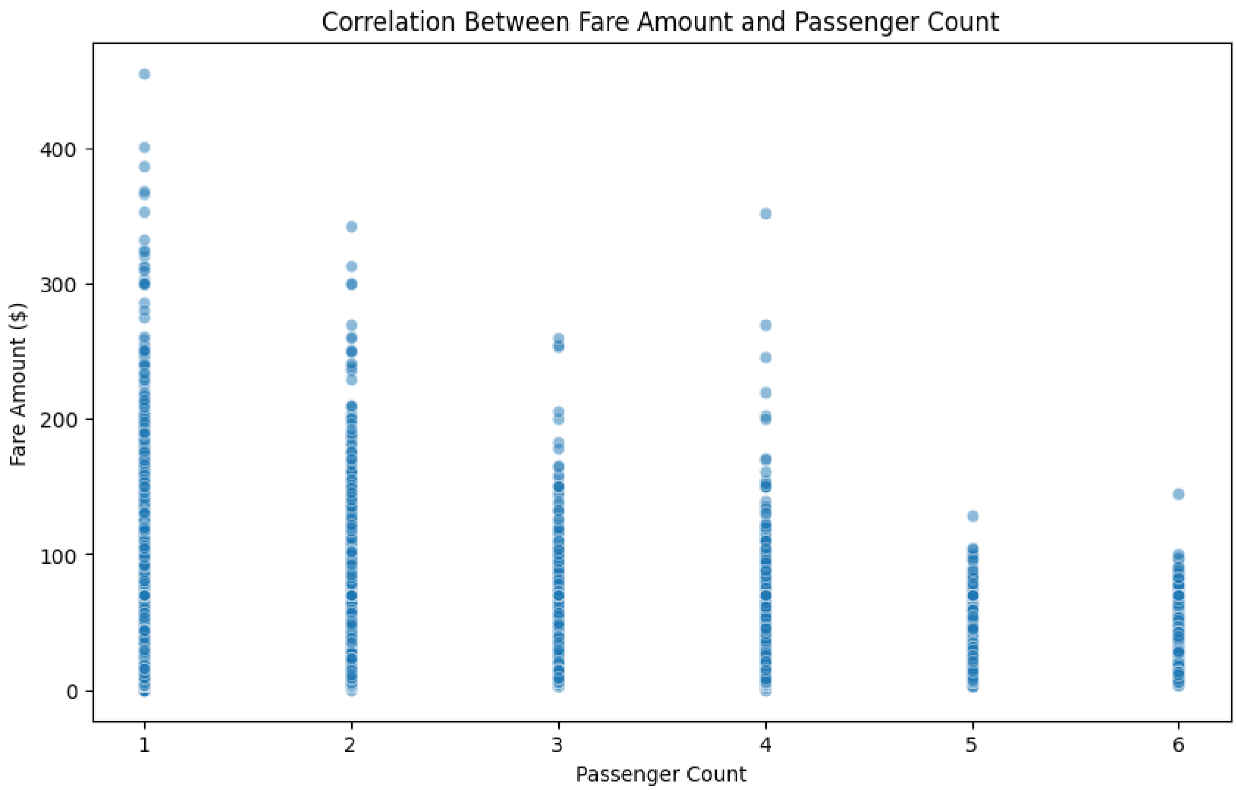
3.1.5. Analyse the relationship between fare/tips and trips/passengers

```
# Show relationship between fare and trip duration


# Convert datetime columns to datetime format
df_cleaned['tpep_pickup_datetime'] =
pd.to_datetime(df_cleaned['tpep_pickup_datetime'])
df_cleaned['tpep_dropoff_datetime'] =
pd.to_datetime(df_cleaned['tpep_dropoff_datetime'])

# Calculate trip duration in hours
df_cleaned['trip_duration'] = (df_cleaned['tpep_dropoff_datetime'] -
df_cleaned['tpep_pickup_datetime']).dt.total_seconds() / 3600

# 1. Correlation between fare_amount and trip_duration
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df_cleaned['trip_duration'], y=df_cleaned['fare_amount'],
alpha=0.5)
plt.title('Correlation Between Fare Amount and Trip Duration')
plt.xlabel('Trip Duration (hours)')
plt.ylabel('Fare Amount ($)')
plt.show()

correlation = df_cleaned['fare_amount'].corr(df_cleaned['trip_duration'])
print(f"Correlation between fare_amount and trip_duration:
{correlation:.2f}")
```



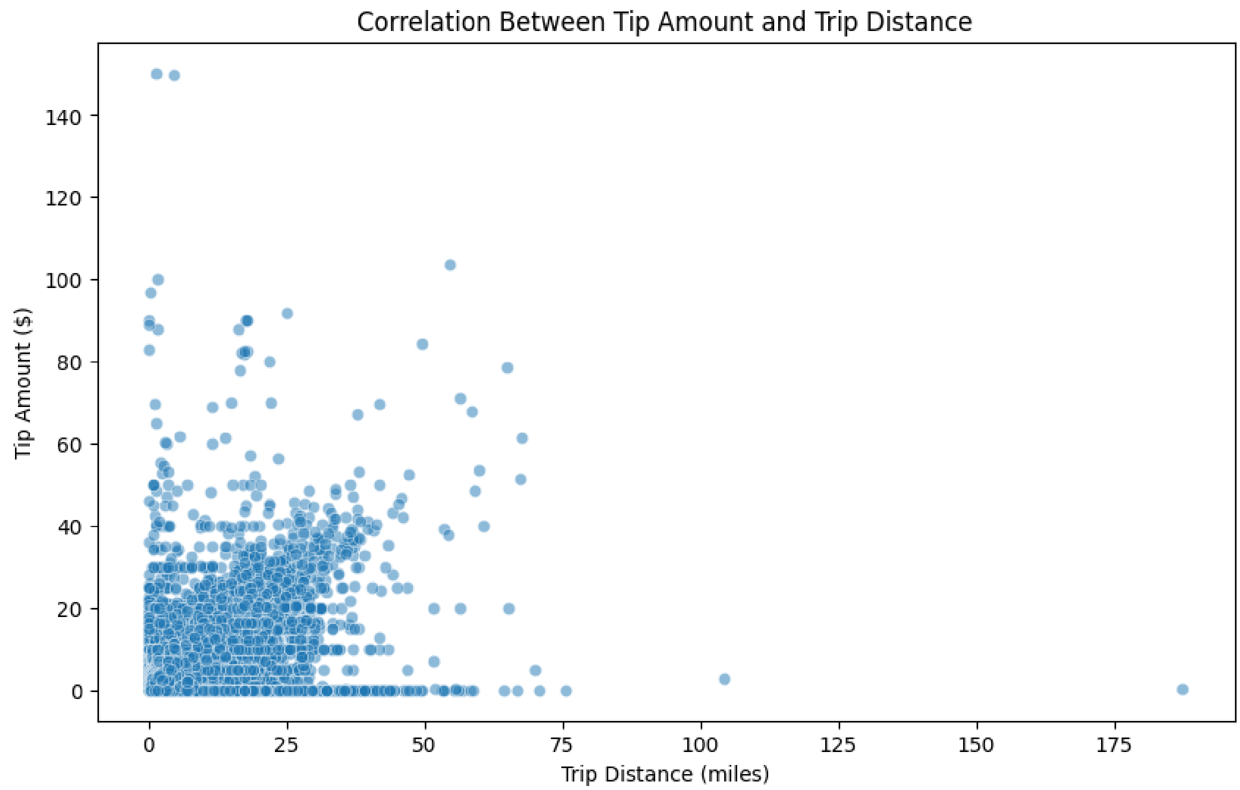

```
Correlation between fare_amount and trip_duration: 0.28
```

```
# Show relationship between fare and number of passengers


# 2. Correlation between fare_amount and passenger_count
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df_cleaned['passenger_count'], y=df_cleaned['fare_amount'],
alpha=0.5)
plt.title('Correlation Between Fare Amount and Passenger Count')
plt.xlabel('Passenger Count')
plt.ylabel('Fare Amount ($)')
plt.show()

correlation = df_cleaned['fare_amount'].corr(df_cleaned['passenger_count'])
print(f"Correlation between fare_amount and passenger_count: {correlation:.2f}")
```




```
Correlation between fare_amount and passenger_count: 0.04
```

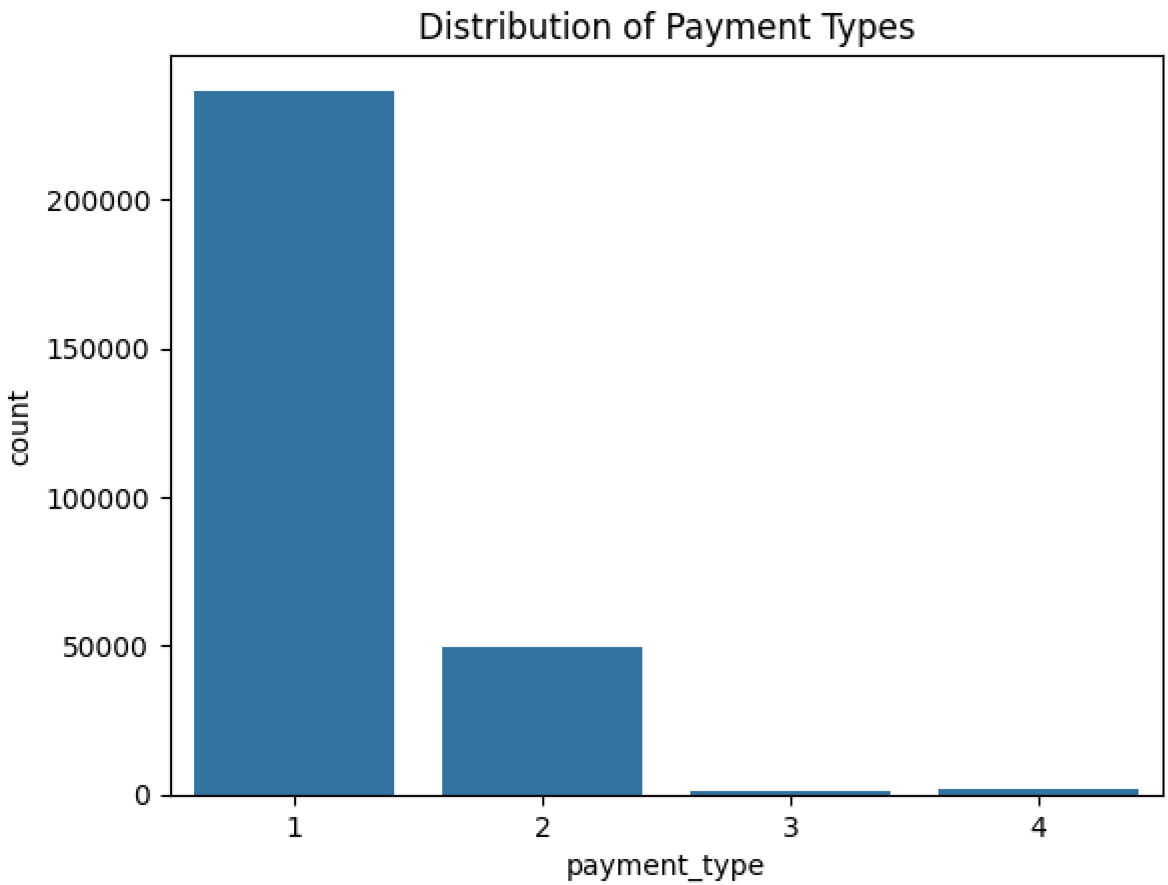
```
# Show relationship between tip and trip distance

# 3. Correlation between tip_amount and trip_distance
plt.figure(figsize=(10, 6))
sns.scatterplot(x=df_cleaned['trip_distance'], y=df_cleaned['tip_amount'],
alpha=0.5)
plt.title('Correlation Between Tip Amount and Trip Distance')
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Tip Amount ($)')
plt.show()

correlation = df_cleaned['tip_amount'].corr(df_cleaned['trip_distance'])
print(f"Correlation between tip_amount and trip_distance: {correlation:.2f}")
```




```
Correlation between tip_amount and trip_distance: 0.59
```

3.1.6. Analyse the distribution of different payment types

```python
# Analyse the distribution of different payment types (payment_type).


sns.countplot(x=df_cleaned['payment_type'])
plt.title('Distribution of Payment Types')
plt.show()
```



3.1.7. Load the taxi zones shapefile and display it

```
import geopandas as gpd
os.chdir(r"D:\Upgrad\Upgrad_DS\DSC75\EDA\Datasets and Dictionary-NYC\Datasets and Dictionary\taxi_zones")

# Read the shapefile using geopandas
zones = gpd.read_file('taxi_zones.shp')# read the .shp file using gpd
zones.head()

print(zones.info())
zones.plot()
```

Merge the zone data with trips data

```
# Merge zones and trip records using locationID and PULocationID

df_cleaned = df_cleaned.merge(zones, left_on='PULocationID',
right_on='LocationID', how='left')
```

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

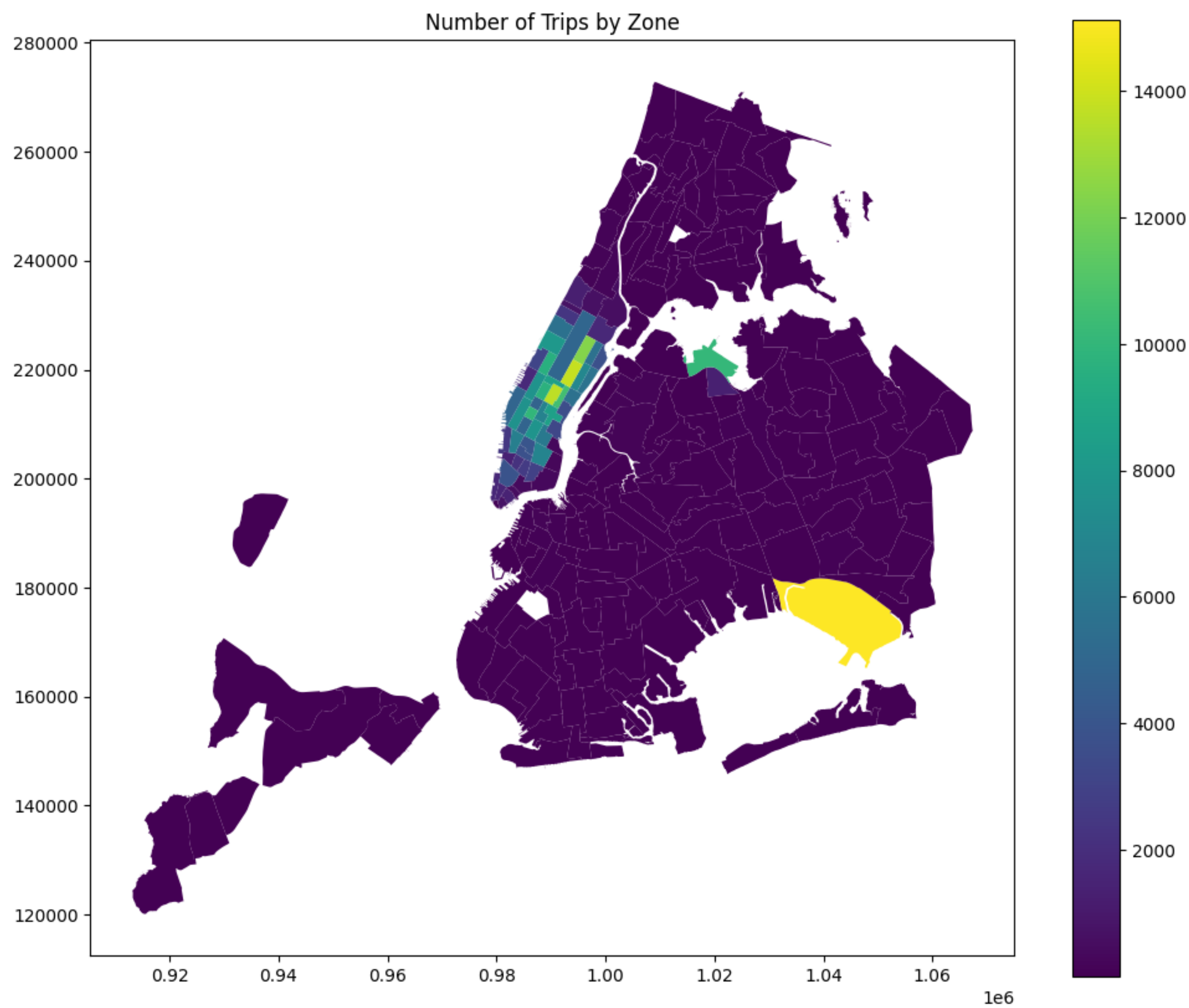
```
trips_by_zone = df_cleaned['PULocationID'].value_counts()
print(trips_by_zone)
```

