NAME - MOHAMMED JAFFER ABBAS

TITLE - EDA NYC YELLOW TAXI DATA ANALYSIS

New York City Yellow Taxi Data

Objective

In this case study you will be learning exploratory data analysis (EDA) with the help of a dataset on yellow taxi rides in New York City. This will enable you to understand why EDA is an important step in the process of data science and machine learning.

Problem Statement

As an analyst at an upcoming taxi operation in NYC, you are tasked to use the 2023 taxi trip data to uncover insights that could help optimise taxi operations. The goal is to analyse patterns in the data that can inform strategic decisions to improve service efficiency, maximise revenue, and enhance passenger experience.

Tasks

You need to perform the following steps for successfully completing this assignment:

1. Data Loading
2. Data Cleaning
3. Exploratory Analysis: Bivariate and Multivariate
4. Creating Visualisations to Support the Analysis
5. Deriving Insights and Stating Conclusions

↳ 3 cells hidden

Data Understanding

The yellow taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts.

The data is stored in Parquet format (*.parquet*). The dataset is from 2009 to 2024. However, for this assignment, we will only be using the data from 2023.

The data for each month is present in a different parquet file. You will get twelve files for each of the months in 2023.

The data was collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers like vendors and taxi hailing apps.

You can find the link to the TLC trip records page here: https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Data Description

You can find the data description here: Data Dictionary

Trip Records

Field Name	description
VendorID	A code indicating the TPEP provider that provided the record. 1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged

Field Name	description
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged
RateCodeID	The final rate code in effect at the end of the trip. 1 = Standard rate 2 = JFK 3 = Newark 4 = Nassau or Westchester 5 = Negotiated fare 6 = Group ride
Store_and_fwd_flag	This flag indicates whether the trip record was held in vehicle memory before sending to the vendor, aka “store and forward,” because the vehicle did not have a connection to th Y= store and forward trip N= not a store and forward trip
Payment_type	A numeric code signifying how the passenger paid for the trip. 1 = Credit card 2 = Cash 3 = No charge 4 = Dispute 5 = Unknown 6 = Voided trip
Fare_amount	The time-and-distance fare calculated by the meter. Extra Miscellaneous extras and surcharges. Currently, this only includes the 0.50 and 1 USD rush hour and overnight charges.
MTA_tax	0.50 USD MTA tax that is automatically triggered based on the metered rate in use.
Improvement_surcharge	0.30 USD improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
Tip_amount	Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
Tolls_amount	Total amount of all tolls paid in trip.
total_amount	The total amount charged to passengers. Does not include cash tips.
Congestion_Surcharge	Total amount collected in trip for NYS congestion surcharge.
Airport_fee	1.25 USD for pick up only at LaGuardia and John F. Kennedy Airports

Although the amounts of extra charges and taxes applied are specified in the data dictionary, you will see that some cases have different values of these charges in the actual data.

Taxi Zones

Each of the trip records contains a field corresponding to the location of the pickup or drop-off of the trip, populated by numbers ranging from 1-263.

These numbers correspond to taxi zones, which may be downloaded as a table or map/shapefile and matched to the trip records using a join.

This is covered in more detail in later sections.

---

1 Data Preparation

[5 marks]

Import Libraries

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


# Import the libraries you will be using for analysis
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns


# Recommended versions
# numpy version: 1.26.4
# pandas version: 2.2.2
# matplotlib version: 3.10.0
# seaborn version: 0.13.2

# Check versions
print("numpy version:", np.__version__)
print("pandas version:", pd.__version__)
print("matplotlib version:", plt.matplotlib.__version__)
print("seaborn version:", sns.__version__)
```

```
numpy version: 2.0.2
pandas version: 2.2.2
matplotlib version: 3.10.0
seaborn version: 0.13.2
```

1.1 Load the dataset

[5 marks]

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/2023-1.parquet")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3041714 entries, 0 to 3066765
Data columns (total 19 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
dtypes: datetime64[us](2), float64(12), int64(4), object(1)
memory usage: 464.1+ MB
```

How many rows are there? Do you think handling such a large number of rows is computationally feasible when we have to combine the data for all twelve months into one?

To handle this, we need to sample a fraction of data from each of the files. How to go about that? Think of a way to select only some portion of the data from each month's file that accurately represents the trends.

Sampling the Data

> One way is to take a small percentage of entries for pickup in every hour of a date. So, for all the days in a month, we can iterate through the hours and select 5% values randomly from those. Use `tpep_pickup_datetime` for this. Separate date and hour from the datetime values and then for each date, select some fraction of trips for each of the 24 hours.

To sample data, you can use the `sample()` method. Follow this syntax:

```
# sampled_data is an empty DF to keep appending sampled data of each hour
# hour_data is the DF of entries for an hour 'X' on a date 'Y'

sample = hour_data.sample(frac = 0.05, random_state = 42)
# sample 0.05 of the hour_data
# random_state is just a seed for sampling, you can define it yourself

sampled_data = pd.concat([sampled_data, sample]) # adding data for this hour to the DF
```

This *sampled_data* will contain 5% values selected at random from each hour.

Note that the code given above is only the part that will be used for sampling and not the complete code required for sampling and combining the data files.

Keep in mind that you sample by date AND hour, not just hour. (Why?)

1.1.1 [5 marks]

Figure out how to sample and combine the files.

Note: It is not mandatory to use the method specified above. While sampling, you only need to make sure that your sampled data represents the overall data of all the months accurately.

```
from google.colab import drive
drive.mount("/content/drive")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Sample the data
# It is recommmended to not load all the files at once to avoid memory overload

import glob

folder_path = "/content/drive/MyDrive/EDA NYC TAXI RECORDS CASE STUDY/Datasets and Dictionary-NYC/Datasets and Dictionary/trip_records/*.parq

# Get a list of all parquet files in the "bb" folder
file_list = glob.glob(folder_path)

# Load, sample, and store each DataFrame in a list
sampled_dfs = [pd.read_parquet(file).sample(frac=0.05, random_state=42) for file in file_list]

# Combine sampled data from all files
final_sampled_df = pd.concat(sampled_dfs, ignore_index=True)

# Check combined data
print(final_sampled_df.info())
print(final_sampled_df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1995989 entries, 0 to 1995988
Data columns (total 20 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  airport_fee            float64
dtypes: datetime64[us](2), float64(13), int64(4), object(1)
memory usage: 304.6+ MB
None
   VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
0         1  2023-11-14 23:00:14   2023-11-14 23:17:00              NaN
1         2  2023-11-05 18:52:19   2023-11-05 18:59:26              1.0
2         2  2023-11-28 07:41:57   2023-11-28 07:55:27              1.0
3         2  2023-11-26 19:10:30   2023-11-26 19:15:38              1.0
4         2  2023-11-02 17:11:16   2023-11-02 18:22:09              1.0

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID  DOLocationID  \
```

```
0          0.00         NaN              None           161           141
1          1.29         1.0                 N           249           246
2          1.71         1.0                 N           140            43
3          0.56         1.0                 N           237           237
4         18.03         2.0                 N           132            48

   payment_type  fare_amount  extra  mta_tax  tip_amount  tolls_amount  \
0             0        14.25    0.0      0.5        0.00          0.00
1             2         9.30    0.0      0.5        0.00          0.00
2             2        14.20    0.0      0.5        0.00          0.00
3             1         7.20    0.0      0.5        1.00          0.00
4             1        70.00    5.0      0.5       17.19          6.94

   improvement_surcharge  total_amount  congestion_surcharge  Airport_fee  \
0                    1.0         18.25                   NaN          NaN
1                    1.0         13.30                   2.5         0.00
2                    1.0         18.20                   2.5         0.00
3                    1.0         12.20                   2.5         0.00
4                    1.0        104.88                   2.5         1.75

   airport_fee
0          NaN
```

```
#- Take a small percentage of entries from each hour of every date.
# Iterating through the monthly data:
#   read a month file -> day -> hour: append sampled data -> move to next hour -> move to next day after 24 hours -> move to next month file
# Create a single dataframe for the year combining all the monthly data

# Select the folder having data files
import os

# Select the folder having data files
os.chdir('/content/drive/MyDrive/EDA NYC TAXI RECORDS CASE STUDY/Datasets and Dictionary-NYC/Datasets and Dictionary/trip_records')

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

# initialise 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

        # We will store the sampled data for the current date in this df by appending the sampled data from each hour to this
        # After completing iteration through each date, we will append this data to the final dataframe.
        sampled_data = pd.DataFrame()

        # Loop through dates and then loop through every hour of each date

            # Iterate through each hour of the selected date

                # Sample 5% of the hourly data randomly

                # add data of this hour to the dataframe

        # Concatenate the sampled data of all the dates to a single dataframe
        df = pd.concat([df, sampled_data])# we initialised this empty DF earlier

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

After combining the data files into one DataFrame, convert the new DataFrame to a CSV or parquet file and store it to use directly.

Ideally, you can try keeping the total entries to around 250,000 to 300,000.

```
# Store the df in csv/parquet
# df.to_parquet('')
final_sampled_df.to_parquet("sampled_nyc_taxi.parquet", index=False)
```

2 Data Cleaning

[30 marks]

Now we can load the new data directly.

```
# Load the new data file
df = pd.read_parquet("/content/drive/MyDrive/EDA NYC TAXI RECORDS CASE STUDY/Datasets and Dictionary-NYC/Datasets and Dictionary/trip_record
```

```
df.head(12)
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID
0	1	2023-11-14 23:00:14	2023-11-14 23:17:00	NaN	0.00	NaN	None	161
1	2	2023-11-05 18:52:19	2023-11-05 18:59:26	1.0	1.29	1.0	N	249
2	2	2023-11-28 07:41:57	2023-11-28 07:55:27	1.0	1.71	1.0	N	140
3	2	2023-11-26 19:10:30	2023-11-26 19:15:38	1.0	0.56	1.0	N	237
4	2	2023-11-02 17:11:16	2023-11-02 18:22:09	1.0	18.03	2.0	N	132
5	2	2023-11-04 23:02:27	2023-11-04 23:22:31	1.0	1.53	1.0	N	48
6	2	2023-11-10 19:17:41	2023-11-10 19:27:10	1.0	1.14	1.0	N	141
7	2	2023-11-04 15:15:58	2023-11-04 15:23:27	1.0	1.13	1.0	N	234
8	2	2023-11-04 22:38:59	2023-11-04 22:48:59	3.0	1.32	1.0	N	68
9	1	2023-11-12 17:03:23	2023-11-12 17:09:19	1.0	1.50	1.0	N	143
10	1	2023-11-06 15:48:21	2023-11-06 16:06:20	NaN	2.00	NaN	None	234
11	2	2023-11-29 22:29:37	2023-11-29 23:06:22	1.0	7.35	1.0	N	50

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1995989 entries, 0 to 1995988
Data columns (total 20 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  airport_fee            float64
dtypes: datetime64[us](2), float64(13), int64(4), object(1)
memory usage: 304.6+ MB
```

2.1 Fixing Columns

[10 marks]

Fix/drop any columns as you seem necessary in the below sections

2.1.1 [2 marks]

Fix the index and drop unnecessary columns

```
# Fix the index and drop any columns that are not needed

columns_to_drop = ["store_and_fwd_flag",  "extra", "mta_tax","improvement_surcharge"]

df.drop(columns=columns_to_drop, inplace=True, errors="ignore")
df.reset_index(drop=True, inplace=True)  # Fix the index
print(df.columns)
```

```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'RatecodeID', 'PULocationID',
       'DOLocationID', 'payment_type', 'fare_amount', 'tip_amount',
       'tolls_amount', 'total_amount', 'congestion_surcharge', 'Airport_fee',
       'airport_fee'],
      dtype='object')
```

2.1.2 [3 marks]

There are two airport fee columns. This is possibly an error in naming columns. Let's see whether these can be combined into a single column.

```
# Combine the two airport fee columns

# Ensure all values are summed correctly, handling NaNs
df["Airport_fee"] = df["Airport_fee"].fillna(0) + df["airport_fee"].fillna(0)

# Drop the redundant column
df.drop(columns=["airport_fee"], inplace=True, errors="ignore")
```

2.1.3 [5 marks]

Fix columns with negative (monetary) values

```
# check where values of fare amount are negative

neg_fares = df[df["fare_amount"] < 0]
print(neg_fares)
```

```
Empty DataFrame
Columns: [VendorID, tpep_pickup_datetime, tpep_dropoff_datetime, passenger_count, trip_distance, RatecodeID, PULocationID, DOLocationID,
Index: []
```

Did you notice something different in the `RatecodeID` column for above records?