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

```
# Merge trip counts back to the zones GeoDataFrame

zones = zones.merge(trips_by_zone.rename('trip_count'), left_on='LocationID',
right_index=True, how='left')
```

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

```
# Define figure and axis
fig, ax = plt.subplots(1, 1, figsize = (12, 10))

# Plot the map and display it

zones.plot(column='trip_count', ax = ax, legend=True)
plt.title('Number of Trips by Zone')
plt.show()
```



Conclude with results

```
# Sort the zones DataFrame by the number of trips (descending order)
zones_sorted = zones.sort_values(by='trip_count', ascending=False)

# Display the sorted DataFrame
print(zones_sorted[['LocationID', 'zone', 'trip_count']].head(10))  # Display
top 10 zones
```

```
LocationID                         zone  trip_count
131         132              JFK Airport     15135.0
236         237    Upper East Side South     13788.0
160         161           Midtown Center     13625.0
235         236    Upper East Side North     12329.0
161         162             Midtown East     10574.0
137         138        LaGuardia Airport     10093.0
```

```
185          186  Penn Station/Madison Sq West       9990.0
229          230     Times Sq/Theatre District       9695.0
141          142           Lincoln Square East       9575.0

169          170                   Murray Hill       8566.0
```

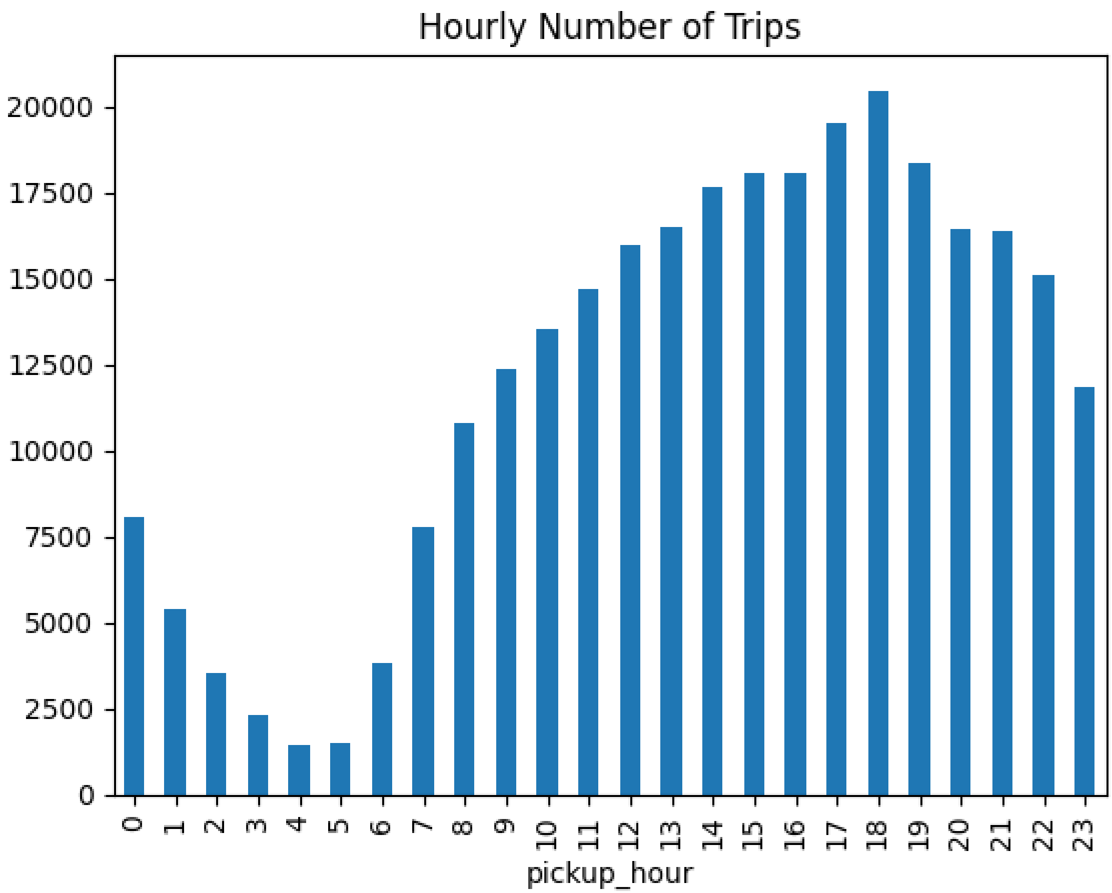
3.2. Detailed EDA: Insights and Strategies

Identify slow routes by comparing average speeds on different routes

```
# Remove invalid trips (where trip_duration <= 0)
df_cleaned = df_cleaned[(df_cleaned['trip_duration']>0)]


df_cleaned['trip_duration'] = (df_cleaned['tpep_dropoff_datetime'] -
df_cleaned['tpep_pickup_datetime']).dt.total_seconds() / 3600
df_cleaned['speed'] = df_cleaned['trip_distance'] /
df_cleaned['trip_duration']
slow_routes = df_cleaned.groupby(['PULocationID',
'DOLocationID'])['speed'].mean().sort_values()
print(slow_routes.head(10))
```

Calculate the hourly number of trips and identify the busy hours

```
# Visualise the number of trips per hour and find the busiest hour

hourly_trips = df_cleaned['pickup_hour'].value_counts().sort_index()
hourly_trips.plot(kind='bar')
plt.title('Hourly Number of Trips')
plt.show()
```

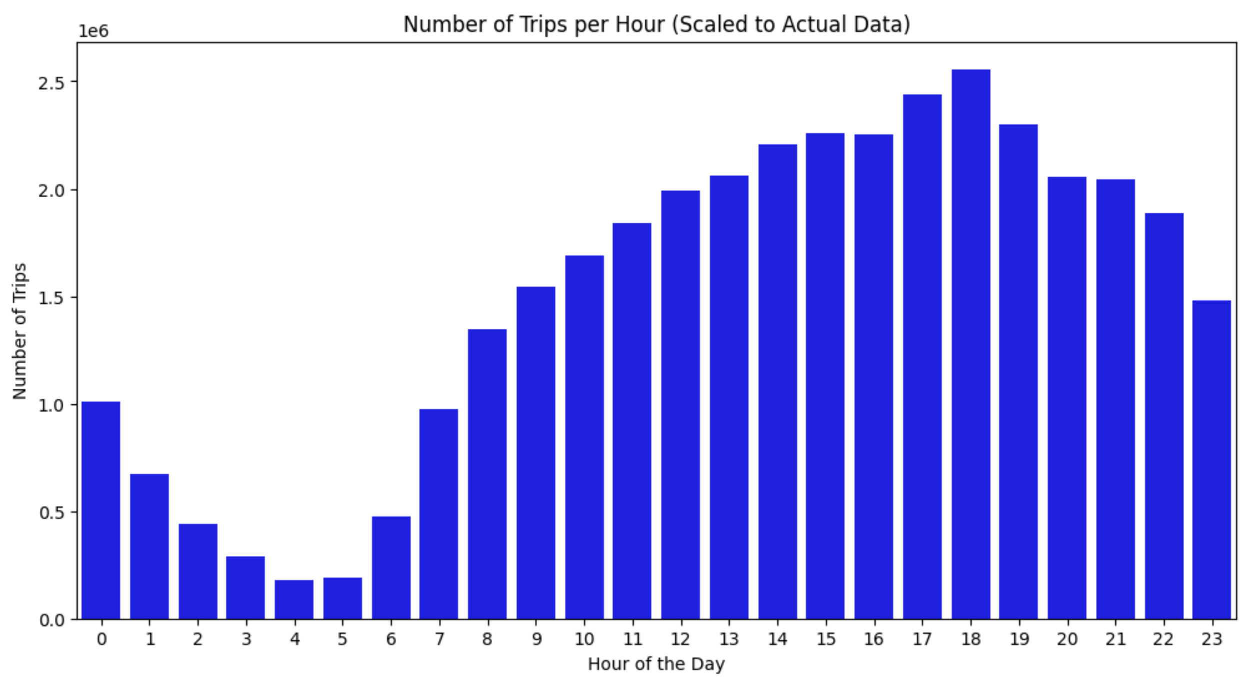


We identified in previous plot that the busiest hrs are between 10 AM to 10 PM

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

```
# Convert tpep_pickup_datetime to datetime format
df_cleaned['tpep_pickup_datetime'] =
pd.to_datetime(df_cleaned['tpep_pickup_datetime'])

# Step 1: Extract the hour from the pickup datetime
df_cleaned['pickup_hour'] = df_cleaned['tpep_pickup_datetime'].dt.hour

# Step 2: Count the number of trips per hour
trips_per_hour = df_cleaned['pickup_hour'].value_counts().sort_index()

# Step 3: Scale up the number of trips by the sampling ratio (0.001)
sampling_ratio = 0.008
```

```
trips_per_hour_scaled = trips_per_hour / sampling_ratio

# Step 4: Visualize the number of trips per hour
plt.figure(figsize=(12, 6))
sns.barplot(x=trips_per_hour_scaled.index, y=trips_per_hour_scaled.values,
color='blue')
plt.title('Number of Trips per Hour (Scaled to Actual Data)')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Trips')
plt.xticks(range(0, 24))  # Ensure all hours are displayed
plt.show()

# Step 5: Find the busiest hour
busiest_hour = trips_per_hour_scaled.idxmax()
busiest_hour_trips = trips_per_hour_scaled.max()
print(f"The busiest hour is {busiest_hour}:00 with approximately
{busiest_hour_trips:.0f} trips.")
```

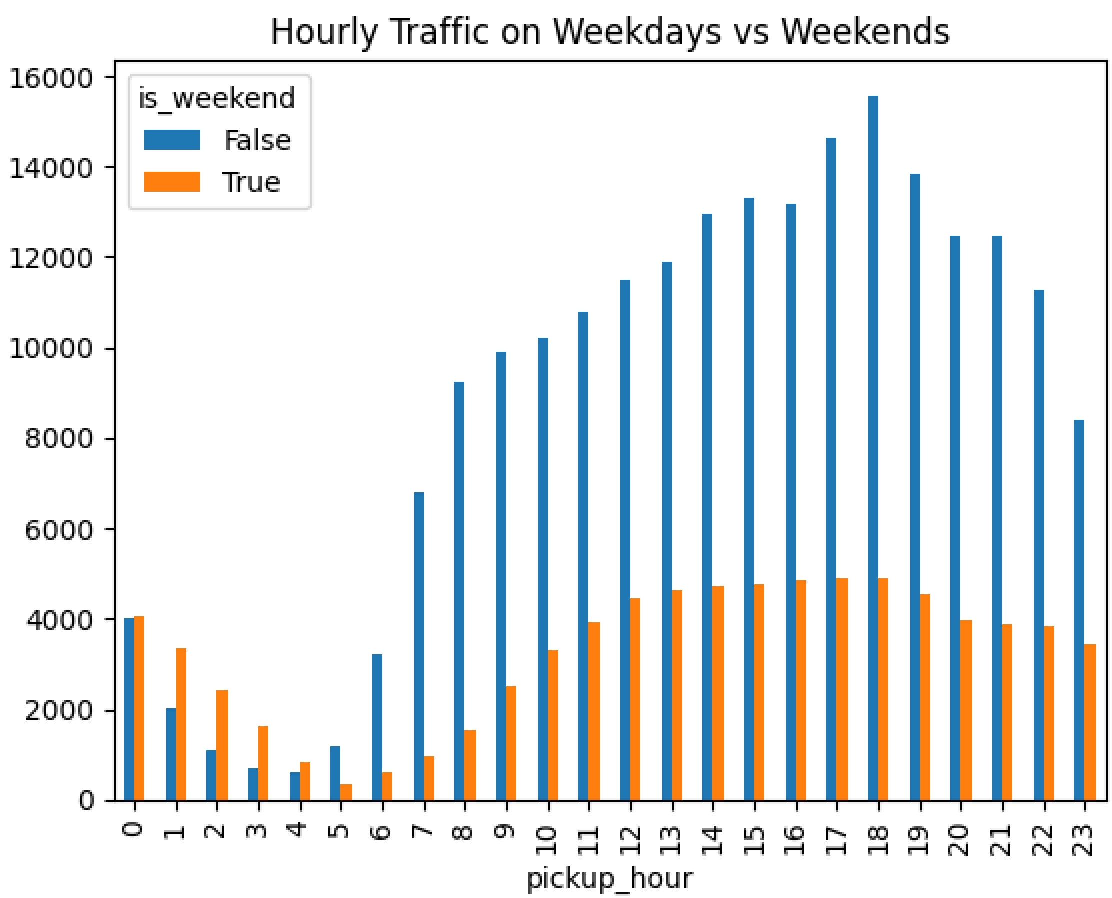



```
The busiest hour is 18:00 with approximately 2556750 trips.
```

3.2.2. Compare hourly traffic on weekdays and weekends

```
# Compare traffic trends for the week days and weekends
```

```
df_cleaned['is_weekend'] = df_cleaned['pickup_day'].isin(['Saturday', 'Sunday'])
hourly_traffic = df_cleaned.groupby(['pickup_hour',
'is_weekend']).size().unstack()
hourly_traffic.plot(kind='bar')
plt.title('Hourly Traffic on Weekdays vs Weekends')
plt.show()
```



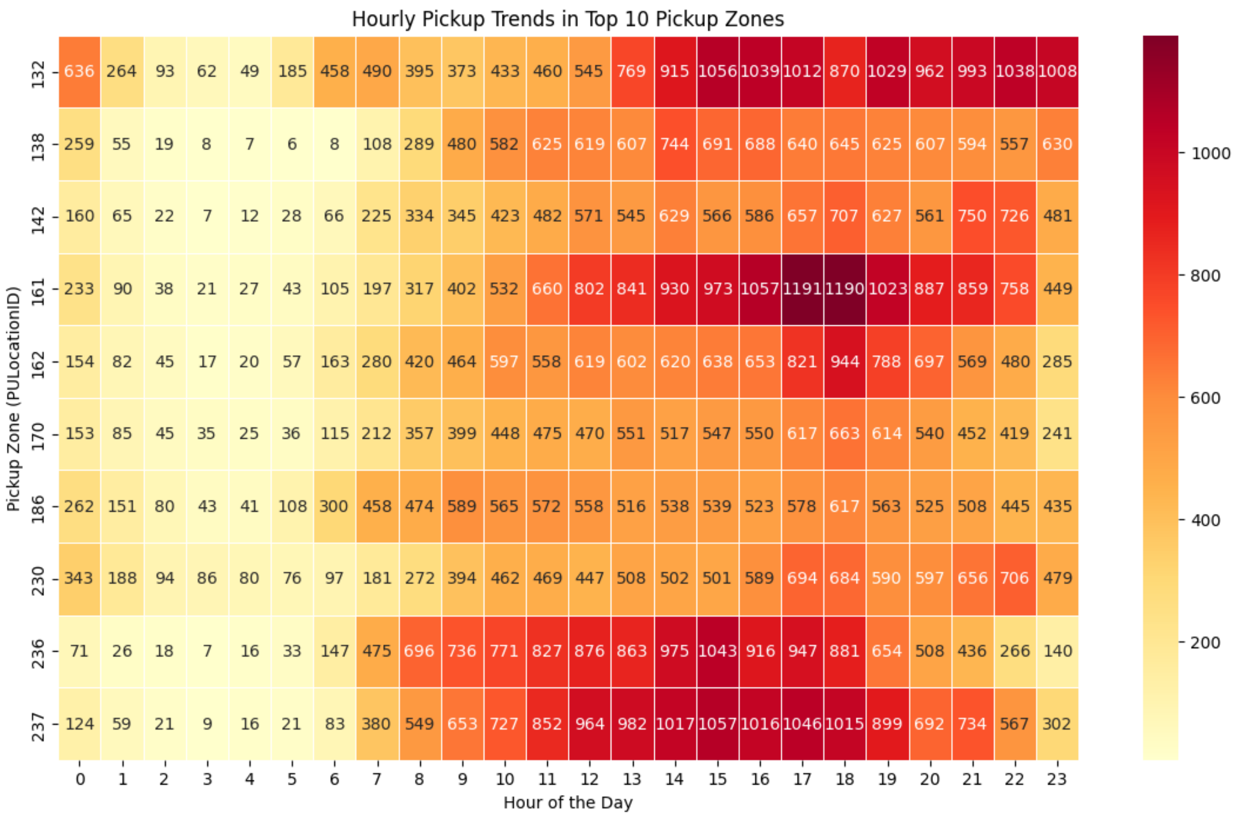
3.2.3. Identify the top 10 zones with high hourly pickups and drops

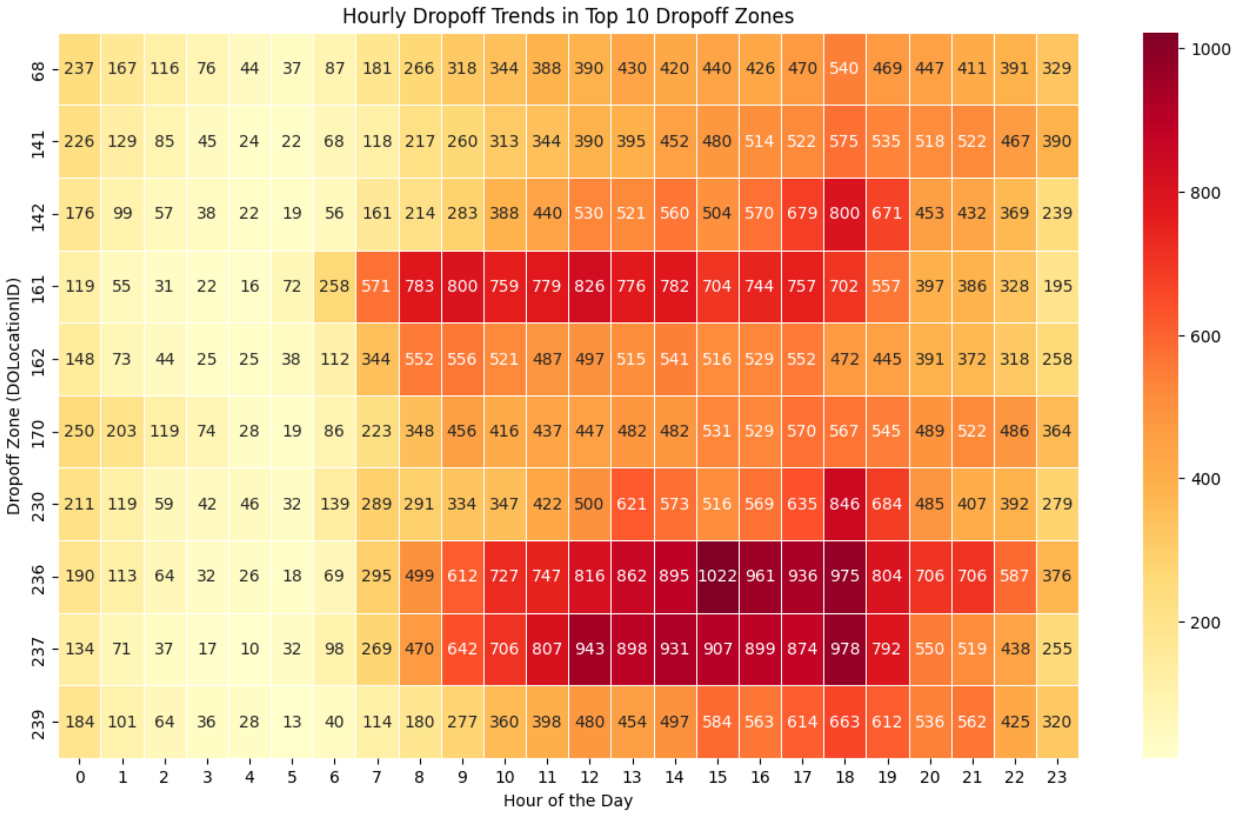
```
# Find top 10 pickup and dropoff zones

top_pickup_zones = df_cleaned['PULocationID'].value_counts().head(10).index
top_dropoff_zones = df_cleaned['DOLocationID'].value_counts().head(10).index
print(top_pickup_zones, top_dropoff_zones)
```

```
Index([132, 237, 161, 236, 162, 138, 186, 230, 142, 170], dtype='int64',
name='PULocationID') Index([236, 237, 161, 230, 170, 162, 142, 239, 141, 68],
dtype='int64', name='DOLocationID')
```

```
# Step 2: Extract hour from pickup and dropoff datetime
df_cleaned['pickup_hour'] = df_cleaned['tpep_pickup_datetime'].dt.hour
df_cleaned['dropoff_hour'] = df_cleaned['tpep_dropoff_datetime'].dt.hour

# Step 3: Create a pivot table for hourly pickups in top 10 pickup zones
pickup_pivot =
df_cleaned[df_cleaned['PULocationID'].isin(top_pickup_zones)].pivot_table(
    index='PULocationID', columns='pickup_hour', values='tpep_pickup_datetime',
aggfunc='count', fill_value=0
)

# Step 4: Create a pivot table for hourly dropoffs in top 10 dropoff zones
dropoff_pivot =
df_cleaned[df_cleaned['DOLocationID'].isin(top_dropoff_zones)].pivot_table(
    index='DOLocationID', columns='dropoff_hour', values='tpep_dropoff_datetime',
aggfunc='count', fill_value=0
)

# Step 5: Plot heatmaps for pickup and dropoff trends
plt.figure(figsize=(14, 8))
sns.heatmap(pickup_pivot, cmap='YlOrRd', annot=True, fmt='d', linewidths=0.5)
plt.title('Hourly Pickup Trends in Top 10 Pickup Zones')
plt.xlabel('Hour of the Day')
plt.ylabel('Pickup Zone (PULocationID)')
plt.show()

plt.figure(figsize=(14, 8))
sns.heatmap(dropoff_pivot, cmap='YlOrRd', annot=True, fmt='d', linewidths=0.5)
plt.title('Hourly Dropoff Trends in Top 10 Dropoff Zones')
plt.xlabel('Hour of the Day')
plt.ylabel('Dropoff Zone (DOLocationID)')
plt.show()

# Step 6: Compare pickup and dropoff trends for the same zone (if any overlap)
common_zones = set(top_pickup_zones).intersection(set(top_dropoff_zones))
if common_zones:
    for zone in common_zones:
        plt.figure(figsize=(10, 6))
        pickup_data = df_cleaned[df_cleaned['PULocationID'] == zone]
        dropoff_data = df_cleaned[df_cleaned['DOLocationID'] == zone]
        hourly_pickups = pickup_data['pickup_hour'].value_counts().sort_index()
        hourly_dropoffs =
dropoff_data['dropoff_hour'].value_counts().sort_index()
```

```
        sns.lineplot(x=hourly_pickups.index, y=hourly_pickups.values, label='Pickups')
        sns.lineplot(x=hourly_dropoffs.index, y=hourly_dropoffs.values, label='Dropoffs')

        plt.title(f'Pickup and Dropoff Trends in Zone {zone}')
        plt.xlabel('Hour of the Day')
        plt.ylabel('Number of Trips')
        plt.xticks(range(0, 24))
        plt.legend()
        plt.show()
else:
    print("No common zones in the top 10 pickup and dropoff zones.")
```





Find the ratio of pickups and dropoffs in each zone

```
# Find the top 10 and bottom 10 pickup/dropoff ratios

pickup_dropoff_ratio = df_cleaned['PULocationID'].value_counts() /
df_cleaned['DOLocationID'].value_counts()
print(pickup_dropoff_ratio)
pickup_dropoff_ratio.sort_values(ascending=True).head(10)
pickup_dropoff_ratio.sort_values(ascending=False).head(10)
```

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

```
# During night hours (11pm to 5am) find the top 10 pickup and dropoff zones
# Note that the top zones should be of night hours and not the overall top
zones
night_trips = df_cleaned[(df_cleaned['pickup_hour'] >= 23) |
(df_cleaned['pickup_hour'] < 6)]
top_night_zones = night_trips['PULocationID'].value_counts().head(10)
print(top_night_zones)
```

3.2.5. Find the revenue share for nighttime and daytime hours

```
# Filter for night hours (11 PM to 5 AM)

night_revenue = night_trips['total_amount'].sum()
day_revenue = df_cleaned[~((df_cleaned['pickup_hour'] >= 22) |
(df_cleaned['pickup_hour'] < 6))]['total_amount'].sum()
print(f"Nighttime Revenue Share: {night_revenue / (night_revenue + day_revenue) *
100:.2f}%")
```

```
Nighttime Revenue Share: 17.36%
```

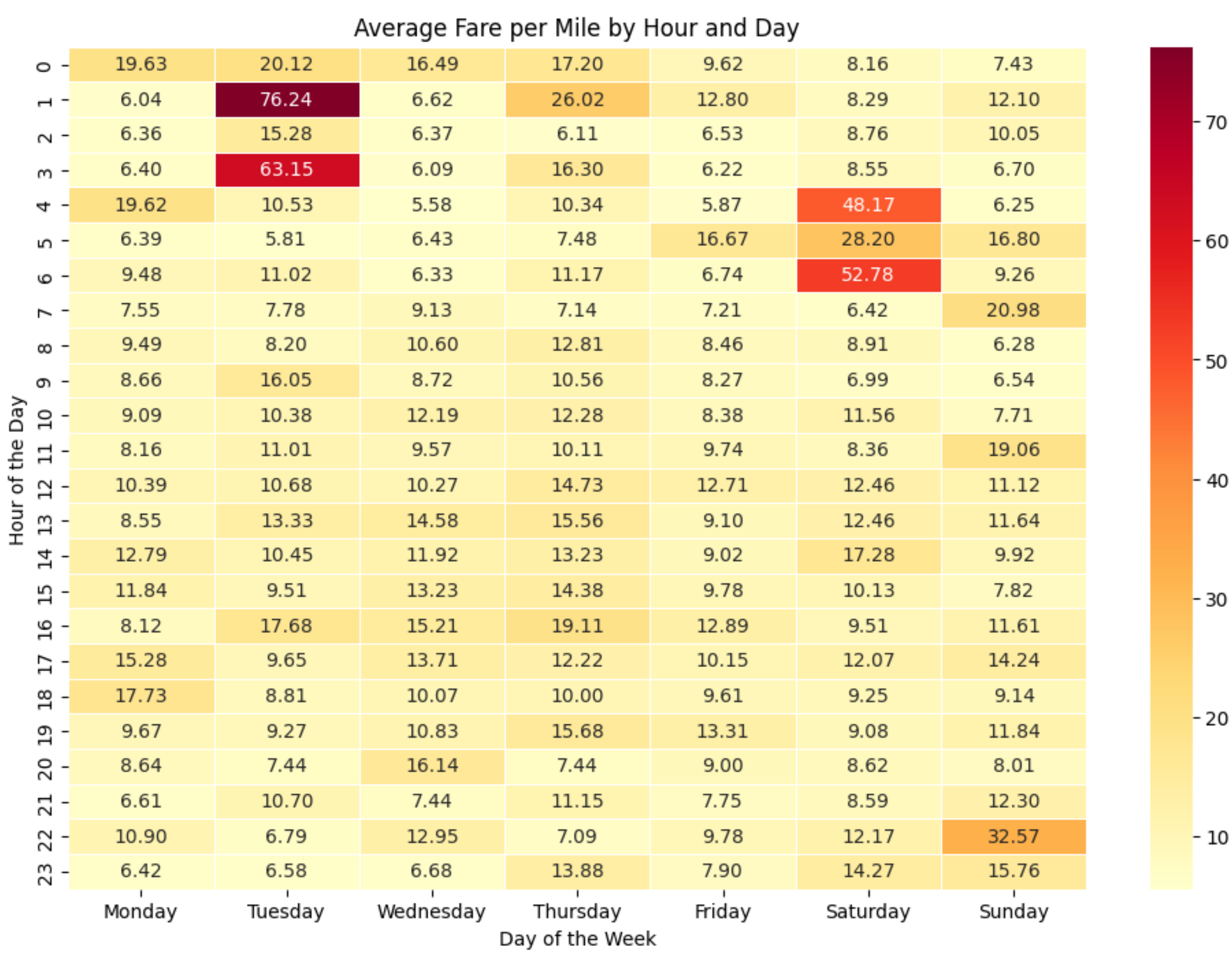
3.2.6. For the different passenger counts, find the average fare per mile per passenger

```
df_cleaned['fare_per_mile_per_passenger'] = df_cleaned['fare_amount'] /
(df_cleaned['trip_distance'] * df_cleaned['passenger_count'])
fare_by_passengers =
df_cleaned.groupby('passenger_count')['fare_per_mile_per_passenger'].mean()
print(fare_by_passengers)
```

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

```
# Compare the average fare per mile for different days and for different times of
the day

# Step 1: Calculate the average fare per mile
df_cleaned['fare_per_mile'] = df_cleaned['fare_amount'] /
df_cleaned['trip_distance']

pivot_table = df_cleaned.pivot_table(
    index='pickup_hour',  # Rows: Hours of the day
    columns='pickup_day',  # Columns: Days of the week
    values='fare_per_mile',  # Values: Average fare per mile
    aggfunc='mean',  # Calculate the mean
    fill_value=0  # Fill missing values with 0
)
```