```
# Analyse RatecodeID for the negative fare amounts

negative_fares = df[df["fare_amount"] < 0]

if negative_fares.empty:
    print("No negative fare amounts found")
else:
    print(negative_fares["RatecodeID"].value_counts())
```

```
No negative fare amounts found
```

```
# Find which columns have negative values

negative_cols = df.select_dtypes(include=["number"]).columns  # only numeric columns
negative_values = df[negative_cols].lt(0).any()  # if any column has negative values

# Display columns with negative values
negative_columns = negative_values[negative_values].index.tolist()

if not negative_columns:
    print("No negative values found\n")
else:
    print("Columns with negative values:", negative_columns)
    print(df[negative_columns].describe())
```

```
Columns with negative values: ['tolls_amount', 'total_amount', 'congestion_surcharge', 'Airport_fee']
       tolls_amount  total_amount  congestion_surcharge   Airport_fee
count  1.995989e+06  1.995989e+06          1.927597e+06  1.995989e+06
mean   6.016428e-01  2.898268e+01          2.308864e+00  1.386304e-01
std    2.194102e+00  2.306150e+01          6.646610e-01  4.584819e-01
min   -6.940000e+00 -1.769000e+01         -2.500000e+00 -1.750000e+00
```

```
25%    0.000000e+00  1.596000e+01          2.500000e+00  0.000000e+00
50%    0.000000e+00  2.100000e+01          2.500000e+00  0.000000e+00
75%    0.000000e+00  3.100000e+01          2.500000e+00  0.000000e+00
max    1.320400e+02  2.203140e+03          2.500000e+00  1.750000e+00
```

```
# fix these negative values

# Replace negative values with 0
df[df.select_dtypes(include=["number"]).columns] = df.select_dtypes(include=["number"]).clip(lower=0)
```

2.2 Handling Missing Values

[10 marks]

2.2.1 [2 marks]

Find the proportion of missing values in each column

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

# Calculate the proportion of missing values for each column
missing_values = df.isnull().mean() * 100

# Display only columns with missing values
missing_values = missing_values[missing_values > 0].sort_values(ascending=False)

print("Proportion of missing values :\n",missing_values)
```

```
Proportion of missing values :
 passenger_count         3.426472
RatecodeID              3.426472
congestion_surcharge    3.426472
dtype: float64
```

2.2.2 [3 marks]

Handling missing values in `passenger_count`

```
# Display the rows with null values

missing_rows = df[df.isnull().any(axis=1)]
print("Rows with missing values:\n",missing_rows)
```

```
Rows with missing values:
           VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
0               1  2023-11-14 23:00:14   2023-11-14 23:17:00              NaN
10              1  2023-11-06 15:48:21   2023-11-06 16:06:20              NaN
20              1  2023-11-19 18:29:46   2023-11-19 18:56:43              NaN
42              1  2023-11-01 18:52:15   2023-11-01 19:03:08              NaN
68              2  2023-11-18 20:46:07   2023-11-18 21:07:32              NaN
...           ...                  ...                   ...              ...
1995941         2  2023-03-30 21:29:17   2023-03-30 21:36:03              NaN
1995943         2  2023-05-02 11:39:06   2023-05-02 11:55:42              NaN
1995955         2  2023-09-22 12:09:00   2023-09-22 13:23:35              NaN
1995962         2  2023-09-05 07:02:27   2023-09-05 07:40:52              NaN
1995977         1  2023-08-13 04:08:25   2023-08-13 04:26:48              NaN

         trip_distance  RatecodeID  PULocationID  DOLocationID  payment_type  \
0                 0.00         NaN           161           141             0
10                2.00         NaN           234           163             0
20                0.00         NaN           262           244             0
42                1.20         NaN           239           142             0
68                4.35         NaN           262            79             0
...                ...         ...           ...           ...           ...
1995941           1.64         NaN           236           239             0
1995943           1.71         NaN           163           233             0
1995955          18.11         NaN           163           132             0
1995962          17.80         NaN            79           132             0
1995977           0.00         NaN           148           163             0

         fare_amount  tip_amount  tolls_amount  total_amount  \
0              14.25        0.00          0.00         18.25
10             15.60        3.92          0.00         23.52
20             26.74        0.00          0.00         30.74
42             11.40        3.58          0.00         21.48
68             23.71        4.16          0.00         31.87
...              ...         ...           ...           ...
```

```
1995941        12.26        3.25          0.00         19.51
1995943        13.86        3.57          0.00         21.43
1995955        85.19       20.61          6.94        116.74
1995962        55.84        8.98          0.00         68.82
1995977        22.82        0.00          0.00         26.82

         congestion_surcharge  Airport_fee
0                         NaN          0.0
10                        NaN          0.0
20                        NaN          0.0
42                        NaN          0.0
68                        NaN          0.0
...                       ...          ...
1995941                   NaN          0.0
1995943                   NaN          0.0
1995955                   NaN          0.0
1995962                   NaN          0.0
1995977                   NaN          0.0

[68392 rows x 15 columns]
```

Did you find zeroes in passenger_count? Handle these.

```
# Replace zero values in 'passenger_count' with the median
df.loc[df["passenger_count"] == 0, "passenger_count"] = df["passenger_count"].median()
# Display rows with any null values after handling passenger_count
missing_rows_after = df[df.isnull().any(axis=1)]
```

2.2.3 [2 marks]

Handle missing values in `RatecodeID`

```
# Fix missing values in 'RatecodeID'
# Impute missing values in 'RatecodeID' with the most frequent value (mode)

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

2.2.4 [3 marks]

Impute NaN in `congestion_surcharge`

```
# handle null values in congestion_surcharge
# Impute missing values in 'congestion_surcharge' with the median
df["congestion_surcharge"].fillna(df["congestion_surcharge"].median(), inplace=True)
```

Are there missing values in other columns? Did you find NaN values in some other set of columns? Handle those missing values below.

```
# Handle any remaining missing values
# Fill numeric columns with median & categorical columns with mode
for col in df.columns:
    if df[col].isnull().sum() > 0:  # process columns with missing values
        if df[col].dtype == "object":
            df[col].fillna(df[col].mode()[0], inplace=True)
        else:  # Numeric columns
            df[col].fillna(df[col].median(), inplace=True)
```

2.3 Handling Outliers

[10 marks]

Before we start fixing outliers, let's perform outlier analysis.

```
# Describe the data and check if there are any potential outliers present
# Check for potential out of place values in various columns

# Display summary statistics
print("Dataset Summary:\n", df.describe())

# Define a function to detect potential outliers using the IQR method
def detect_outliers(df, cols):
    outlier_info = {}
```

```
        for col in cols:
            Q1 = df[col].quantile(0.25)
            Q3 = df[col].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR

            # Count outliers
            outlier_count = ((df[col] < lower_bound) | (df[col] > upper_bound)).sum()
            if outlier_count > 0:
                outlier_info[col] = outlier_count

        return outlier_info

    # Select numeric columns for outlier detection
    numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
    outliers = detect_outliers(df, numeric_cols)

    # Display detected outliers
    print("\nPotential Outliers :")
    for col, count in outliers.items():
        print(f" -> {col} : {count} outliers")
```

```
Dataset Summary:
             VendorID        tpep_pickup_datetime       tpep_dropoff_datetime  \
count  1.995989e+06                     1995989                     1995989
mean   1.736363e+00  2023-07-02 20:06:37.795917  2023-07-02 20:24:10.640049
min    1.000000e+00         2001-01-01 00:06:49         2001-01-01 15:42:11
25%    1.000000e+00         2023-04-02 16:10:52         2023-04-02 16:32:39
50%    2.000000e+00         2023-06-27 15:27:26         2023-06-27 15:47:07
75%    2.000000e+00         2023-10-06 19:52:30         2023-10-06 20:11:16
max    6.000000e+00         2023-12-31 23:58:14         2024-01-01 23:02:22
std    4.460855e-01                         NaN                         NaN

       passenger_count  trip_distance    RatecodeID  PULocationID  \
count     1.995989e+06   1.995989e+06  1.995989e+06  1.995989e+06
mean      1.372384e+00   4.216713e+00  1.629388e+00  1.652181e+02
min       1.000000e+00   0.000000e+00  1.000000e+00  1.000000e+00
25%       1.000000e+00   1.050000e+00  1.000000e+00  1.320000e+02
50%       1.000000e+00   1.790000e+00  1.000000e+00  1.620000e+02
75%       1.000000e+00   3.400000e+00  1.000000e+00  2.340000e+02
max       9.000000e+00   1.865141e+05  9.900000e+01  2.650000e+02
std       8.657841e-01   2.716442e+02  7.367115e+00  6.400023e+01

       DOLocationID  payment_type   fare_amount    tip_amount  tolls_amount  \
count  1.995989e+06  1.995989e+06  1.995989e+06  1.995989e+06  1.995989e+06
mean   1.640002e+02  1.162986e+00  1.989948e+01  3.559998e+00  6.016463e-01
min    1.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    1.130000e+02  1.000000e+00  9.300000e+00  1.000000e+00  0.000000e+00
50%    1.620000e+02  1.000000e+00  1.350000e+01  2.850000e+00  0.000000e+00
75%    2.340000e+02  1.000000e+00  2.233000e+01  4.450000e+00  0.000000e+00
max    2.650000e+02  4.000000e+00  2.194700e+03  3.000000e+02  1.320400e+02
std    6.988474e+01  5.077539e-01  1.850121e+01  4.075621e+00  2.194095e+00

       total_amount  congestion_surcharge   Airport_fee
count  1.995989e+06          1.995989e+06  1.995989e+06
mean   2.898287e+01          2.315503e+00  1.386446e-01
min    0.000000e+00          0.000000e+00  0.000000e+00
25%    1.596000e+01          2.500000e+00  0.000000e+00
50%    2.100000e+01          2.500000e+00  0.000000e+00
75%    3.100000e+01          2.500000e+00  0.000000e+00
max    2.203140e+03          2.500000e+00  1.750000e+00
std    2.306124e+01          6.536076e-01  4.584540e-01

Potential Outliers :
 -> VendorID : 485 outliers
 -> passenger_count : 446450 outliers
 -> trip_distance : 264132 outliers
 -> RatecodeID : 109134 outliers
 -> payment_type : 423773 outliers
 -> fare_amount : 203177 outliers
 -> tip_amount : 153692 outliers
 -> tolls_amount : 163311 outliers
 -> total_amount : 230444 outliers
 -> congestion_surcharge : 147302 outliers
 -> Airport_fee : 170201 outliers
```

2.3.1 [10 marks]

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

Some points you can look for:

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

These are just some suggestions. You can handle outliers in any way you wish, using the insights from above outlier analysis.

How will you fix each of these values? Which ones will you drop and which ones will you replace?

First, let us remove 7+ passenger counts as there are very less instances.

```
# remove passenger_count > 6
# Remove rows where passenger_count is greater than 6
df = df[df["passenger_count"] <= 6]

# Verify if the filtering was successful
print("Updated passenger count distribution:\n", df["passenger_count"].value_counts())
```

```
Updated passenger count distribution:
 passenger_count
1.0    1549539
2.0     291483
3.0      72253
4.0      40533
5.0      25250
6.0      16915
Name: count, dtype: int64
```

```
# Continue with outlier handling

def remove_outliers(df, cols):
    for col in cols:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Remove rows with outliers
        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]

    return df

# Select numeric columns for outlier removal
numeric_cols = df.select_dtypes(include=[np.number]).columns.tolist()
df = remove_outliers(df, numeric_cols)

# Verify the dataset after outlier removal
print(f"Outliers handled successfully.\nUpdated dataset summary:\n{df.describe()}")
```

```
Outliers handled successfully.
Updated dataset summary:
             VendorID        tpep_pickup_datetime       tpep_dropoff_datetime  \
count  971414.000000                      971414                      971414
mean        1.732748  2023-06-30 10:46:04.354300  2023-06-30 10:58:50.035069
min         1.000000         2022-12-31 14:39:43         2022-12-31 14:43:37
25%         1.000000         2023-03-30 08:53:43  2023-03-30 09:06:45.250000
50%         2.000000  2023-06-23 16:16:33.500000  2023-06-23 16:31:03.500000
75%         2.000000  2023-10-05 15:43:55.750000  2023-10-05 15:58:30.750000
max         2.000000         2023-12-31 23:58:02         2024-01-01 23:02:22
std         0.442525                         NaN                         NaN

       passenger_count  trip_distance  RatecodeID   PULocationID  \
count         971414.0  971414.000000    971414.0  971414.000000
mean               1.0       1.776947         1.0     171.251083
min                1.0       0.000000         1.0       4.000000
25%                1.0       0.990000         1.0     140.000000
50%                1.0       1.500000         1.0     163.000000
75%                1.0       2.300000         1.0     236.000000
max                1.0       6.720000         1.0     265.000000
std                0.0       1.083936         0.0      63.995283

        DOLocationID  payment_type   fare_amount     tip_amount  tolls_amount  \
count  971414.000000      971414.0  971414.00000  971414.000000      971414.0
```