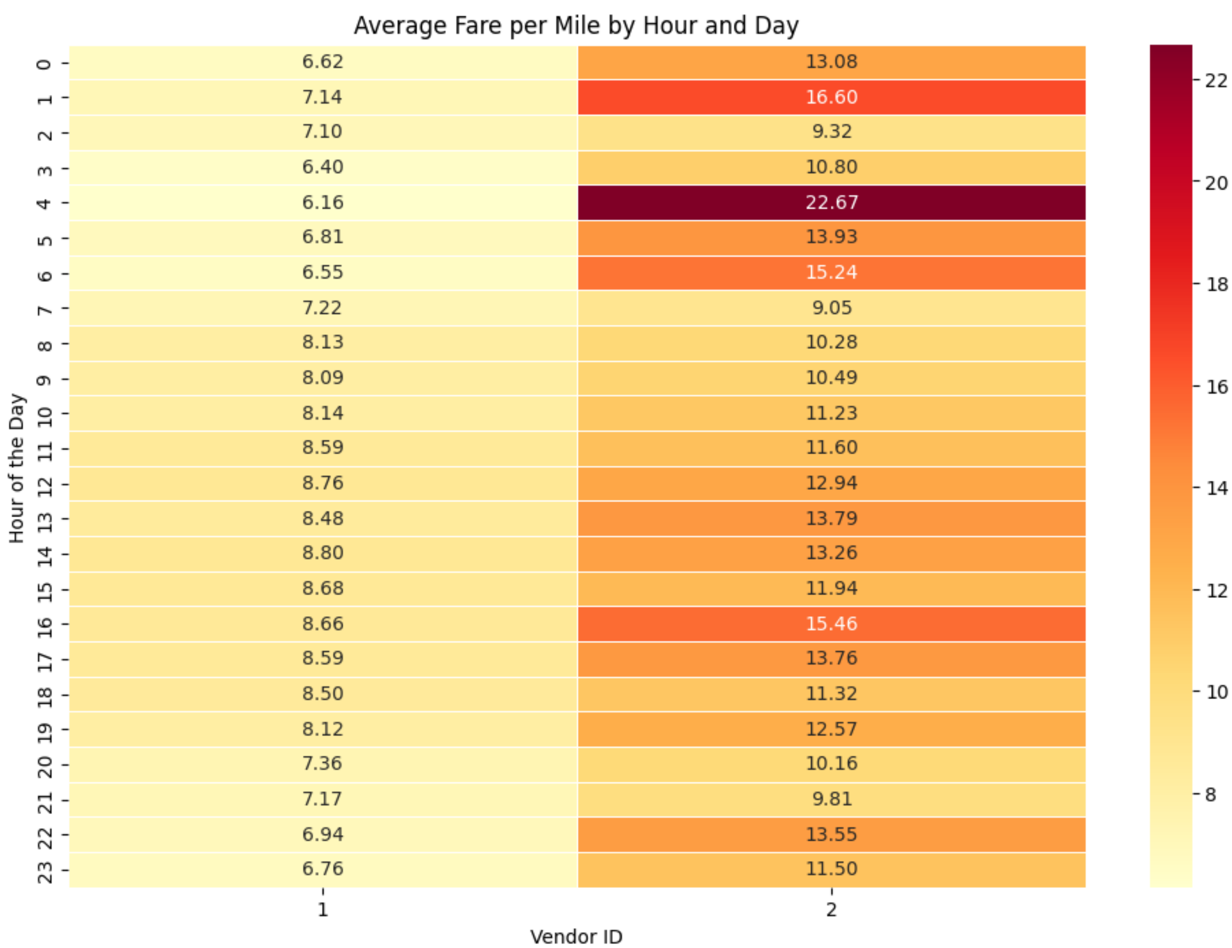
```
# Reorder columns to match days of the week
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
'Sunday']
pivot_table = pivot_table[day_order]

# Step 3: Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(pivot_table, cmap='YlOrRd', annot=True, fmt='.2f', linewidths=0.5)
plt.title('Average Fare per Mile by Hour and Day')
plt.xlabel('Day of the Week')
plt.ylabel('Hour of the Day')
plt.show()
```



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

```
# Compare fare per mile for different vendors

pivot_table = df_cleaned.pivot_table(
```

```
    index='pickup_hour',  # Rows: Hours of the day
    columns='VendorID',  # Columns: Vendor
    values='fare_per_mile',  # Values: Average fare per mile
    aggfunc='mean',  # Calculate the mean
    fill_value=0  # Fill missing values with 0
)



# Step 3: Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(pivot_table, cmap='YlOrRd', annot=True, fmt='.2f', linewidths=0.5)
plt.title('Average Fare per Mile by Hour and Day')
plt.xlabel('Vendor ID')
plt.ylabel('Hour of the Day')
plt.show()
```

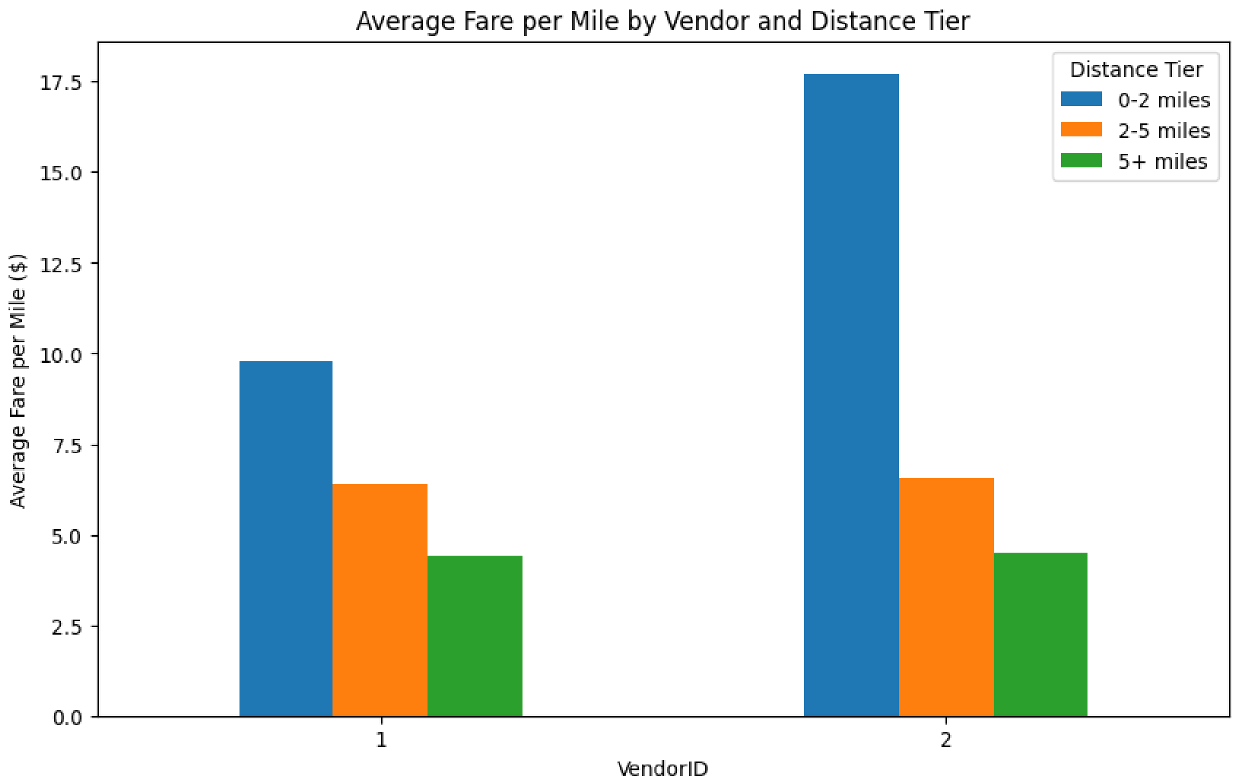


3.2.9. Compare the fare rates of different vendors in a distance-tiered fashion

```
# Defining distance tiers
df_cleaned['distance_tier'] = pd.cut(df_cleaned['trip_distance'], bins=[0, 2, 5,
float('inf')], labels=['0-2 miles', '2-5 miles', '5+ miles'])

# Group by VendorID and distance_tier, and calculate the average fare per mile
fare_by_vendor_distance = df_cleaned.groupby(['VendorID',
'distance_tier'])['fare_per_mile'].mean().unstack()

# Print the results
print(fare_by_vendor_distance)
```

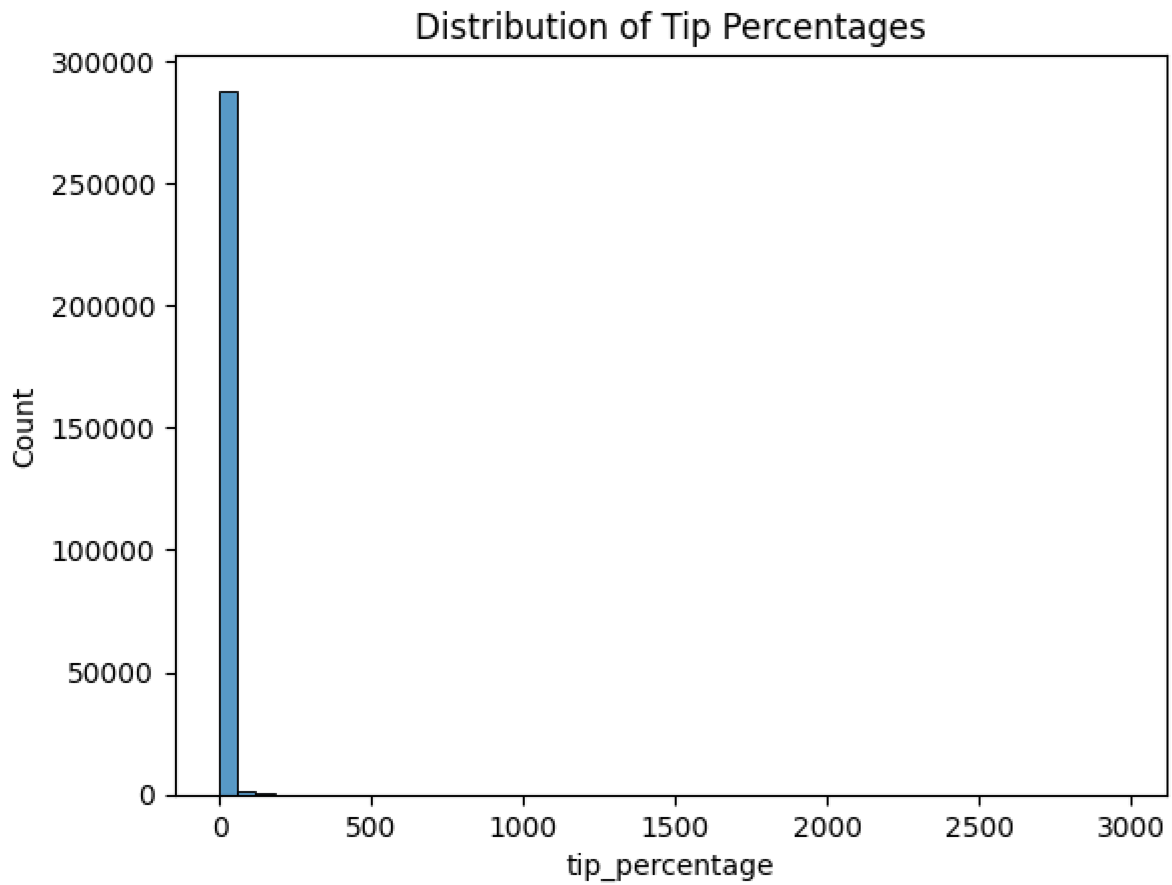
```
distance_tier  0-2 miles  2-5 miles  5+ miles
VendorID
1               9.784701   6.379998  4.415945

2              17.723540   6.543767  4.502578
```

```
# Plot the results
fare_by_vendor_distance.plot(kind='bar', figsize=(10, 6))
plt.title('Average Fare per Mile by Vendor and Distance Tier')
plt.xlabel('VendorID')
plt.ylabel('Average Fare per Mile ($)')
plt.xticks(rotation=0)
plt.legend(title='Distance Tier')
plt.show()
```