```
mean      169.586606          1.0      12.85576       3.080375          0.0
min         4.000000          1.0       0.00000       0.000000          0.0
25%       137.000000          1.0       8.60000       2.000000          0.0
50%       163.000000          1.0      12.10000       3.000000          0.0
75%       236.000000          1.0      16.30000       4.000000          0.0
max       265.000000          1.0      29.60000       7.210000          0.0
std        67.162404          0.0       5.44602       1.437078          0.0

        total_amount  congestion_surcharge  Airport_fee
count  971414.000000              971414.0     971414.0
mean       20.718933                   2.5          0.0
min         4.000000                   2.5          0.0
25%        15.750000                   2.5          0.0
50%        19.650000                   2.5          0.0
75%        24.800000                   2.5          0.0
max        38.940000                   2.5          0.0
std         6.438226                   0.0          0.0
```

```
# Do any columns need standardising?

from sklearn.preprocessing import StandardScaler

# Identify numeric columns (excluding categorical ones like IDs)
numeric_cols = df.select_dtypes(include=["number"]).columns.tolist()

# Decide which columns need standardization
columns_to_standardize = [col for col in numeric_cols if df[col].std() > 1]

if columns_to_standardize:
    print("Columns that need standardization:", columns_to_standardize)
```

```
Columns that need standardization: ['trip_distance', 'PULocationID', 'DOLocationID', 'fare_amount', 'tip_amount', 'total_amount']
```

3 Exploratory Data Analysis

[90 marks]

```
df.columns.tolist()
```

```
['VendorID',
 'tpep_pickup_datetime',
 'tpep_dropoff_datetime',
 'passenger_count',
 'trip_distance',
 'RatecodeID',
 'PULocationID',
 'DOLocationID',
 'payment_type',
 'fare_amount',
 'tip_amount',
 'tolls_amount',
 'total_amount',
 'congestion_surcharge',
 'Airport_fee']
```

3.1 General EDA: Finding Patterns and Trends

[40 marks]

3.1.1 [3 marks]

Categorise the varaibles into Numerical or Categorical.

- `VendorID`:
- `tpep_pickup_datetime`:
- `tpep_dropoff_datetime`:
- `passenger_count`:
- `trip_distance`:
- `RatecodeID`:
- `PULocationID`:
- `DOLocationID`:
- `payment_type`:

- `pickup_hour`:
- `trip_duration`:

The following monetary parameters belong in the same category, is it categorical or numerical?

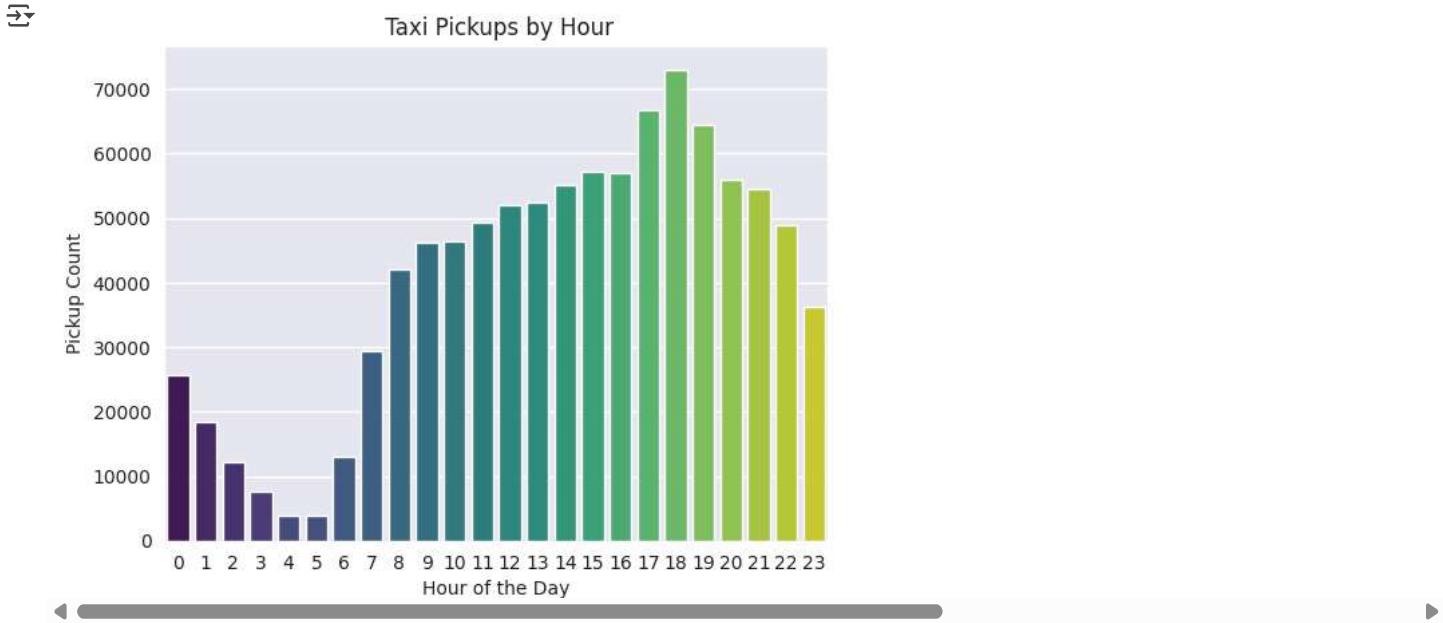
- `fare_amount`
- `extra`
- `mta_tax`
- `tip_amount`
- `tolls_amount`
- `improvement_surcharge`
- `total_amount`
- `congestion_surcharge`
- `airport_fee`

Temporal Analysis

3.1.2 [5 marks]

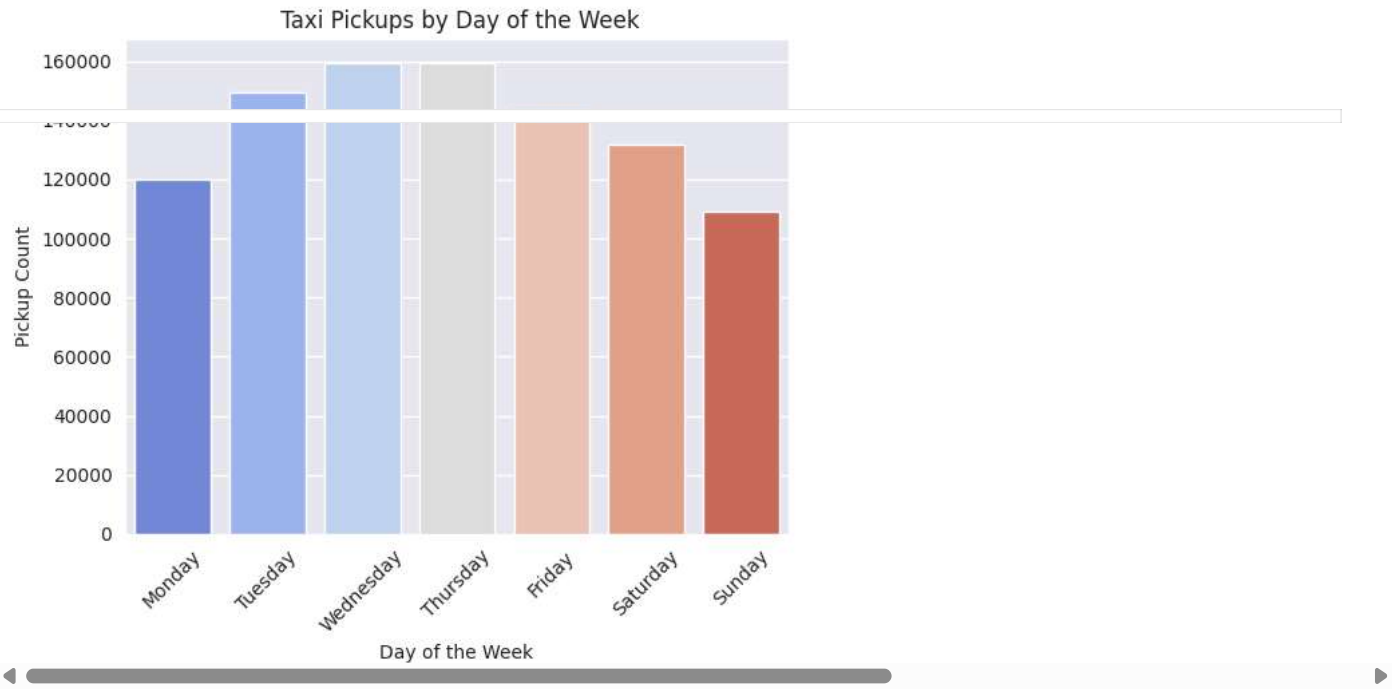
Analyse the distribution of taxi pickups by hours, days of the week, and months.

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

# Convert datetime columns 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()
df["pickup_month"] = df["tpep_pickup_datetime"].dt.month_name()

# Set up visualization style
sns.set_style("darkgrid")

# Plot Hourly Trends
sns.countplot(x="pickup_hour", data=df, palette="viridis")
plt.title(" Taxi Pickups by Hour")
plt.xlabel("Hour of the Day")
plt.ylabel("Pickup Count")
plt.xticks(range(24))
plt.show()
```

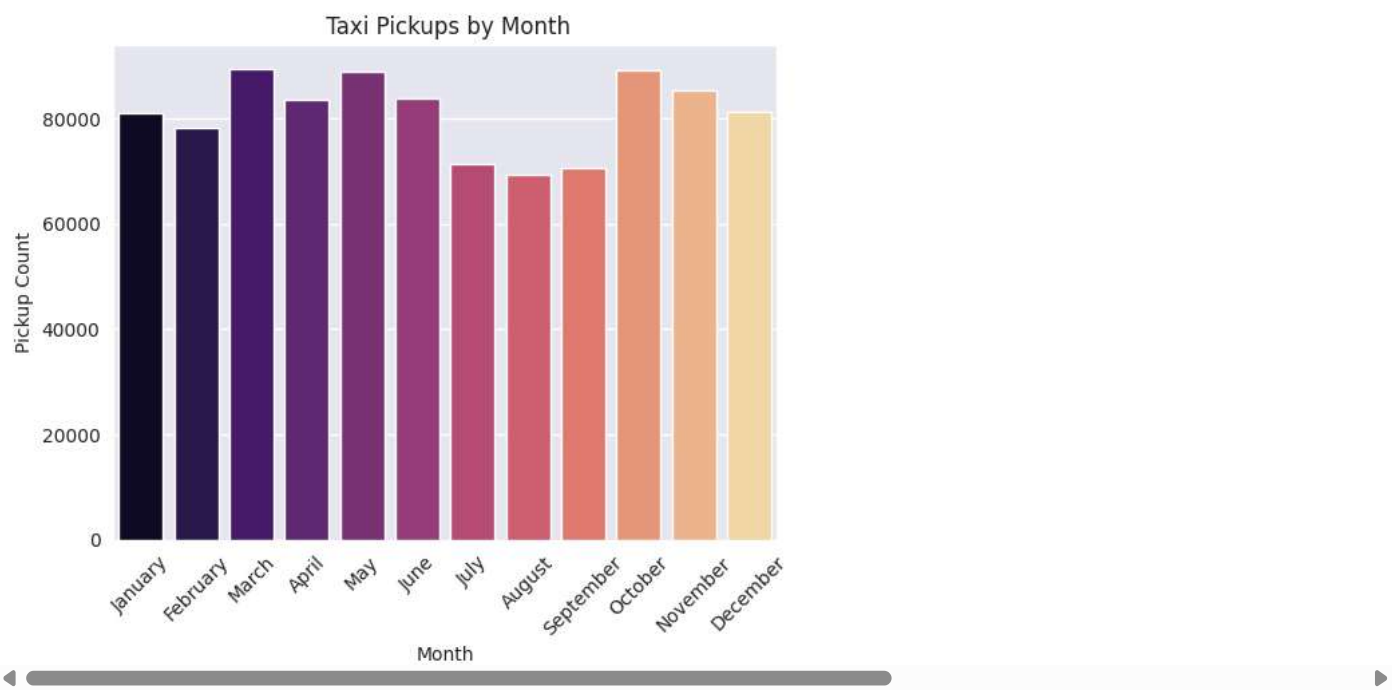



```
# Find and show the daily trends in taxi pickups (days of the week)
# Plot Daily Trends
sns.countplot(x="pickup_day", data=df, order=["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"], palette="coolwa
plt.title(" Taxi Pickups by Day of the Week")
plt.xlabel("Day of the Week")
```

```
plt.ylabel("Pickup Count")
plt.xticks(rotation=45)
plt.show()
```




```
# Show the monthly trends in pickups

# Plot Monthly Trends
sns.countplot(x="pickup_month", data=df, order=[
    "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"
], palette="magma")
plt.title(" Taxi Pickups by Month")
plt.xlabel("Month")
plt.ylabel("Pickup Count")
plt.xticks(rotation=45)
plt.show()
```

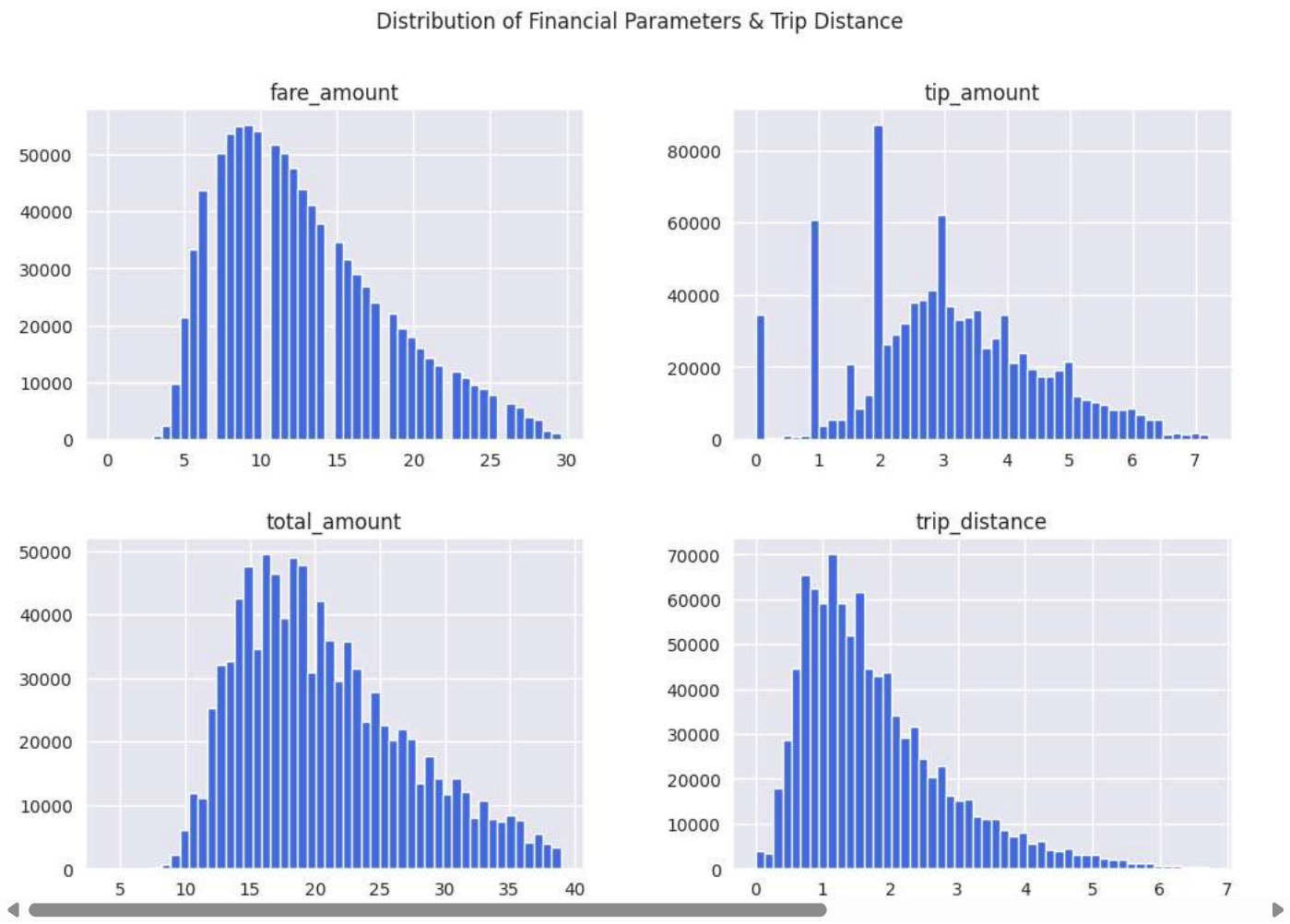


Financial Analysis

Take a look at the financial parameters like `fare_amount`, `tip_amount`, `total_amount`, and also `trip_distance`. Do these contain zero/negative values?

```
# Analyse the above parameters

# Select relevant columns
financial_cols = ["fare_amount", "tip_amount", "total_amount", "trip_distance"]

# Check for zero or negative values
zero_negative_counts = (df[financial_cols] <= 0).sum()

# Display results
print("Count of Zero/Negative Values in Key Parameters:")
print(zero_negative_counts)

# Visualize distribution of financial parameters
df[financial_cols].hist(bins=50, figsize=(12, 8), layout=(2, 2), color="royalblue")
plt.suptitle("Distribution of Financial Parameters & Trip Distance")
plt.show()
```

```
Count of Zero/Negative Values in Key Parameters:
fare_amount          3
tip_amount       32448
total_amount         0
trip_distance     2779
dtype: int64
```



Do you think it is beneficial to create a copy DataFrame leaving out the zero values from these?

3.1.3 [2 marks]

Filter out the zero values from the above columns.

Note: The distance might be 0 in cases where pickup and drop is in the same zone. Do you think it is suitable to drop such cases of zero distance?

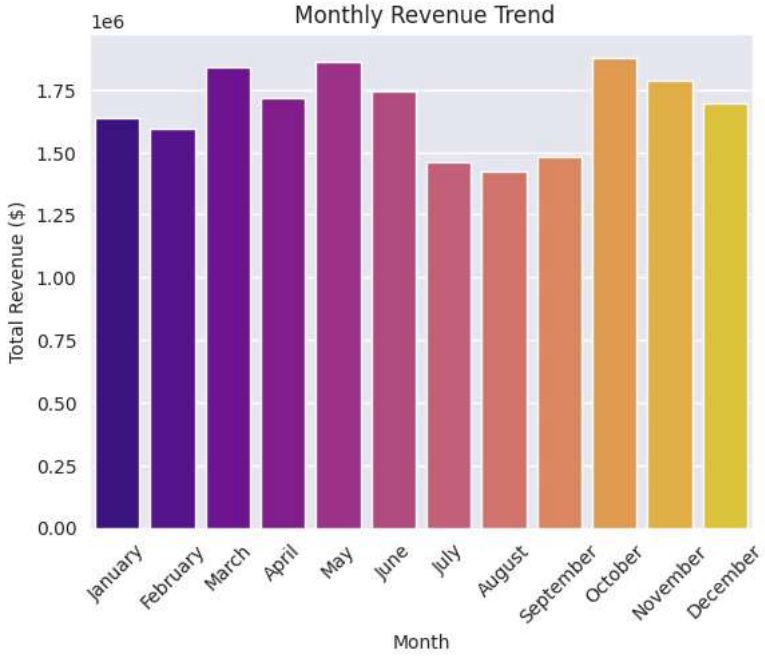
```
# Create a df with non zero entries for the selected parameters.

# Define columns to filter
financial_cols = ["fare_amount", "tip_amount", "total_amount", "trip_distance"]

# Keep only rows where all selected parameters are greater than zero
df_filtered = df[(df[financial_cols] > 0).all(axis=1)].copy()

# Verify the changes
print(f"New dataset shape after filtering: {df_filtered.shape}")
print("Summary of filtered data:\n", df_filtered[financial_cols].describe())
```

```
New dataset shape after filtering: (936328, 18)
Summary of filtered data:
           fare_amount     tip_amount   total_amount  trip_distance
count  936328.000000  936328.000000  936328.000000  936328.000000
mean       12.855236       3.187304      20.824502       1.780774
std         5.420516       1.340569       6.424099       1.075386
min         2.800000       0.010000       7.020000       0.010000
25%         8.600000       2.160000      15.950000       1.000000
50%        12.100000       3.000000      19.680000       1.510000
75%        16.300000       4.020000      24.840000       2.300000
max        29.600000       7.210000      38.940000       6.720000
```

3.1.4 [3 marks]

Analyse the monthly revenue (`total_amount`) trend

```
# Group data by month and analyse monthly revenue

# Group by month and sum total revenue
monthly_revenue = df.groupby("pickup_month")["total_amount"].sum().reindex([
    "January", "February", "March", "April", "May", "June",
    "July", "August", "September", "October", "November", "December"
])

# Plot Monthly Revenue Trend
sns.barplot(x=monthly_revenue.index, y=monthly_revenue.values, palette="plasma")
plt.title(" Monthly Revenue Trend")
plt.xlabel("Month")
plt.ylabel("Total Revenue ($)")
plt.xticks(rotation=45)
plt.show()

# Display revenue values
print("Monthly Revenue Breakdown:\n", monthly_revenue)
```



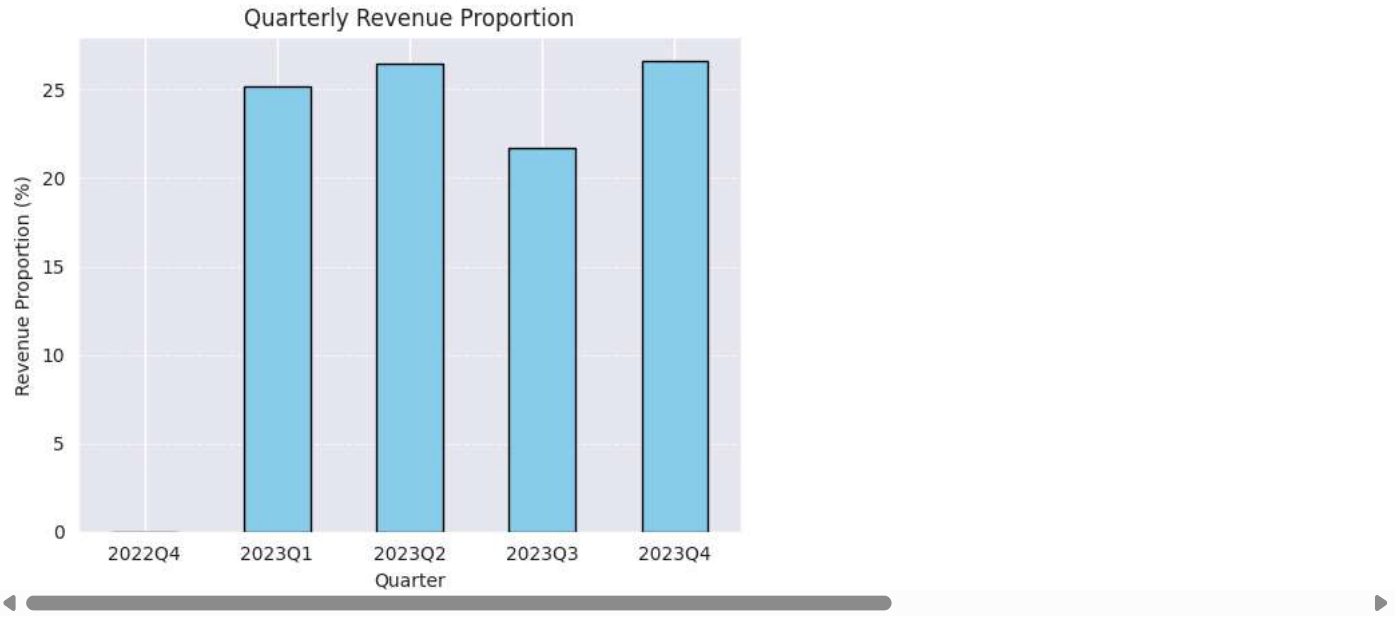

```
Monthly Revenue Breakdown:
 pickup_month
January      1635585.97
February     1595924.48
March        1840307.64
April        1719557.32
May          1862026.71
June         1741318.26
July         1462771.20
August       1423877.52
September    1482901.47
October      1879708.44
November     1788059.52
December     1694623.06
```

3.1.5 [3 marks]

Show the proportion of each quarter of the year in the revenue

```
# Calculate proportion of each quarter

# Map months to quarters
df["quarter"] = df["tpep_pickup_datetime"].dt.to_period("Q")

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

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

print("Quarterly Revenue Proportion (%):\n", quarterly_revenue_proportion)

quarterly_revenue_proportion.plot(kind="bar", color="skyblue", edgecolor="black")

plt.xlabel("Quarter")
plt.ylabel("Revenue Proportion (%)")
plt.title("Quarterly Revenue Proportion")
plt.xticks(rotation=0)
plt.grid(axis="y", linestyle="--", alpha=0.7)

# Show plot
plt.show()
```

```
Quarterly Revenue Proportion (%):
 quarter
2022Q4     0.000064
2023Q1    25.199500
2023Q2    26.447020
2023Q3    21.710258
2023Q4    26.643157
Freq: Q-DEC, Name: total_amount, dtype: float64
```