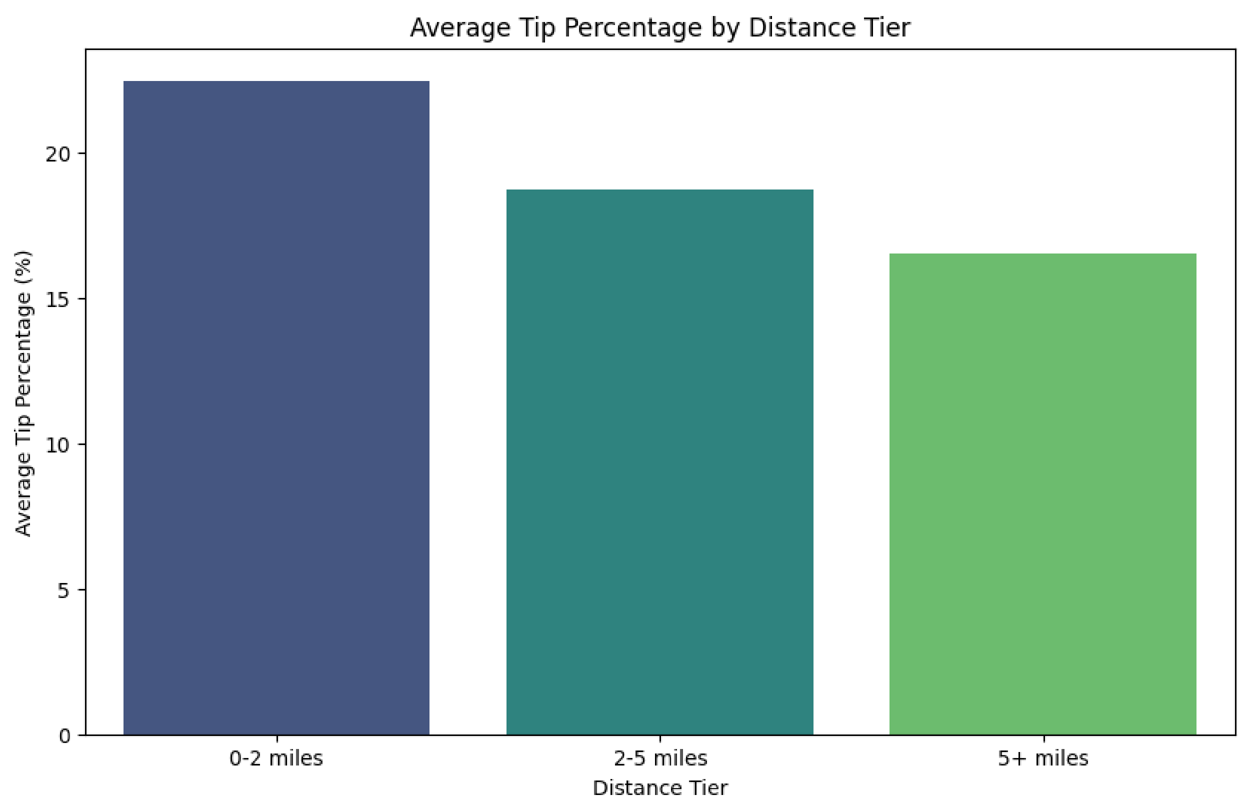
3.2.10. Analyse the tip percentages

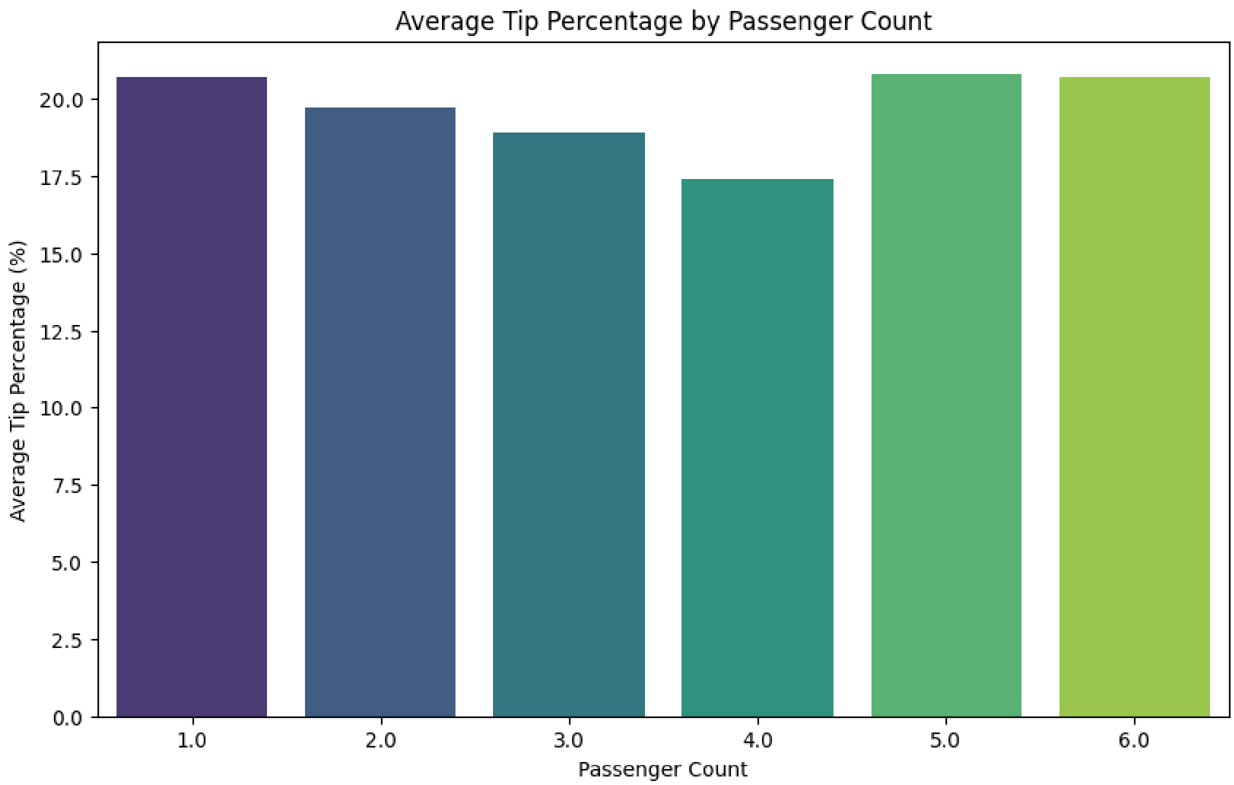
```
#  Analyze tip percentages based on distances, passenger counts and pickup times

df_cleaned['tip_percentage'] = df_cleaned['tip_amount'] /
df_cleaned['fare_amount'] * 100
sns.histplot(df_cleaned['tip_percentage'], bins=50)
plt.title('Distribution of Tip Percentages')
plt.show()
```

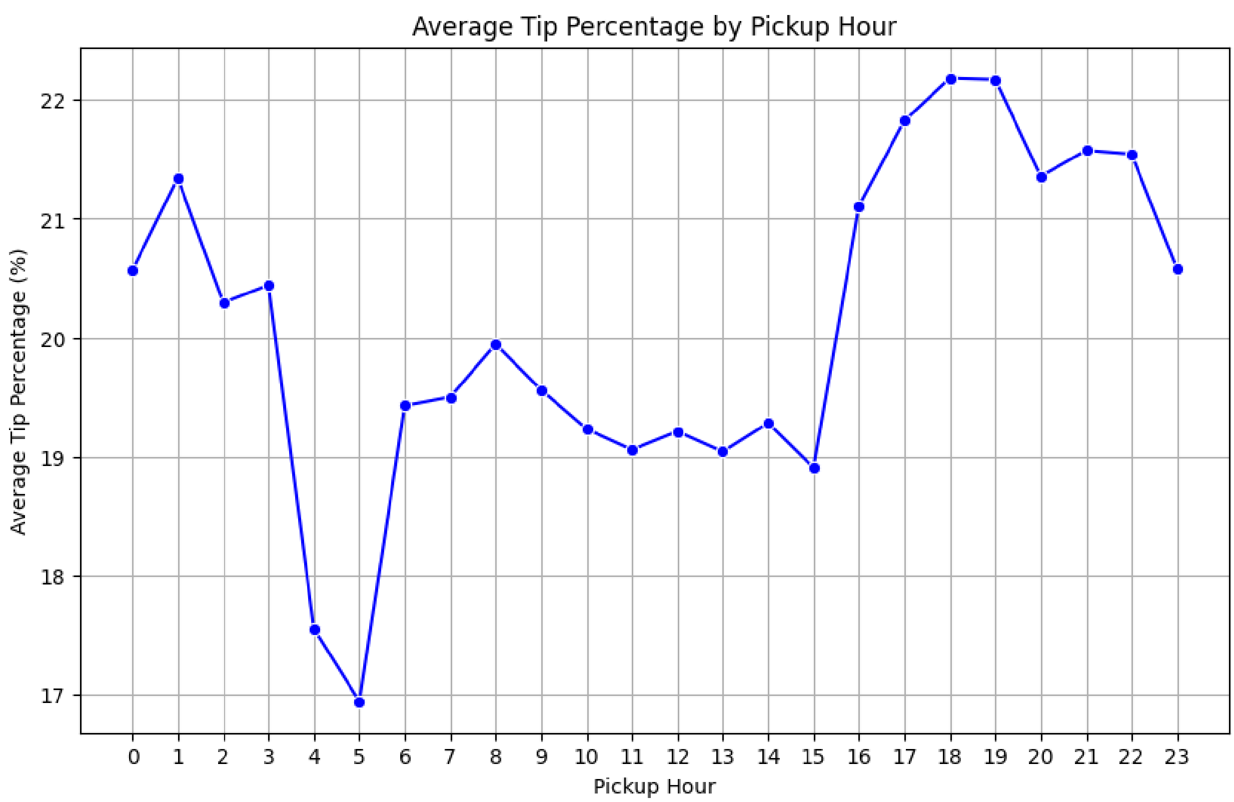



```
# Compare trips with tip percentage < 10% to trips with tip percentage > 25%

tip_by_distance = df_cleaned.groupby('distance_tier')['tip_percentage'].mean()

# Analyze tip percentages by passenger count
# Group by passenger count and calculate average tip percentage
tip_by_passenger = df_cleaned.groupby('passenger_count')['tip_percentage'].mean()

# Analyze tip percentages by pickup time

# Group by pickup hour and calculate average tip percentage
tip_by_hour = df_cleaned.groupby('pickup_hour')['tip_percentage'].mean()

# Visualize the results
# Plot tip percentages by distance
plt.figure(figsize=(10, 6))
sns.barplot(x=tip_by_distance.index, y=tip_by_distance.values, palette='viridis')
plt.title('Average Tip Percentage by Distance Tier')
plt.xlabel('Distance Tier')
plt.ylabel('Average Tip Percentage (%)')
plt.show()
```

```
# Plot tip percentages by passenger count
plt.figure(figsize=(10, 6))
sns.barplot(x=tip_by_passenger.index, y=tip_by_passenger.values,
palette='viridis')
plt.title('Average Tip Percentage by Passenger Count')
plt.xlabel('Passenger Count')
plt.ylabel('Average Tip Percentage (%)')
plt.show()

# Plot tip percentages by pickup hour
plt.figure(figsize=(10, 6))
sns.lineplot(x=tip_by_hour.index, y=tip_by_hour.values, marker='o', color='blue')
plt.title('Average Tip Percentage by Pickup Hour')
plt.xlabel('Pickup Hour')
plt.ylabel('Average Tip Percentage (%)')
plt.xticks(range(0, 24))
plt.grid(True)
plt.show()
```

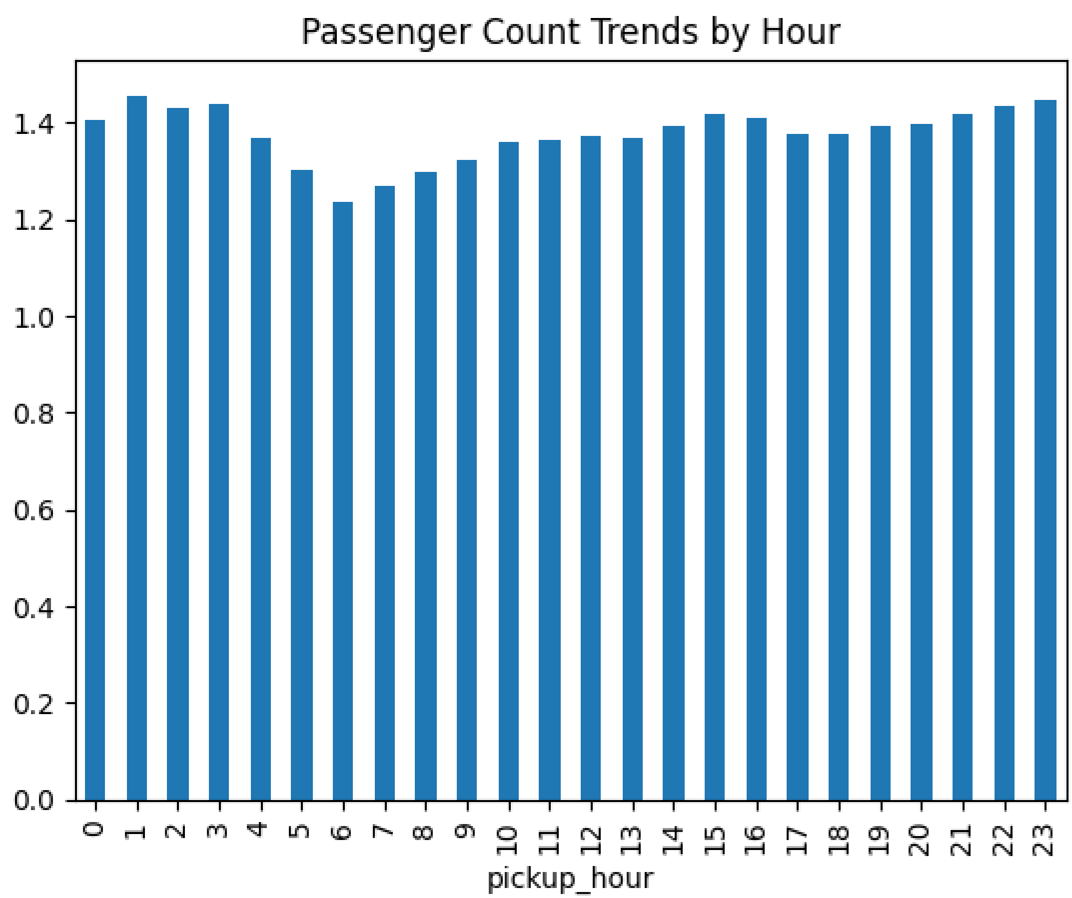


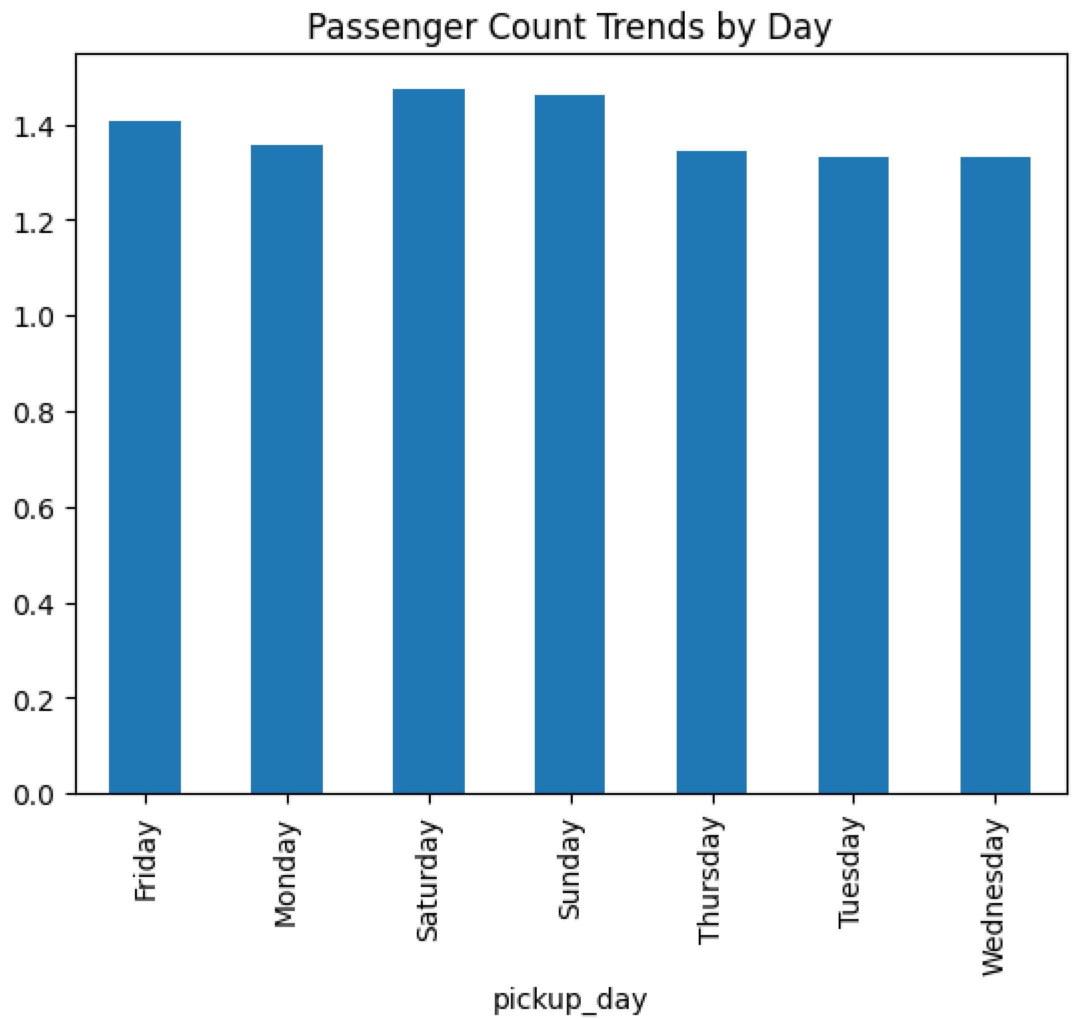

Average Tip Percentage by Passenger Count
Average Tip Percentage (%)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
1.0
2.0
3.0
4.0
5.0
6.0
Passenger Count


Average Tip Percentage by Pickup Hour
Average Tip Percentage (%)
17
18
19
20
21
22
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Pickup Hour