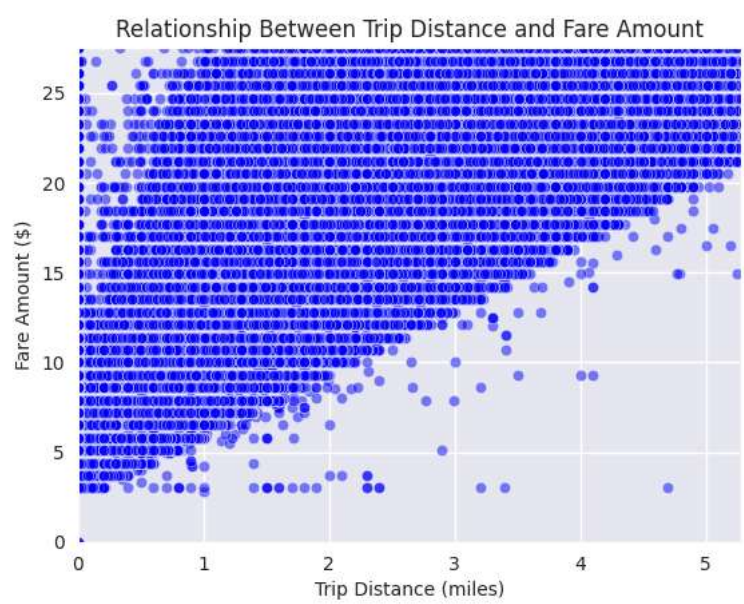


3.1.6 [3 marks]

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

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

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

sns.scatterplot(x=df["trip_distance"], y=df["fare_amount"], alpha=0.5, color="blue")
plt.title("Relationship Between Trip Distance and Fare Amount")
plt.xlabel("Trip Distance (miles)")
plt.ylabel("Fare Amount ($)")
plt.ylim(0, df["fare_amount"].quantile(0.99))  #exclude extreme outliers
plt.xlim(0, df["trip_distance"].quantile(0.99))
plt.show()

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



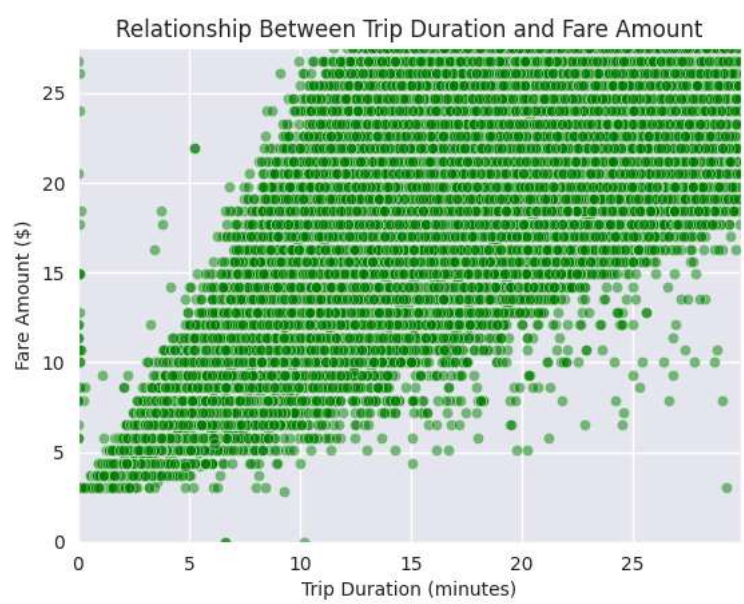
Correlation between Trip Distance and Fare Amount: 0.87

3.1.7 [5 marks]

Find and visualise the correlation between:

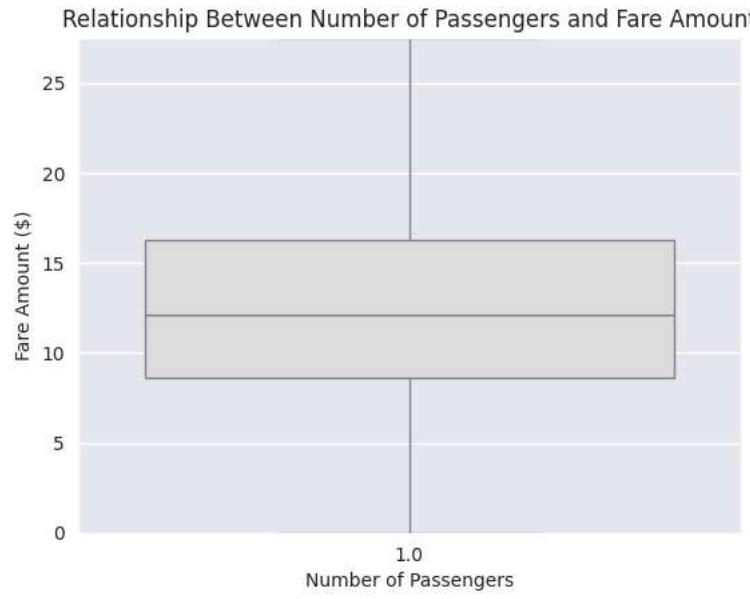
1. `fare_amount` and trip duration (pickup time to dropoff time)
2. `fare_amount` and `passenger_count`
3. `tip_amount` and `trip_distance`

```
# Show relationship between fare and trip duration

# Calculate trip duration in minutes
df["trip_duration"] = (df["tpep_dropoff_datetime"] - df["tpep_pickup_datetime"]).dt.total_seconds() / 60

sns.scatterplot(x=df["trip_duration"], y=df["fare_amount"], alpha=0.5, color="green")
plt.title("Relationship Between Trip Duration and Fare Amount")
plt.xlabel("Trip Duration (minutes)")
plt.ylabel("Fare Amount ($)")
plt.ylim(0, df["fare_amount"].quantile(0.99))  # Limit y-axis to exclude extreme outliers
plt.xlim(0, df["trip_duration"].quantile(0.99))  # Limit x-axis for better visibility
plt.show()

# Calculate correlation
correlation = df["trip_duration"].corr(df["fare_amount"])
print(f"Correlation between Trip Duration and Fare Amount: {correlation:.2f}")
```




```
Correlation between Trip Duration and Fare Amount: 0.18
```

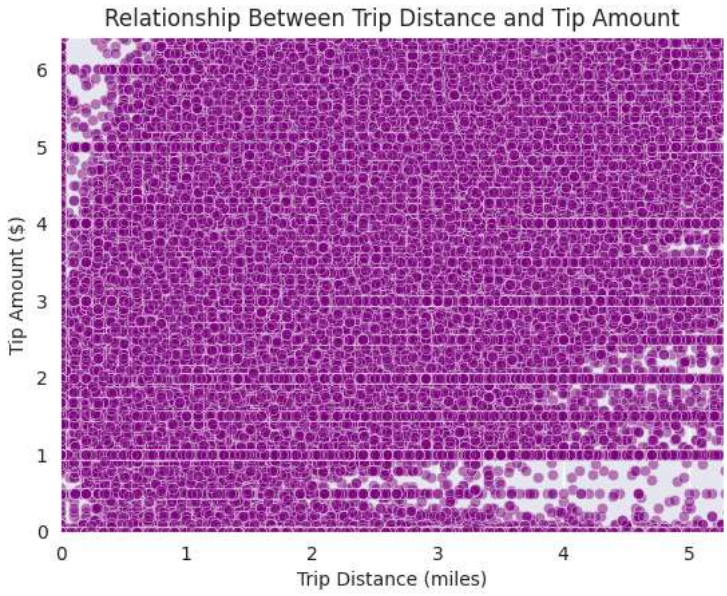
```
# Show relationship between fare and number of passengers

sns.boxplot(x=df["passenger_count"], y=df["fare_amount"], palette="coolwarm")
plt.title("Relationship Between Number of Passengers and Fare Amount")
plt.xlabel("Number of Passengers")
plt.ylabel("Fare Amount ($)")
plt.ylim(0, df["fare_amount"].quantile(0.99))  # Limit y-axis to exclude extreme outliers
plt.show()

# Calculate average fare per passenger count
fare_by_passenger = df.groupby("passenger_count")["fare_amount"].mean().reset_index()
print(f"Average Fare Amount by Number of Passengers:\n {fare_by_passenger}")
```




```
Average Fare Amount by Number of Passengers:
    passenger_count  fare_amount
0              1.0    12.85576
```

```
# Show relationship between tip and trip distance

sns.scatterplot(x=df["trip_distance"], y=df["tip_amount"], alpha=0.5, color="purple")
plt.title("Relationship Between Trip Distance and Tip Amount")
plt.xlabel("Trip Distance (miles)")
plt.ylabel("Tip Amount ($)")
plt.ylim(0, df["tip_amount"].quantile(0.99))  # Limit y-axis to exclude extreme outliers
plt.xlim(0, df["trip_distance"].quantile(0.99))  # Limit x-axis for better visibility
plt.show()
```

```
correlation = df["trip_distance"].corr(df["tip_amount"])
print(f"Correlation between Trip Distance and Tip Amount: {correlation:.2f}")

# Calculate average tip per mile
df["tip_per_mile"] = df["tip_amount"] / df["trip_distance"]
print("\nAverage Tip Per Mile:\n", df["tip_per_mile"].describe())
```




```
Correlation between Trip Distance and Tip Amount: 0.48

Average Tip Per Mile:
 count    9.712730e+05
mean              inf
std               NaN
min      0.000000e+00
25%      1.372549e+00
50%      1.969849e+00
75%      2.729560e+00
max               inf
```

3.1.8 [3 marks]

Analyse the distribution of different payment types (`payment_type`)

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

payment_counts = df["payment_type"].value_counts()
print(f"{payment_counts}")
```

```
payment_type
1    971414
Name: count, dtype: int64
```

- 1= Credit card
- 2= Cash
- 3= No charge
- 4= Dispute

Geographical Analysis

For this, you have to use the *taxi_zones.shp* file from the *taxi_zones* folder.

There would be multiple files inside the folder (such as *.shx, .sbx, .sbn* etc). You do not need to import/read any of the files other than the shapefile, *taxi_zones.shp*.

Do not change any folder structure - all the files need to be present inside the folder for it to work.

The folder structure should look like this:

```
Taxi Zones
|- taxi_zones.shp.xml
|- taxi_zones.prj
|- taxi_zones.sbn
|- taxi_zones.shp
|- taxi_zones.dbf
|- taxi_zones.shx
|- taxi_zones.sbx
```

You only need to read the `taxi_zones.shp` file. The *shp* file will utilise the other files by itself.

We will use the *GeoPandas* library for geopgraphical analysis

```
import geopandas as gpd
```

More about geopandas and shapefiles: About

Reading the shapefile is very similar to *Pandas*. Use `gpd.read_file()` function to load the data (*taxi_zones.shp*) as a GeoDataFrame.
Documentation: Reading and Writing Files

```
# !pip install geopandas
```

3.1.9 [2 marks]
Load the shapefile and display it.

```
# import geopandas as gpd


# Read the shapefile using geopandas
import geopandas as gpd

# Path to the shapefile (update the path if needed)
shapefile_path = "/content/drive/MyDrive/EDA NYC TAXI RECORDS CASE STUDY/Datasets and Dictionary-NYC/Datasets and Dictionary/taxi_zones/taxi

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

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

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry
0	1	0.116357	0.000782	Newark Airport	1	EWR	POLYGON ((933100.918 192536.086, 933091.011 19...
1	2	0.433470	0.004866	Jamaica Bay	2	Queens	MULTIPOLYGON (((1033269.244 172126.008, 103343...
2	3	0.084341	0.000314	Allerton/Pelham Gardens	3	Bronx	POLYGON ((1026308.77 256767.698, 1026495.593 2...
3	4	0.043567	0.000112	Alphabet City	4	Manhattan	POLYGON ((992073.467 203714.076, 992068.667 20...
4	5	0.092146	0.000498	Arden Heights	5	Staten Island	POLYGON ((935843.31 144283.336, 936046.565 144...

Now, if you look at the DataFrame created, you will see columns like: `OBJECTID`, `Shape_Leng`, `Shape_Area`, `zone`, `LocationID`, `borough`, `geometry`.

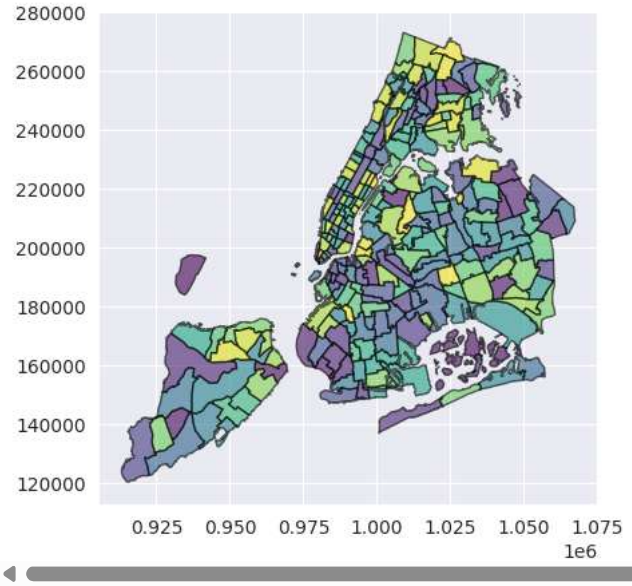
Now, the `locationID` here is also what we are using to mark pickup and drop zones in the trip records.

The geometric parameters like shape length, shape area and geometry are used to plot the zones on a map.