3.2.11. Analyse the trends in passenger count

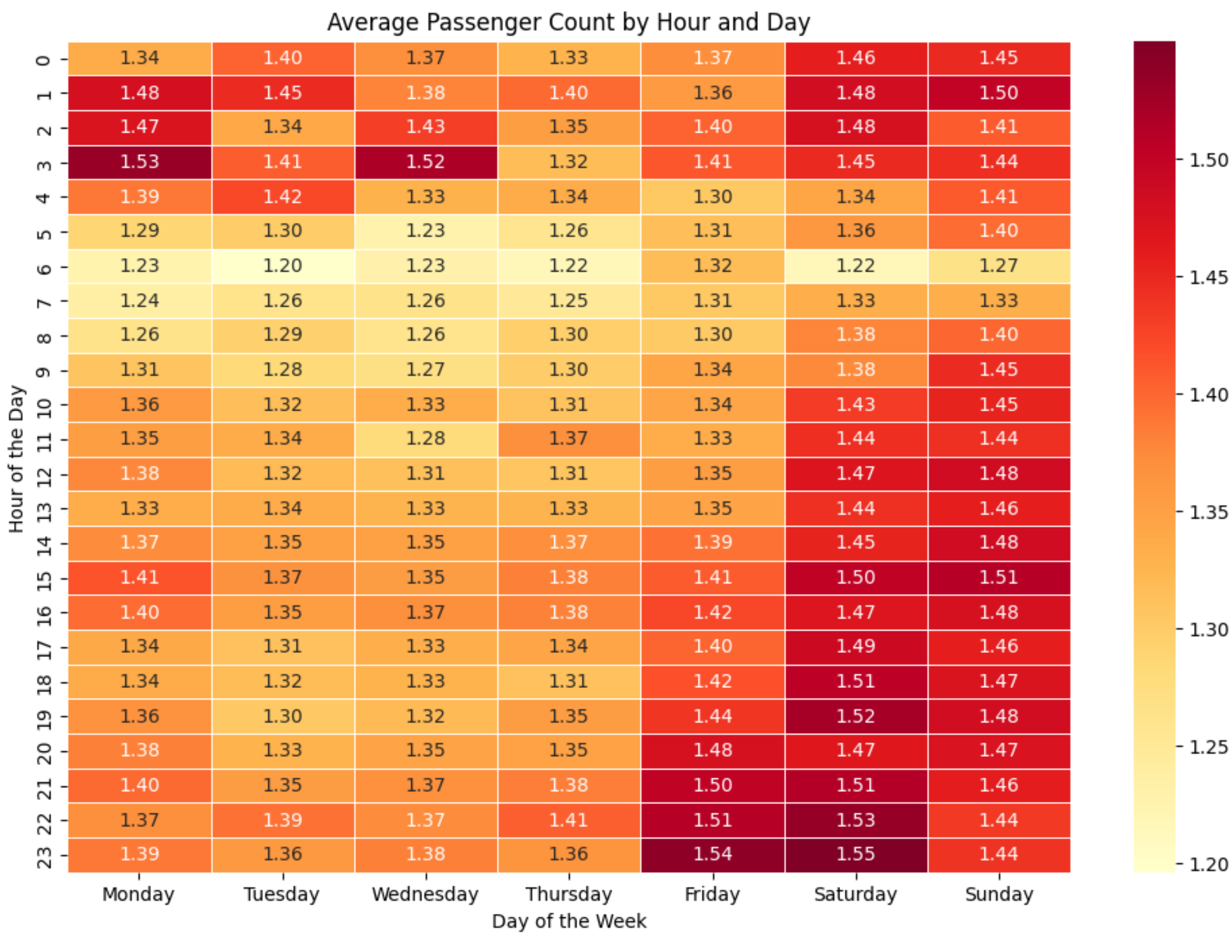
```
# See how passenger count varies across hours and days


passenger_trends = df_cleaned.groupby('pickup_hour')['passenger_count'].mean()
passenger_trends.plot(kind='bar')
plt.title('Passenger Count Trends by Hour')
plt.show()


passenger_trends2 = df_cleaned.groupby('pickup_day')['passenger_count'].mean()
passenger_trends2.plot(kind='bar')
plt.title('Passenger Count Trends by Day')
plt.show()
```






```
pivot_table = df_cleaned.pivot_table(
    index='pickup_hour',  # Rows: Hours of the day
    columns='pickup_day',  # Columns: Days of the week
    values='passenger_count',  # Values: Average passenger count
    aggfunc='mean',  # Calculate the mean
    fill_value=0  # Fill missing values with 0
)

# Reorder columns to match days of the week
pivot_table = pivot_table[day_order]

# Step 3: Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(pivot_table, cmap='YlOrRd', annot=True, fmt='.2f', linewidths=0.5)
plt.title('Average Passenger Count by Hour and Day')
plt.xlabel('Day of the Week')
plt.ylabel('Hour of the Day')
```

```python
plt.show()
```

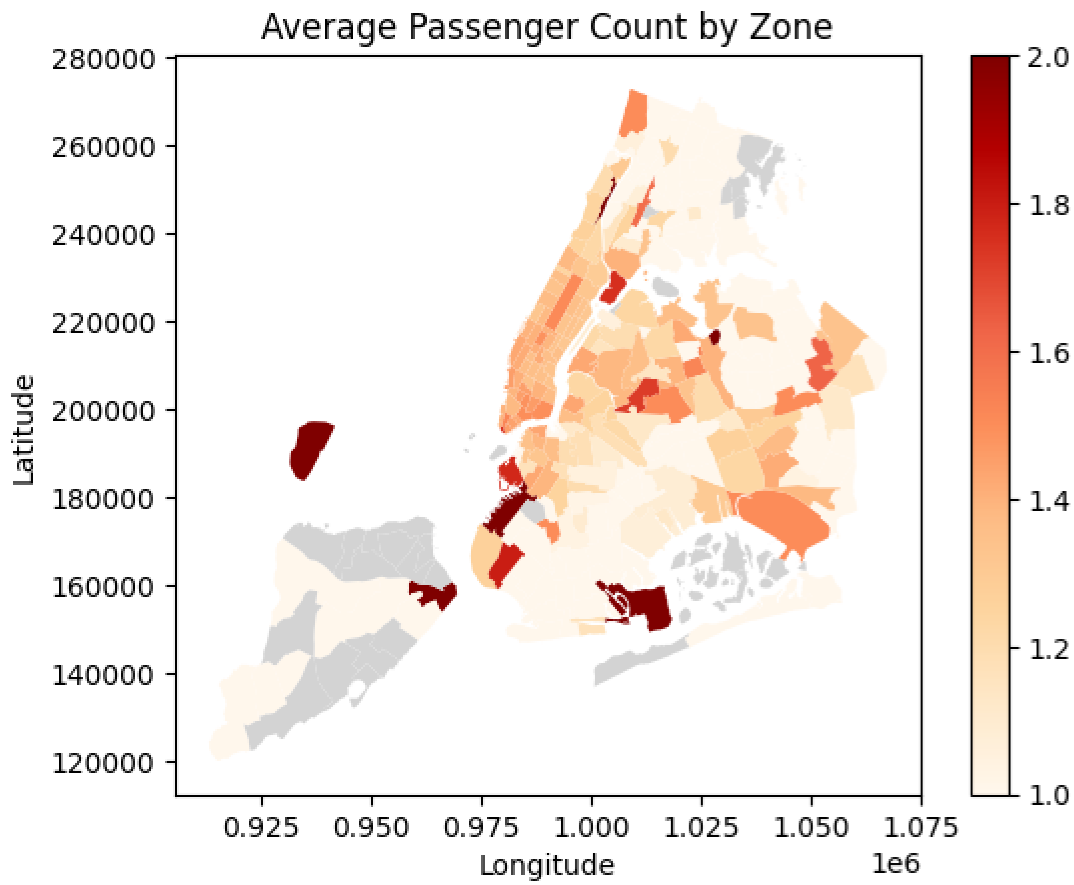


3.2.12. Analyse the variation of passenger counts across zones

```python
# How does passenger count vary across zones

passenger_by_zone = df_cleaned.groupby('PULocationID')['passenger_count'].mean()
print(passenger_by_zone.sort_values().head())
print(passenger_by_zone.sort_values(ascending=False).head())
```

```
PULocationID
3     1.0
5     1.0
9     1.0
11    1.0
15    1.0
Name: passenger_count, dtype: float64
PULocationID
1      2.0
6      2.0
120    2.0
```

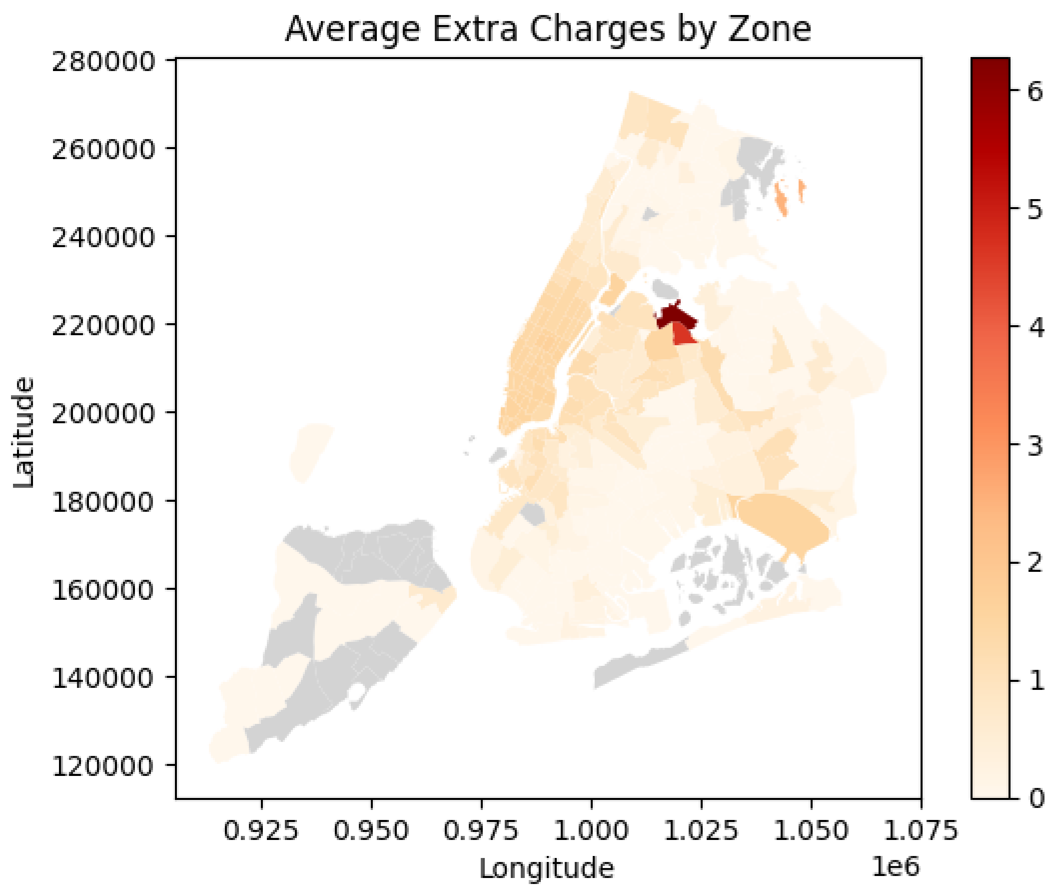
```
228    2.0
253    2.0
Name: passenger_count, dtype: float64

# For a more detailed analysis, we can use the zones_with_trips GeoDataFrame
# Create a new column for the average passenger count in each zone.
avg_passenger_by_zone =
df_cleaned.groupby('PULocationID')['passenger_count'].mean().reset_index()
avg_passenger_by_zone.columns = ['LocationID', 'avg_passenger_count']

# Step 2: Merge the average passenger count with the zones data
zones_with_trips = zones.merge(avg_passenger_by_zone, left_on='LocationID',
right_on='LocationID', how='left')

# Step 3: Visualize the average passenger count across zones
plt.figure(figsize=(12, 8))
zones_with_trips.plot(column='avg_passenger_count', legend=True, cmap='OrRd',
missing_kwds={'color': 'lightgrey'})
plt.title('Average Passenger Count by Zone')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



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

```
# How often is each surcharge applied?

extra_charges_by_zone =
df_cleaned.groupby('PULocationID')['extra'].mean().reset_index()
extra_charges_by_zone.columns = ['LocationID', 'avg_extra_charges']

# Step 2: Merge the extra charges data with the zones data
zones_with_extra = zones.merge(extra_charges_by_zone, left_on='LocationID',
right_on='LocationID', how='left')

# Step 3: Visualize the average extra charges across zones
plt.figure(figsize=(12, 8))
zones_with_extra.plot(column='avg_extra_charges', legend=True, cmap='OrRd',
missing_kwds={'color': 'lightgrey'})
plt.title('Average Extra Charges by Zone')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
```

```
plt.show()
```



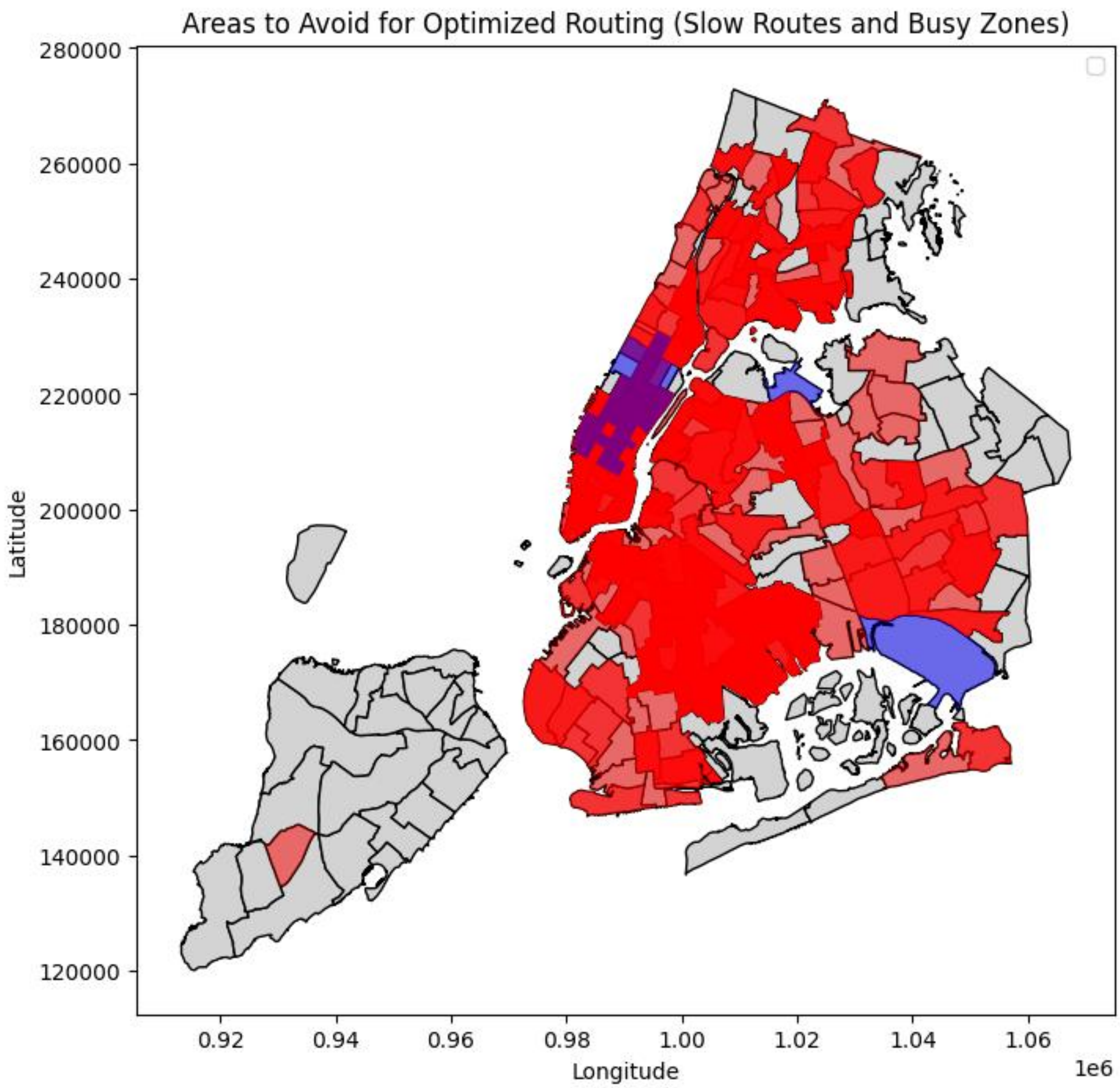
4. Conclusions

4.1. Final Insights and Recommendations

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