This can be easily done using the `plot()` method.

```
print(zones.info())
zones.plot(edgecolor="black", cmap="viridis", alpha=0.6)
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 263 entries, 0 to 262
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   OBJECTID    263 non-null    int32
 1   Shape_Leng  263 non-null    float64
 2   Shape_Area  263 non-null    float64
 3   zone        263 non-null    object
 4   LocationID  263 non-null    int32
 5   borough     263 non-null    object
 6   geometry    263 non-null    geometry
dtypes: float64(2), geometry(1), int32(2), object(2)
memory usage: 12.5+ KB
None
<Axes: >
```



Now, you have to merge the trip records and zones data using the location IDs.

3.1.10 [3 marks]

Merge the zones data into trip data using the `locationID` and `PULocationID` columns.

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

# Ensure the columns are of the same data type
zones["LocationID"] = zones["LocationID"].astype(int)
df["PULocationID"] = df["PULocationID"].astype(int)

# Merge the datasets
merged_df = df.merge(zones, left_on="PULocationID", right_on="LocationID", how="left")
print(merged_df)
```

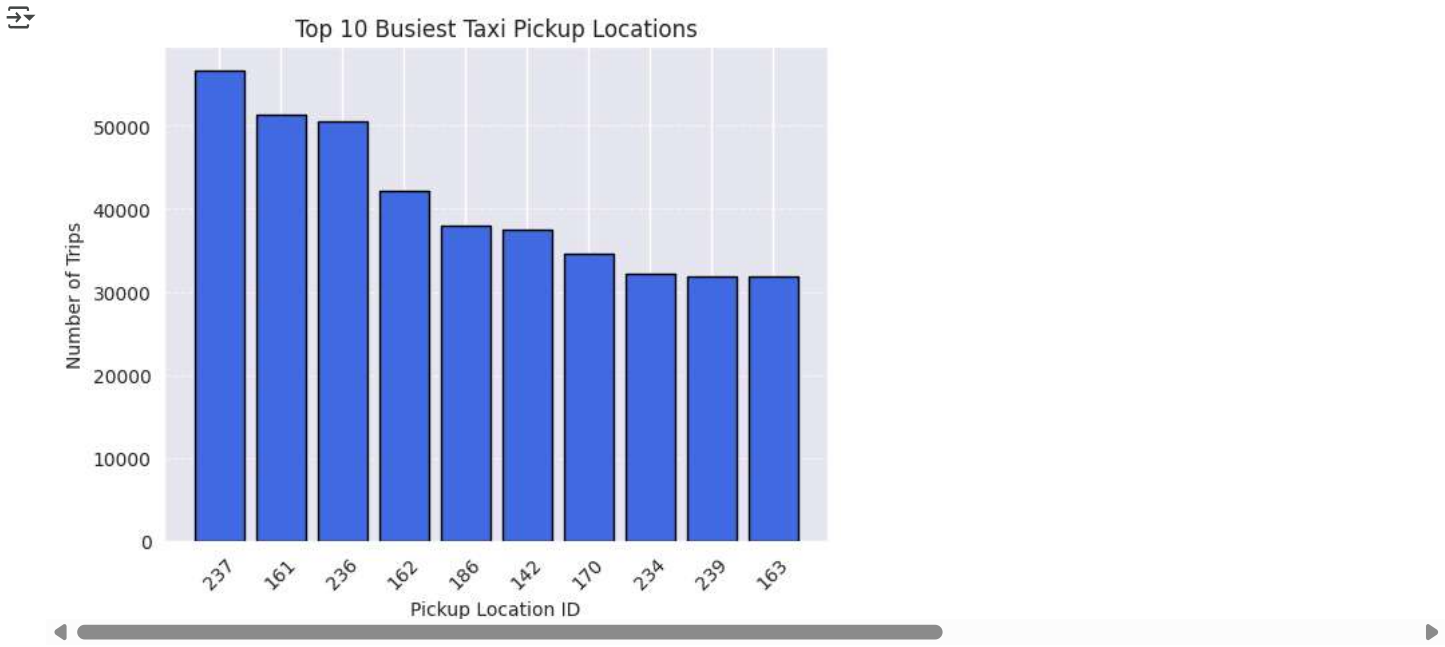
```
        VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
0              2  2023-11-26 19:10:30   2023-11-26 19:15:38              1.0
1              2  2023-11-04 23:02:27   2023-11-04 23:22:31              1.0
2              2  2023-11-04 15:15:58   2023-11-04 15:23:27              1.0
3              1  2023-11-12 17:03:23   2023-11-12 17:09:19              1.0
4              2  2023-11-17 14:30:05   2023-11-17 14:38:02              1.0
...          ...                  ...                   ...              ...
971409         1  2023-10-29 14:22:03   2023-10-29 14:39:35              1.0
971410         1  2023-09-26 15:48:22   2023-09-26 16:19:13              1.0
971411         1  2023-09-26 17:13:43   2023-09-26 17:23:27              1.0
971412         2  2023-11-17 20:12:22   2023-11-17 20:35:06              1.0
971413         2  2023-04-25 16:40:42   2023-04-25 16:48:01              1.0

        trip_distance  RatecodeID  PULocationID  DOLocationID  payment_type  \
0                0.56         1.0           237           237             1
1                1.53         1.0            48           158             1
2                1.13         1.0           234           170             1
3                1.50         1.0           143           264             1
4                1.03         1.0           107            90             1
...               ...         ...           ...           ...           ...
971409           2.90         1.0           170           236             1
971410           2.90         1.0           162           238             1
```

```
971411           1.60         1.0           141           75             1
971412           4.00         1.0           161          202             1
971413           0.85         1.0           141          262             1

        fare_amount  ...  quarter  trip_duration  tip_per_mile  OBJECTID  \
0               7.2  ...   2023Q4       5.133333      1.785714     237.0
1              18.4  ...   2023Q4      20.066667      3.058824      48.0
2               9.3  ...   2023Q4       7.483333      2.353982     234.0
3               9.3  ...   2023Q4       5.933333      1.766667     143.0
4               9.3  ...   2023Q4       7.950000      2.844660     107.0
...             ...  ...      ...            ...           ...       ...
971409         18.4  ...   2023Q4      17.533333      1.724138     170.0
971410         26.8  ...   2023Q3      30.850000      2.120690     162.0
971411         10.0  ...   2023Q3       9.733333      2.062500     141.0
971412         22.6  ...   2023Q4      22.733333      1.380000     161.0
971413          7.9  ...   2023Q2       7.316667      3.388235     141.0

        Shape_Leng  Shape_Area                  zone LocationID    borough  \
0         0.042213    0.000096  Upper East Side South      237.0  Manhattan
1         0.043747    0.000094          Clinton East       48.0  Manhattan
2         0.036072    0.000073             Union Sq      234.0  Manhattan
3         0.054180    0.000151    Lincoln Square West      143.0  Manhattan
4         0.038041    0.000075              Gramercy      107.0  Manhattan
...            ...         ...                   ...        ...        ...
971409    0.045769    0.000074           Murray Hill      170.0  Manhattan
971410    0.035270    0.000048          Midtown East      162.0  Manhattan
971411    0.041514    0.000077       Lenox Hill West      141.0  Manhattan
971412    0.035804    0.000072        Midtown Center      161.0  Manhattan
971413    0.041514    0.000077       Lenox Hill West      141.0  Manhattan

                                              geometry
0       POLYGON ((993633.442 216961.016, 993507.232 21...
1       POLYGON ((986694.313 214463.846, 986568.184 21...
2       POLYGON ((987029.847 207022.299, 987048.27 206...
3       POLYGON ((989338.1 223572.253, 989368.225 2235...
```

3.1.11 [3 marks]

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 geopandas as gpd
import matplotlib.pyplot as plt
import pandas as pd

# Read the shapefile
shapefile_path = "/content/drive/MyDrive/EDA NYC TAXI RECORDS CASE STUDY/Datasets and Dictionary-NYC/Datasets and Dictionary/taxi_zones/taxi
zones = gpd.read_file(shapefile_path)

# Group by PULocationID and count trips
trip_counts = df.groupby("PULocationID").size().reset_index(name="num_trips")

# Sort by number of trips (descending order)
trip_counts = trip_counts.sort_values(by="num_trips", ascending=False)

# Merge trip counts with the taxi zones GeoDataFrame
zones = zones.merge(trip_counts, left_on="LocationID", right_on="PULocationID", how="left")

# Fill NaN values with 0 for zones with no recorded trips
zones["num_trips"] = zones["num_trips"].fillna(0)

# Plot top 10 busiest pickup locations
top_pickups = trip_counts.head(10)

plt.bar(top_pickups["PULocationID"].astype(str), top_pickups["num_trips"], color="royalblue", edgecolor="black")
plt.xlabel("Pickup Location ID")
plt.ylabel("Number of Trips")
plt.title("Top 10 Busiest Taxi Pickup Locations")
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```

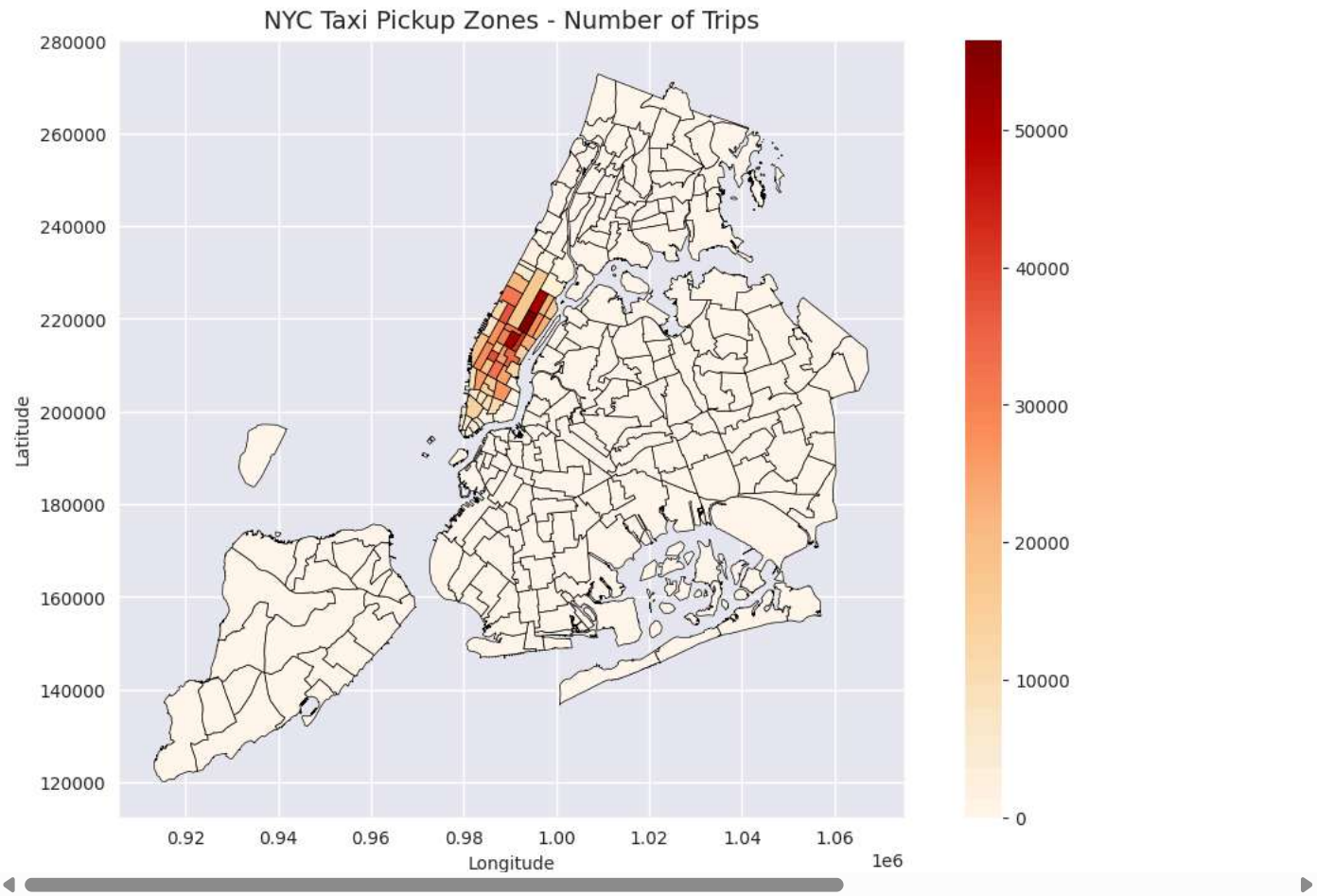


3.1.12 [2 marks]

Now, use the grouped data to add number of trips to the GeoDataFrame.

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

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

shapefile_path = "/content/drive/MyDrive/EDA NYC TAXI RECORDS CASE STUDY/Datasets and Dictionary-NYC/Datasets and Dictionary/taxi_zones/taxi
zones = gpd.read_file(shapefile_path)

# Aggregate trip counts by pickup location
trip_counts = df.groupby("PULocationID").size().reset_index(name="num_trips")

# Merge trip counts back to the zones GeoDataFrame
zones = zones.merge(trip_counts, left_on="LocationID", right_on="PULocationID", how="left")

# Fill NaN values with 0 (for zones with no recorded trips)
zones["num_trips"] = zones["num_trips"].fillna(0)

# Plot the taxi zones with trip density
fig, ax = plt.subplots(figsize=(12, 8))
zones.plot(column="num_trips", cmap="OrRd", linewidth=0.5, edgecolor="black", legend=True, ax=ax)
plt.title("NYC Taxi Pickup Zones - Number of Trips", fontsize=14)
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```



The next step is creating a color map (choropleth map) showing zones by the number of trips taken.

Again, you can use the `zones.plot()` method for this. Plot Method GPD

But first, you need to define the figure and axis for the plot.

`fig, ax = plt.subplots(1, 1, figsize = (12, 10))`

This function creates a figure (fig) and a single subplot (ax)

After setting up the figure and axis, we can proceed to plot the GeoDataFrame on this axis. This is done in the next step where we use the plot method of the GeoDataFrame.

You can define the following parameters in the `zones.plot()` method:

```
column = '',
ax = ax,
legend = True,
legend_kwds = {'label': "label", 'orientation': "<horizontal/vertical>"}
```

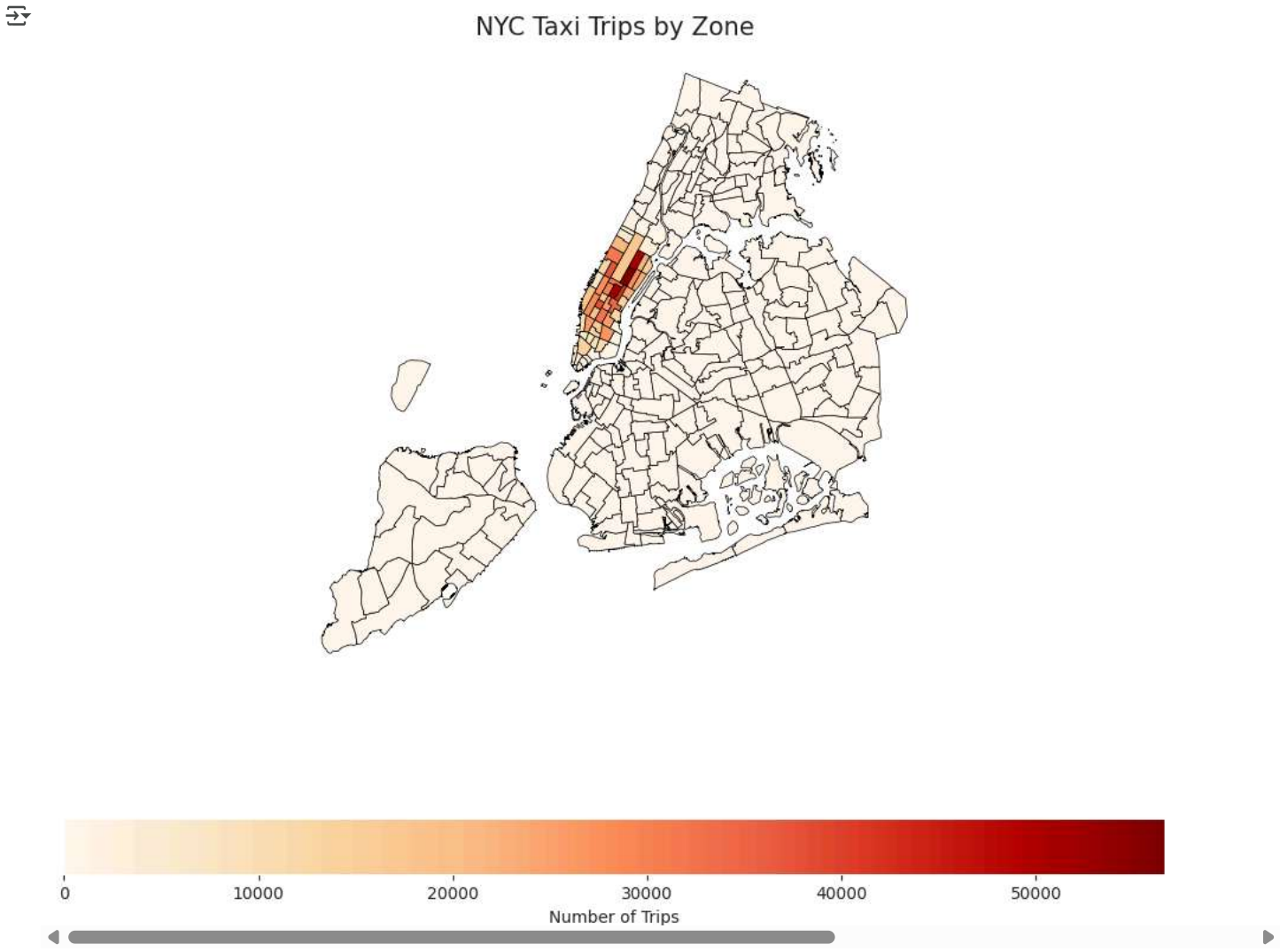
To display the plot, use `plt.show()`.

3.1.13 [3 marks]

Plot a color-coded map showing zone-wise trips

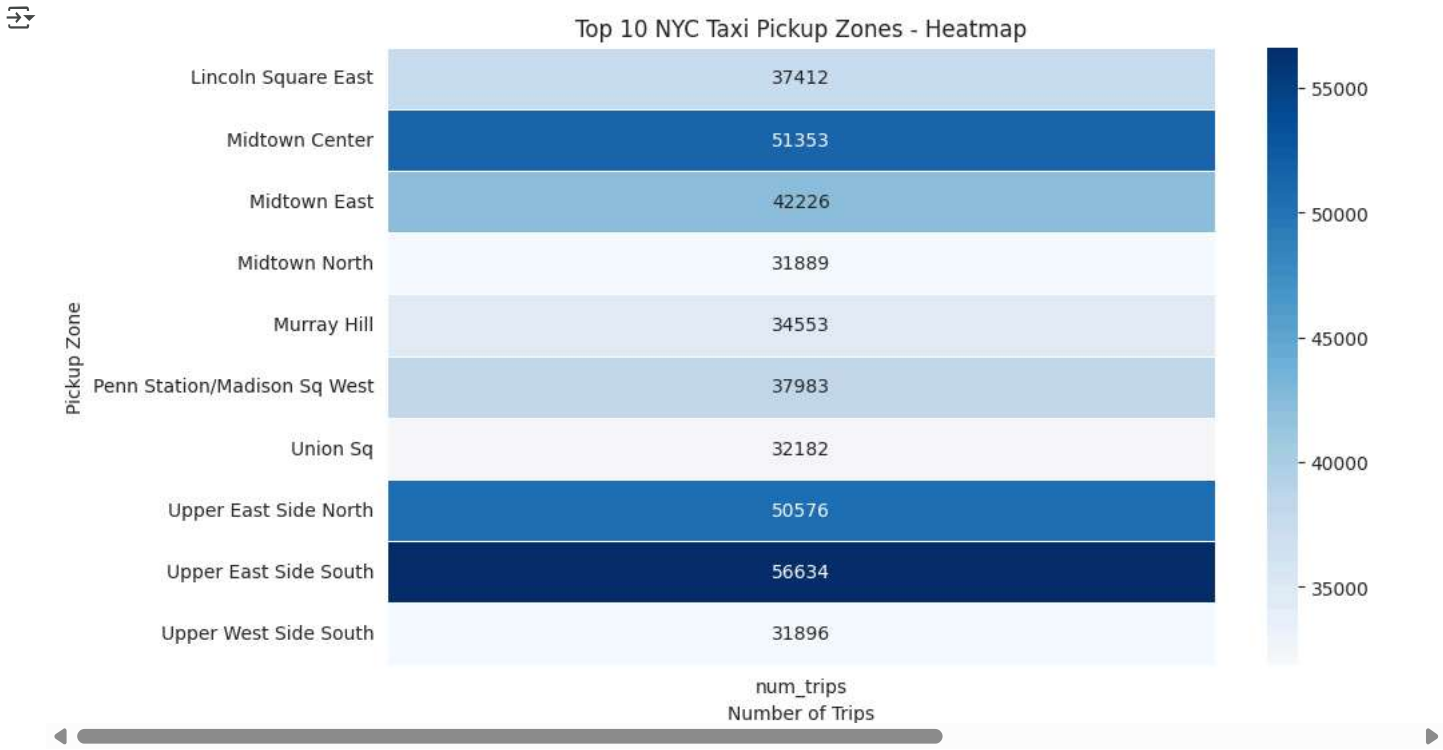
```
# Define figure and axis
fig, ax = plt.subplots(1, 1, figsize=(12, 10))

# Plot the zones with a color map based on num_trips
zones.plot(
    column="num_trips",       # Column to color by
    cmap="OrRd",              # Color map (Orange-Red)
    linewidth=0.5,            # Borderline thickness
    edgecolor="black",        # Borderline color
```

```
    ax=ax,                    # Plot on the defined axis
    legend=True,              # Show legend
    legend_kwds={
        "label": "Number of Trips",
        "orientation": "horizontal"
    }
)

# Set title
ax.set_title("NYC Taxi Trips by Zone", fontsize=15)

# Remove axes for better visualization
ax.set_xticks([])
ax.set_yticks([])
ax.set_frame_on(False)

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




```
# can you try displaying the zones DF sorted by the number of trips?
import geopandas as gpd
import matplotlib.pyplot as plt
import seaborn as sns

# Sort the zones GeoDataFrame by number of trips in descending order
zones_sorted = zones.sort_values(by="num_trips", ascending=False)

# Select the top 10 zones
top_zones = zones_sorted.head(10)

# Create a pivot table for heatmap visualization
heatmap_data = top_zones.pivot_table(index="zone", values="num_trips")

# Plot the heatmap
```

```
plt.figure(figsize=(10, 6))
sns.heatmap(
    heatmap_data,
    annot=True,
    cmap="Blues",
    linewidths=0.5,
    fmt=".0f"
)

# Labels and title
plt.xlabel("Number of Trips")
plt.ylabel("Pickup Zone")
plt.title("Top 10 NYC Taxi Pickup Zones - Heatmap")

plt.show()
```



Here we have completed the temporal, financial and geographical analysis on the trip records.

Compile your findings from general analysis below:

You can consider the following points:

- Busiest hours, days and months
- Trends in revenue collected
- Trends in quarterly revenue
- How fare depends on trip distance, trip duration and passenger counts
- How tip amount depends on trip distance
- Busiest zones

3.2 Detailed EDA: Insights and Strategies

[50 marks]

Having performed basic analyses for finding trends and patterns, we will now move on to some detailed analysis focussed on operational efficiency, pricing strategies, and customer experience.

Operational Efficiency

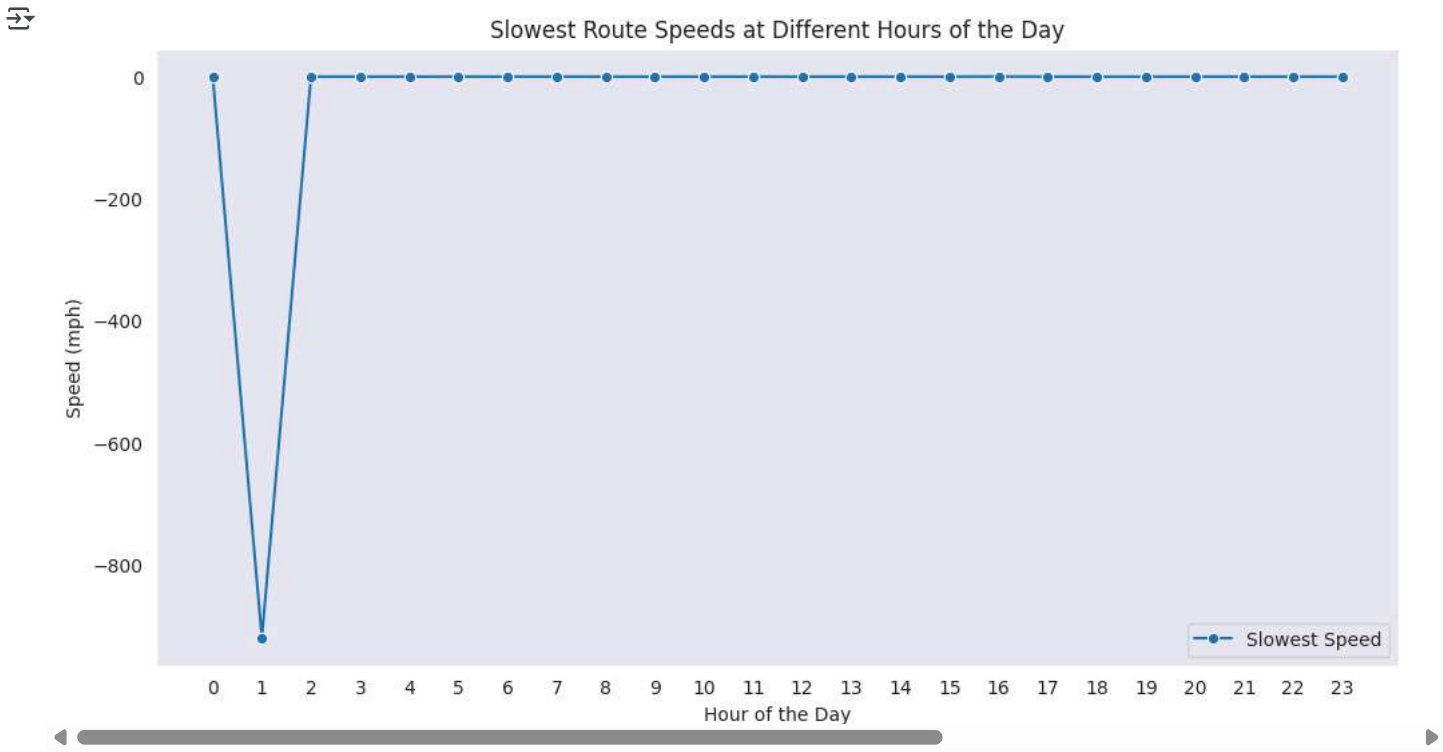
Analyze variations by time of day and location to identify bottlenecks or inefficiencies in routes

3.2.1 [3 marks]

Identify slow routes by calculating the average time taken by cabs to get from one zone to another at different hours of the day.