```
slow_routes = df_cleaned.groupby(['PULocationID',
'DOLocationID'])['speed'].mean().reset_index()

# Identify the slowest routes (e.g., bottom 10%)
slow_routes = slow_routes[slow_routes['speed'] <
slow_routes['speed'].quantile(0.1)]

# Identify busy hours
```

```python
# Define peak hours (e.g., 7 AM to 10 AM and 5 PM to 8 PM)
peak_hours = list(range(10, 22))
# Filter trips during peak hours
peak_trips = df_cleaned[df_cleaned['pickup_hour'].isin(peak_hours)]

# Group by pickup zone and count trips
busy_zones = peak_trips['PULocationID'].value_counts().reset_index()
busy_zones.columns = ['LocationID', 'trip_count']

# Identify the busiest zones (e.g., top 10%)
busy_zones = busy_zones[busy_zones['trip_count'] >
busy_zones['trip_count'].quantile(0.9)]

# Merge slow routes and busy zones with the zones data
# Merge slow routes (using PULocationID as the zone)
slow_zones = zones.merge(slow_routes, left_on='LocationID',
right_on='PULocationID', how='inner')

# Merge busy zones
busy_zones_map = zones.merge(busy_zones, left_on='LocationID',
right_on='LocationID', how='inner')

# Step 4: Visualize on a map
fig, ax = plt.subplots(figsize=(12, 8))

# Plot all zones
zones.plot(ax=ax, color='lightgrey', edgecolor='black')

# Highlight slow routes (zones)
slow_zones.plot(ax=ax, color='red', alpha=0.5, label='Slow Routes')

# Highlight busy zones
busy_zones_map.plot(ax=ax, color='blue', alpha=0.5, label='Busy Zones')

# Add legend and title
plt.legend()
plt.title('Areas to Avoid for Optimized Routing (Slow Routes and Busy Zones)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```



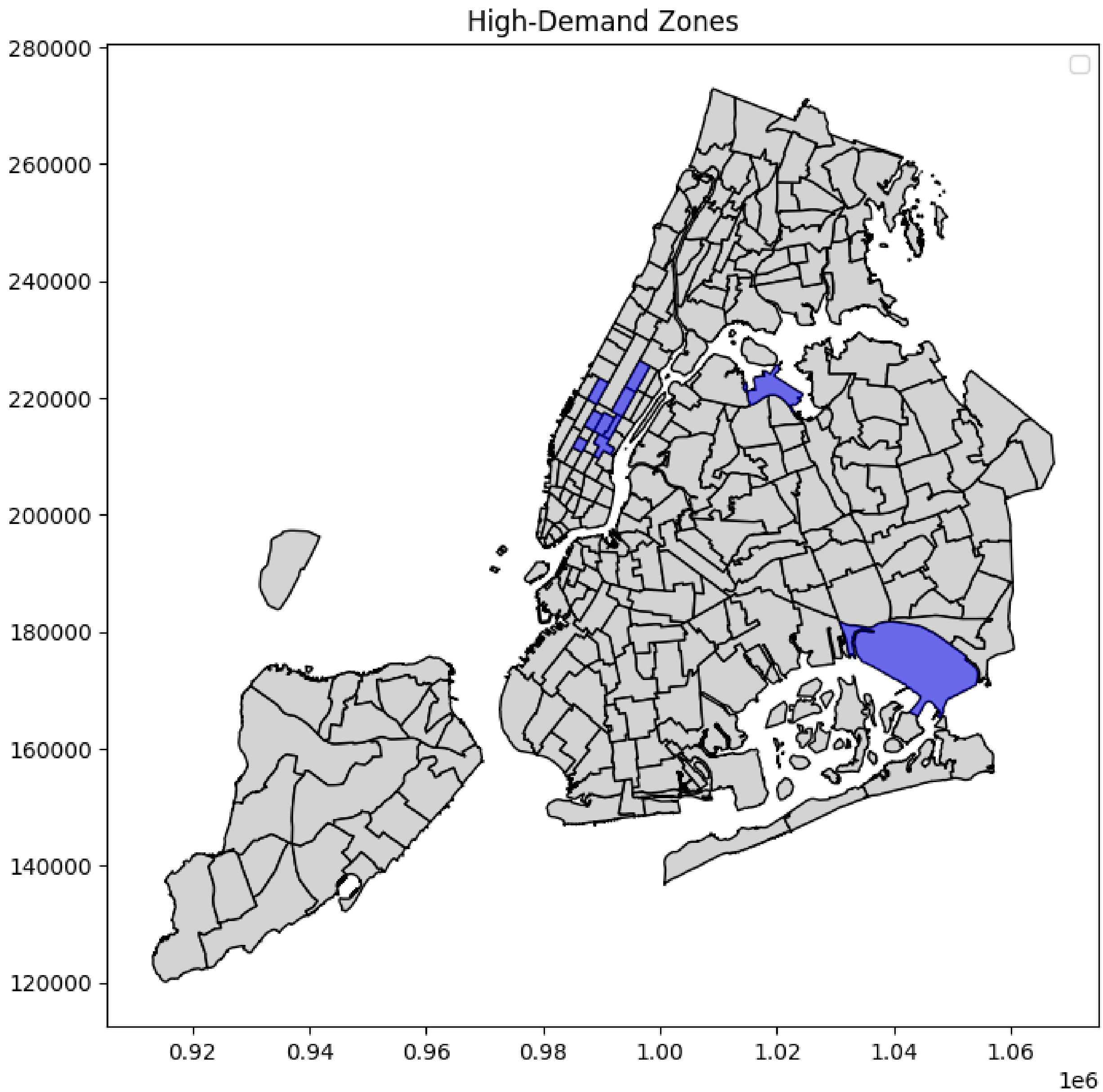
We found that the zones marked in red are slow zones and those in blue are busy zones, we could avoid the red zone during peak hours and place more taxis in busy zones

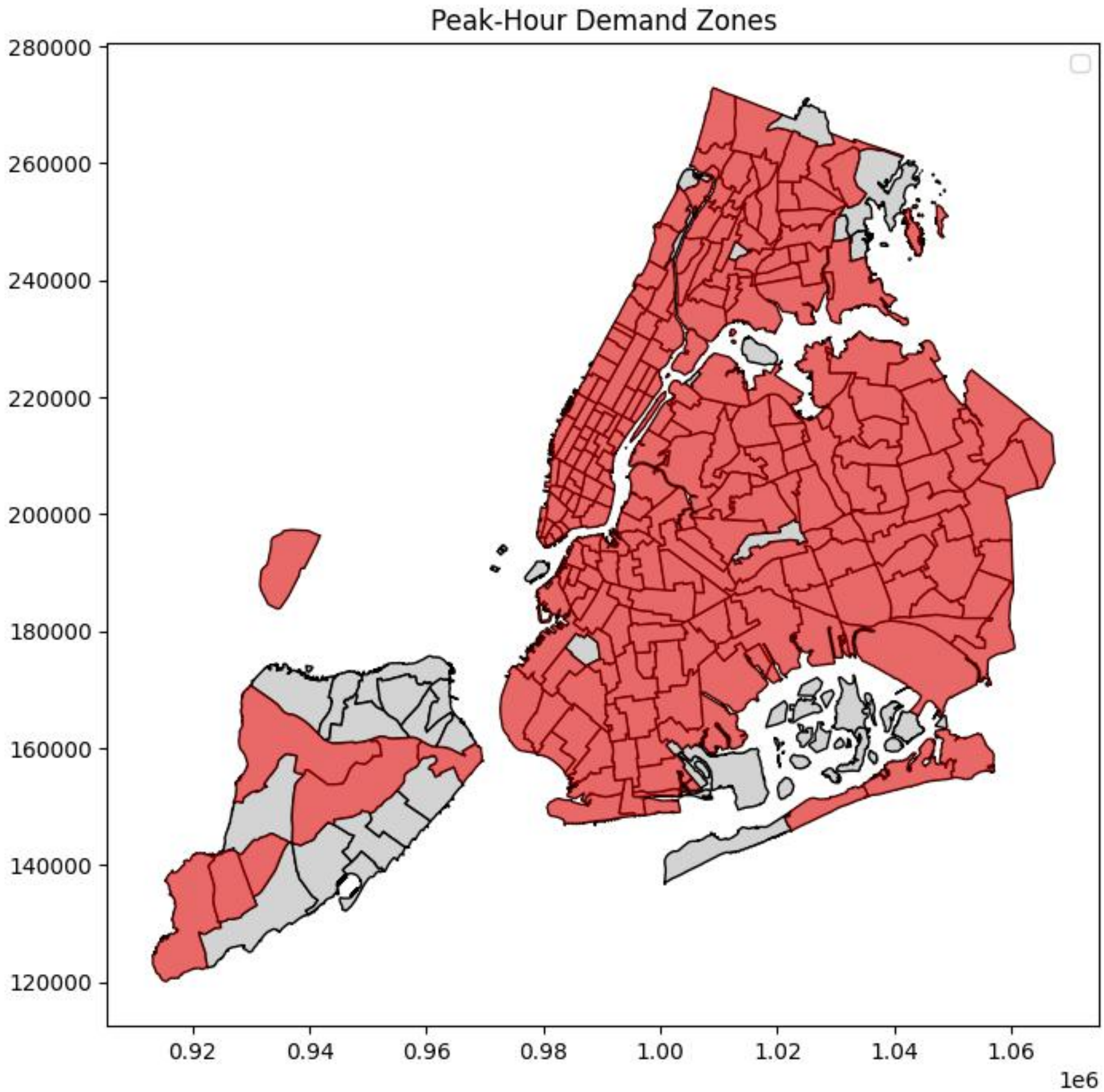
4.1.2. Suggestions on strategically positioning cabs across different zones to make best use of insights uncovered by analysing trip trends across time, days and months.

```
df_cleaned['pickup_month'] = df_cleaned['tpep_pickup_datetime'].dt.month_name()

# Identify high-demand zones
high_demand_zones =
df_cleaned['PULocationID'].value_counts().head(10).reset_index()
```

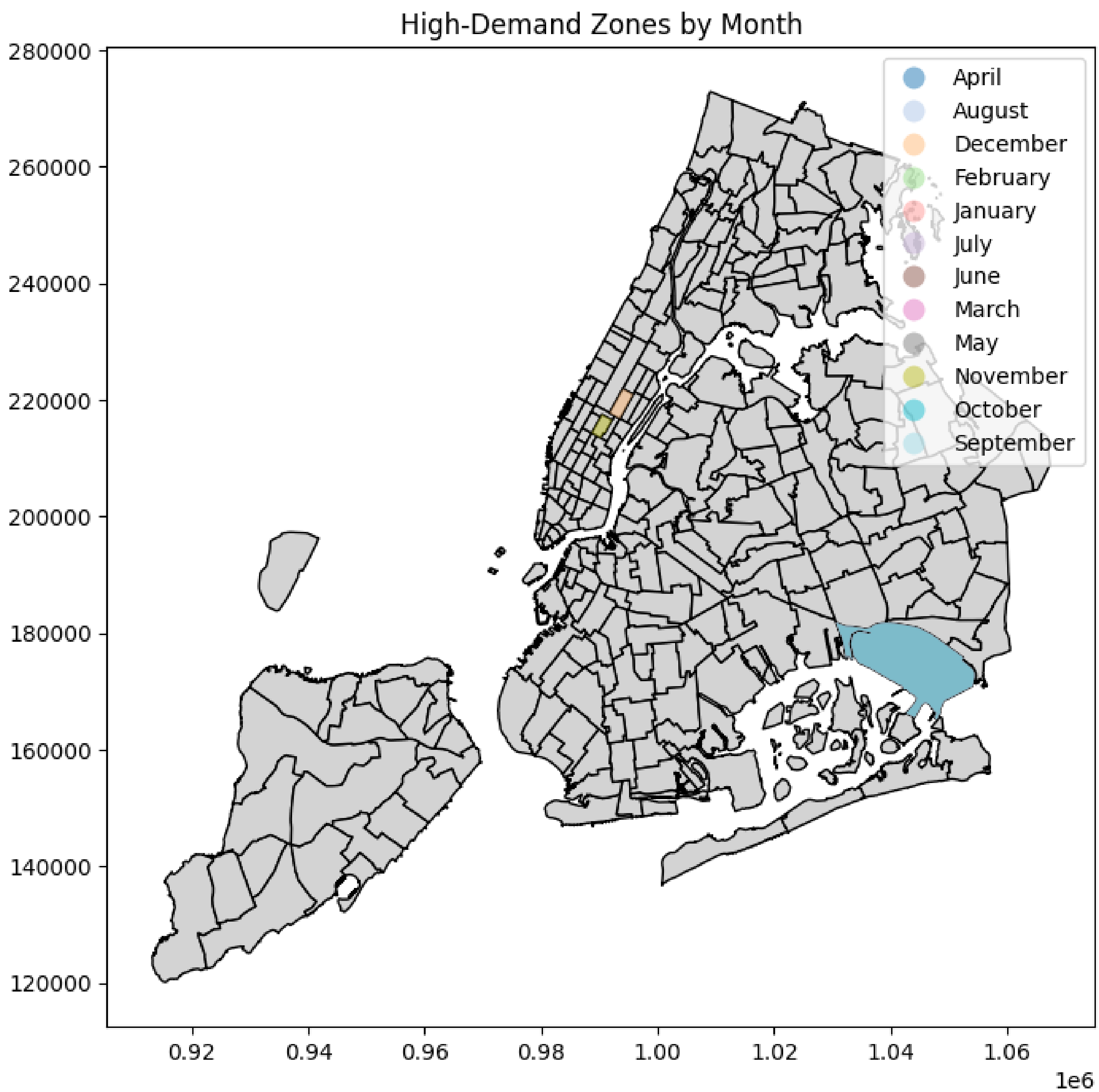
```python
high_demand_zones.columns = ['LocationID', 'trip_count']

# Merge with zones data
high_demand_zones_map = zones.merge(high_demand_zones, left_on='LocationID',
right_on='LocationID', how='inner')

# Step 2: Analyze demand by time of day (peak hours)
peak_hours = list(range(10, 22))
peak_trips = df[df['pickup_hour'].isin(peak_hours)]

# Group by pickup zone and count trips during peak hours
peak_demand_zones = peak_trips['PULocationID'].value_counts().reset_index()
peak_demand_zones.columns = ['LocationID', 'peak_trip_count']

# Merge with zones data
peak_demand_zones_map = zones.merge(peak_demand_zones, left_on='LocationID',
right_on='LocationID', how='inner')

# Step 3: Analyze demand by month
monthly_demand = df.groupby(['pickup_month',
'PULocationID']).size().reset_index(name='trip_count')

# Identify high-demand zones for each month
high_demand_zones_by_month =
monthly_demand.loc[monthly_demand.groupby('pickup_month')['trip_count'].idxmax()]

# Merge with zones data
high_demand_zones_by_month_map = zones.merge(high_demand_zones_by_month,
left_on='LocationID', right_on='PULocationID', how='inner')

# Step 4: Visualize the results
# Plot high-demand zones
fig, ax = plt.subplots(figsize=(12, 8))
zones.plot(ax=ax, color='lightgrey', edgecolor='black')
high_demand_zones_map.plot(ax=ax, color='blue', alpha=0.5, label='High-Demand
Zones')
plt.title('High-Demand Zones')
plt.legend()
plt.show()

# Plot peak-hour demand zones
fig, ax = plt.subplots(figsize=(12, 8))
zones.plot(ax=ax, color='lightgrey', edgecolor='black')
peak_demand_zones_map.plot(ax=ax, color='red', alpha=0.5, label='Peak-Hour Demand
Zones')
```

```python
plt.title('Peak-Hour Demand Zones')
plt.legend()
plt.show()

# Plot high-demand zones by month
fig, ax = plt.subplots(figsize=(12, 8))
zones.plot(ax=ax, color='lightgrey', edgecolor='black')
high_demand_zones_by_month_map.plot(ax=ax, column='pickup_month', legend=True,
cmap='tab20', alpha=0.5)
plt.title('High-Demand Zones by Month')
plt.show()
```





Peak-Hour Demand Zones
280000
260000
240000
220000
200000
180000
160000
140000
120000
0.92
0.94
0.96
0.98
1.00
1.02
1.04
1.06
1e6



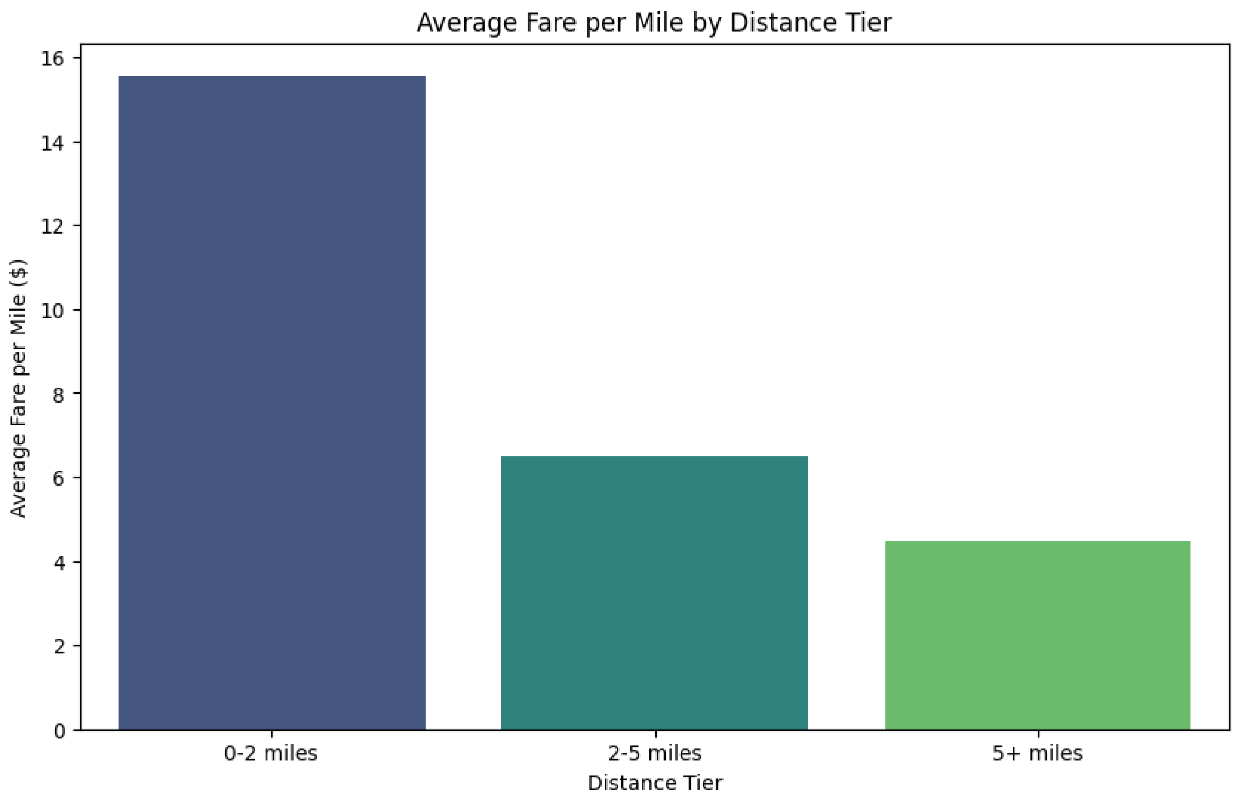
4.1.3. Propose data-driven adjustments to the pricing strategy to maximize revenue while maintaining competitive rates with other vendors.

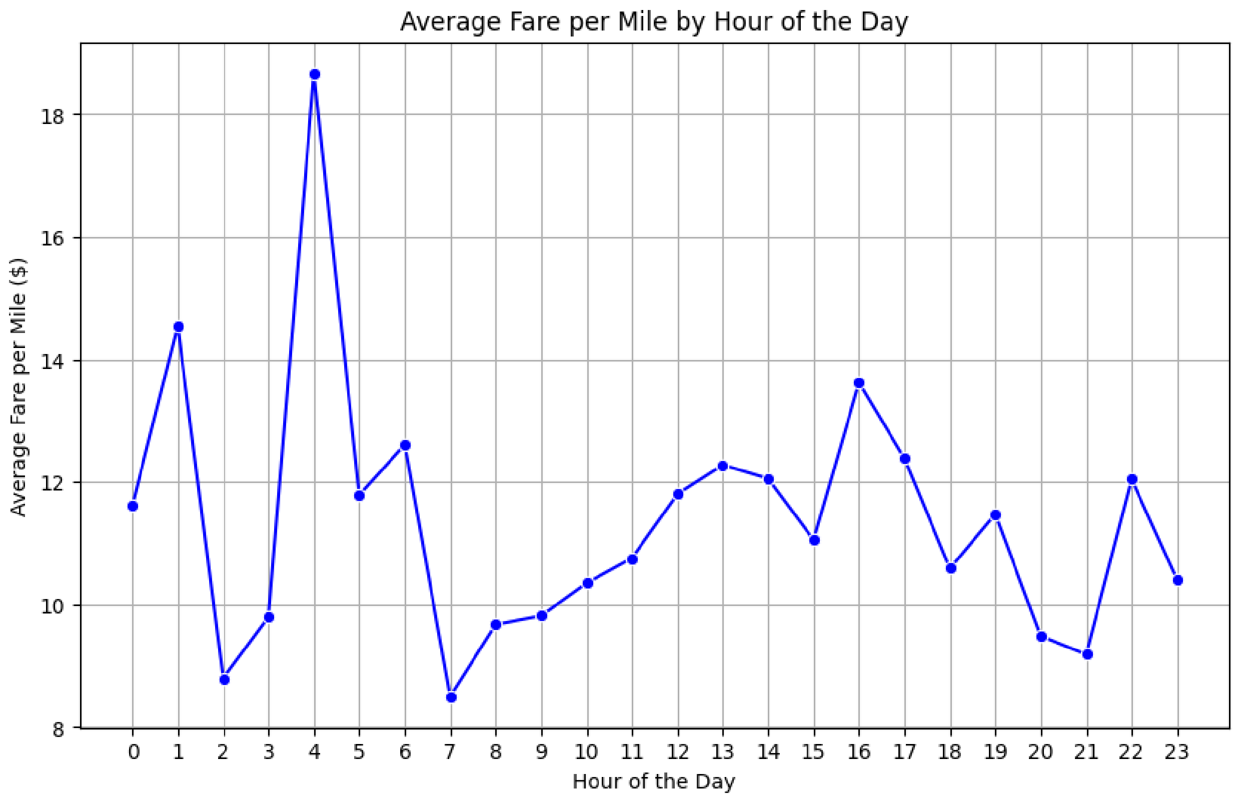
```
# Step 3: Analyze pricing by distance tier
pricing_by_distance =
df_cleaned.groupby('distance_tier')['fare_per_mile'].mean().reset_index()
print("Average Fare per Mile by Distance Tier:")
print(pricing_by_distance)

# Step 4: Analyze pricing by time of day
df_cleaned['pickup_hour'] =
pd.to_datetime(df_cleaned['tpep_pickup_datetime']).dt.hour
```

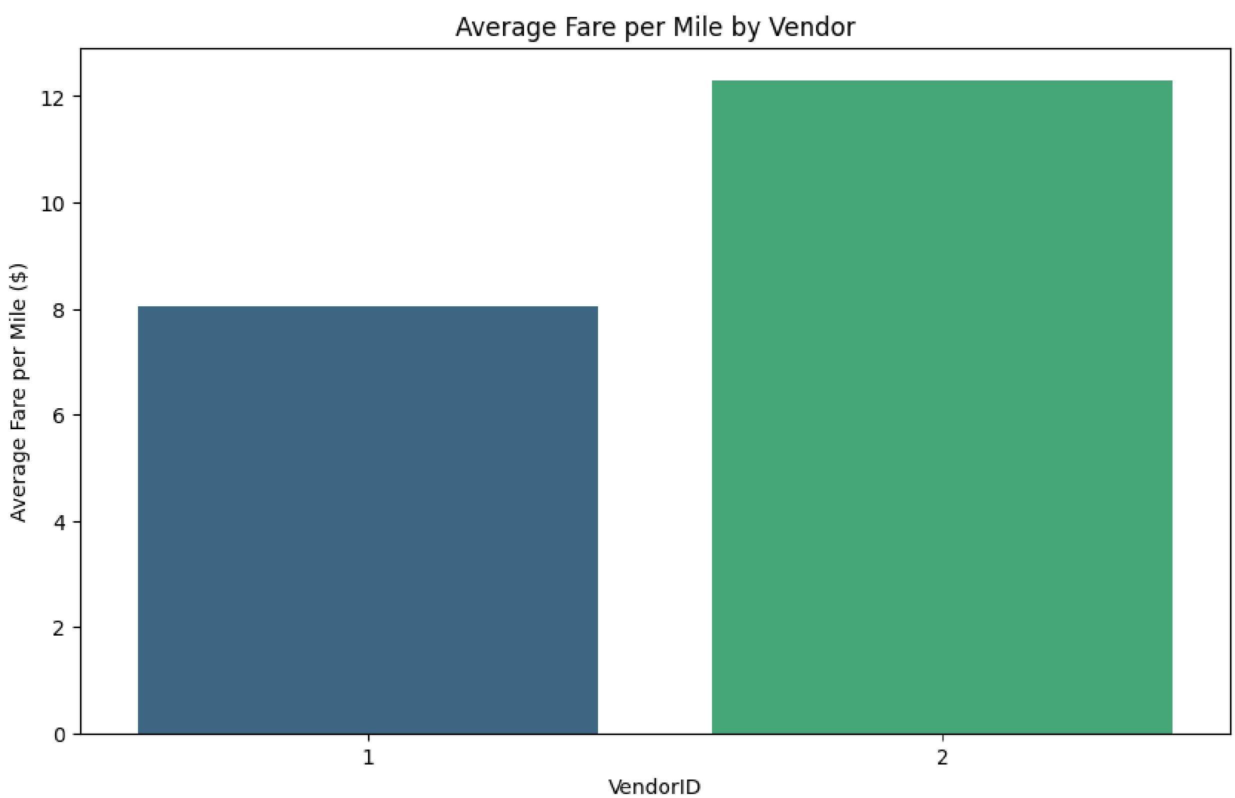
```
pricing_by_hour =
df_cleaned.groupby('pickup_hour')['fare_per_mile'].mean().reset_index()
print("\nAverage Fare per Mile by Hour of the Day:")
print(pricing_by_hour)

# Step 5: Analyze vendor performance
pricing_by_vendor =
df_cleaned.groupby('VendorID')['fare_per_mile'].mean().reset_index()
print("\nAverage Fare per Mile by Vendor:")
print(pricing_by_vendor)

# Step 6: Visualize the results
# Plot pricing by distance tier
plt.figure(figsize=(10, 6))
sns.barplot(x='distance_tier', y='fare_per_mile', data=pricing_by_distance,
palette='viridis')
plt.title('Average Fare per Mile by Distance Tier')
plt.xlabel('Distance Tier')
plt.ylabel('Average Fare per Mile ($)')
plt.show()

# Plot pricing by hour of the day
plt.figure(figsize=(10, 6))
sns.lineplot(x='pickup_hour', y='fare_per_mile', data=pricing_by_hour,
marker='o', color='blue')
plt.title('Average Fare per Mile by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Average Fare per Mile ($)')
plt.xticks(range(0, 24))
plt.grid(True)
plt.show()

# Plot pricing by vendor
plt.figure(figsize=(10, 6))
sns.barplot(x='VendorID', y='fare_per_mile', data=pricing_by_vendor,
palette='viridis')
plt.title('Average Fare per Mile by Vendor')
plt.xlabel('VendorID')
plt.ylabel('Average Fare per Mile ($)')
plt.show()
```

```
Average Fare per Mile by Distance Tier:
  distance_tier  fare_per_mile
0     0-2 miles      15.558469
1     2-5 miles       6.502276
2      5+ miles       4.481891
```

```
Average Fare per Mile by Hour of the Day:
    pickup_hour  fare_per_mile
0             0      11.615445
1             1      14.545308
2             2       8.797200
3             3       9.801487
4             4      18.671254
5             5      11.790490
6             6      12.609890
7             7       8.502808
8             8       9.672973
9             9       9.815616
10           10      10.349524
11           11      10.759231
12           12      11.801775
13           13      12.280112
14           14      12.056407
15           15      11.054626
16           16      13.622629
...
Average Fare per Mile by Vendor:
   VendorID  fare_per_mile
0         1       8.043476
1         2      12.298183
```




Average Fare per Mile by Hour of the Day
Average Fare per Mile ($)
8
10
12
14
16
18
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
Hour of the Day


Average Fare per Mile by Vendor
Average Fare per Mile ($)
0
2
4
6
8
10
12
1
2
VendorID

Pricing Strategy

1. Distance-Based Pricing

Short Trips (0-2 miles): Charge a premium rate due to higher operational costs per mile for short trips.

Example: $15.50 per mile.

Medium Trips (2-5 miles): Charge a moderate rate to balance affordability and profitability.

Example: $6.50 per mile.

Long Trips (5+ miles): Charge a discounted rate to encourage longer trips and maximize vehicle utilization.

Example: $4.50 per mile.

2. Time-Based Pricing

Peak Hours (High Demand): Apply a surge pricing model during peak hours (e.g., 4 AM, 1 AM, 4 PM). Example: Increase fare per mile by 10-20% during peak hours.

Off-Peak Hours (Low Demand): Offer discounted rates during off-peak hours to attract more customers. Example: Reduce fare per mile by 5-10% during off-peak hours.

3. Vendor-Based Pricing

Vendor 1: Maintain competitive pricing to attract cost-sensitive customers.

Example: Keep fare per mile at $8.00.

Vendor 2: Charge a premium rate due to better performance or higher service quality.

Example: Keep fare per mile at $12.30.