Speed on a route *X* for hour *Y* = (*distance of the route X / average trip duration for hour Y*)

```
# Find routes which have the slowest speeds at different times of the day

df["pickup_hour"] = pd.to_datetime(df["tpep_pickup_datetime"]).dt.hour

# Group by pickup location, dropoff location, and hour
route_stats = df.groupby(["PULocationID", "DOLocationID", "pickup_hour"]).agg(
    avg_duration=("trip_duration", "mean"),  # Average trip duration
    avg_distance=("trip_distance", "mean")   # Average trip distance
).reset_index()

# Calculate speed (distance / duration)
route_stats["speed_mph"] = route_stats["avg_distance"] / (route_stats["avg_duration"] / 60)

# Find the slowest routes per hour
slowest_routes = route_stats.sort_values(by=["pickup_hour", "speed_mph"]).groupby("pickup_hour").first().reset_index()

# Plot the slowest speeds over time
plt.figure(figsize=(12, 6))
sns.lineplot(x=slowest_routes["pickup_hour"], y=slowest_routes["speed_mph"], marker="o", label="Slowest Speed")
plt.xlabel("Hour of the Day")
plt.ylabel("Speed (mph)")
plt.title("Slowest Route Speeds at Different Hours of the Day")
plt.xticks(range(24))
plt.grid()
plt.legend()
plt.show()
```



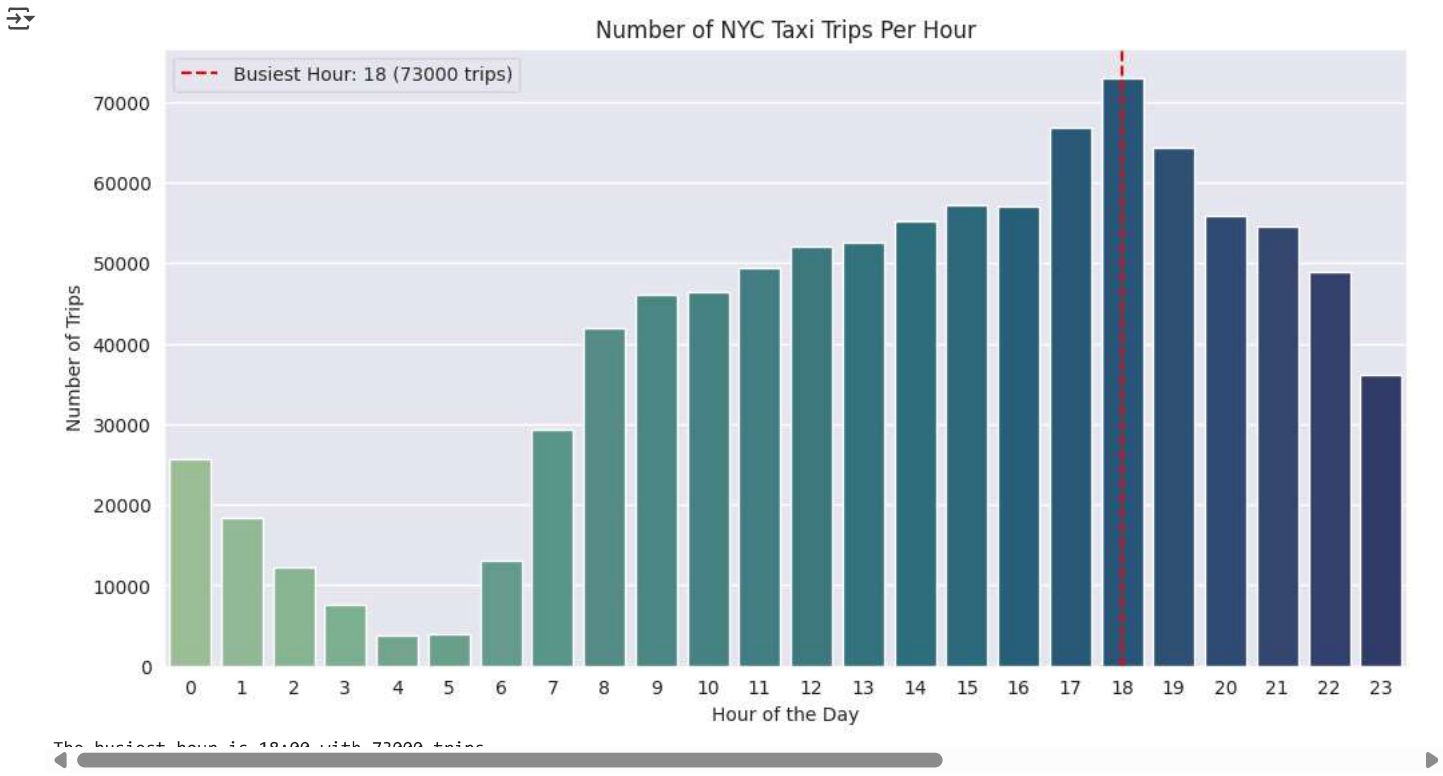
How does identifying high-traffic, high-demand routes help us?

3.2.2 [3 marks]

Calculate the number of trips at each hour of the day and visualise them. Find the busiest hour and show the number of trips for that hour.

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

# Extract hour from pickup time
df["pickup_hour"] = pd.to_datetime(df["tpep_pickup_datetime"]).dt.hour
```

```
# Count trips per hour
hourly_trips = df["pickup_hour"].value_counts().sort_index()

# Find the busiest hour
busiest_hour = hourly_trips.idxmax()
busiest_trips = hourly_trips.max()

# Plot number of trips per hour
plt.figure(figsize=(12, 6))
sns.barplot(x=hourly_trips.index, y=hourly_trips.values, palette="crest")

# Labels and title
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Trips")
plt.title("Number of NYC Taxi Trips Per Hour")
plt.xticks(range(0, 24))  # Show all 24 hours

# Highlight busiest hour
plt.axvline(x=busiest_hour, color="red", linestyle="--", label=f"Busiest Hour: {busiest_hour} ({busiest_trips} trips)")
plt.legend()

# Show plot
plt.show()

# Print busiest hour
print(f"The busiest hour is {busiest_hour}:00 with {busiest_trips} trips.")
```



Remember, we took a fraction of trips. To find the actual number, you have to scale the number up by the sampling ratio.

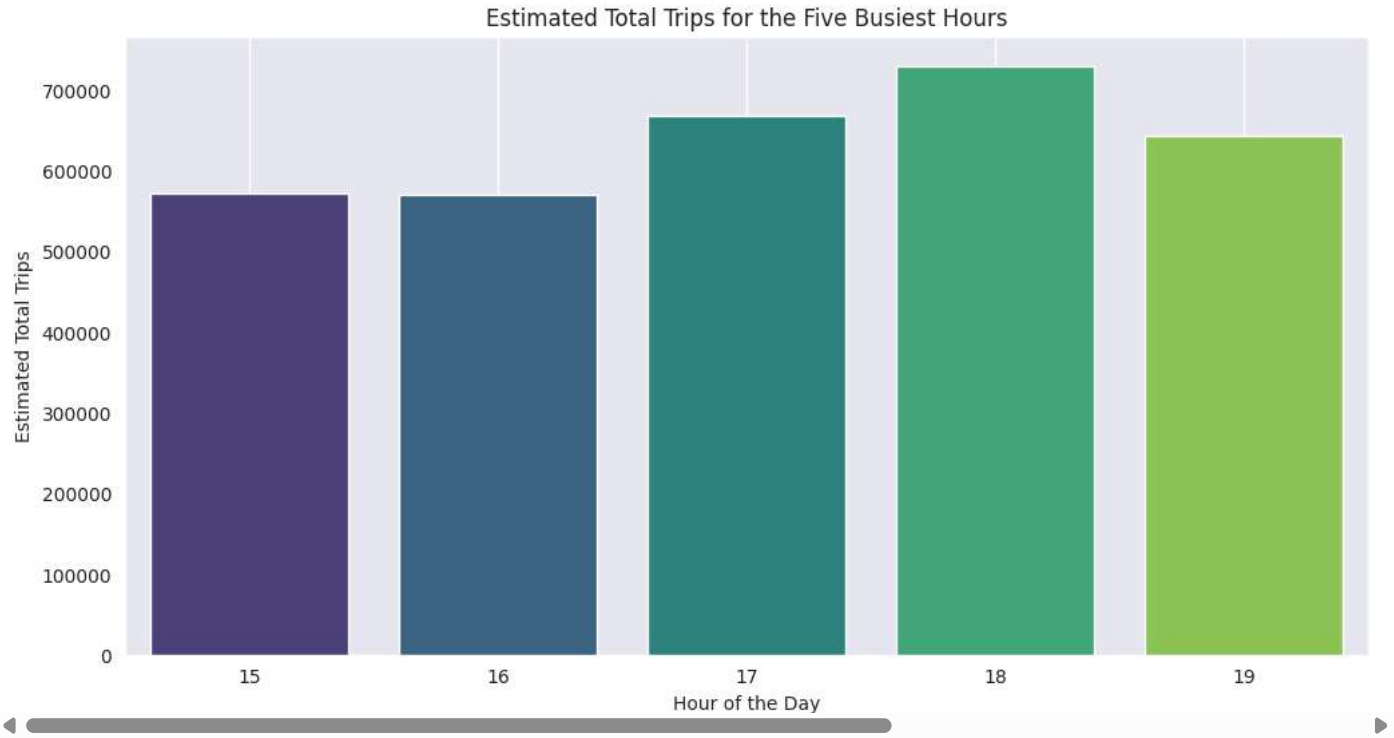
3.2.3 [2 mark]

Find the actual number of trips in the five busiest hours

```
# Scale up the number of trips
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# Assuming df is already loaded and contains necessary columns
df["pickup_hour"] = pd.to_datetime(df["tpep_pickup_datetime"]).dt.hour

# Group by pickup location, dropoff location, and hour
route_stats = df.groupby(["PULocationID", "DOLocationID", "pickup_hour"]).agg(
    avg_duration=("trip_duration", "mean"),  # Average trip duration
    avg_distance=("trip_distance", "mean")   # Average trip distance
).reset_index()
```

```
# Calculate speed (distance / duration)
route_stats["speed_mph"] = route_stats["avg_distance"] / (route_stats["avg_duration"] / 60)

# Find the slowest routes per hour
slowest_routes = route_stats.sort_values(by=["pickup_hour", "speed_mph"]).groupby("pickup_hour").first().reset_index()

# Scale up the number of trips
sample_fraction = 0.1  # Example: 10% of data was sampled
hourly_trips = df["pickup_hour"].value_counts().sort_index()
hourly_trips_scaled = (hourly_trips / sample_fraction).astype(int)

# Find the five busiest hours
top_5_busiest = hourly_trips_scaled.sort_values(ascending=False).head(5)

# Display results
print("Estimated Total Trips for the Five Busiest Hours:")
print(top_5_busiest)

# Plot busiest hours
plt.figure(figsize=(12, 6))
sns.barplot(x=top_5_busiest.index, y=top_5_busiest.values, palette="viridis")
plt.xlabel("Hour of the Day")
plt.ylabel("Estimated Total Trips")
plt.title("Estimated Total Trips for the Five Busiest Hours")
plt.grid()
plt.show()
```

```
Estimated Total Trips for the Five Busiest Hours:
pickup_hour
18    730000
17    667570
19    643620
15    571860
16    569820
Name: count, dtype: int64
```

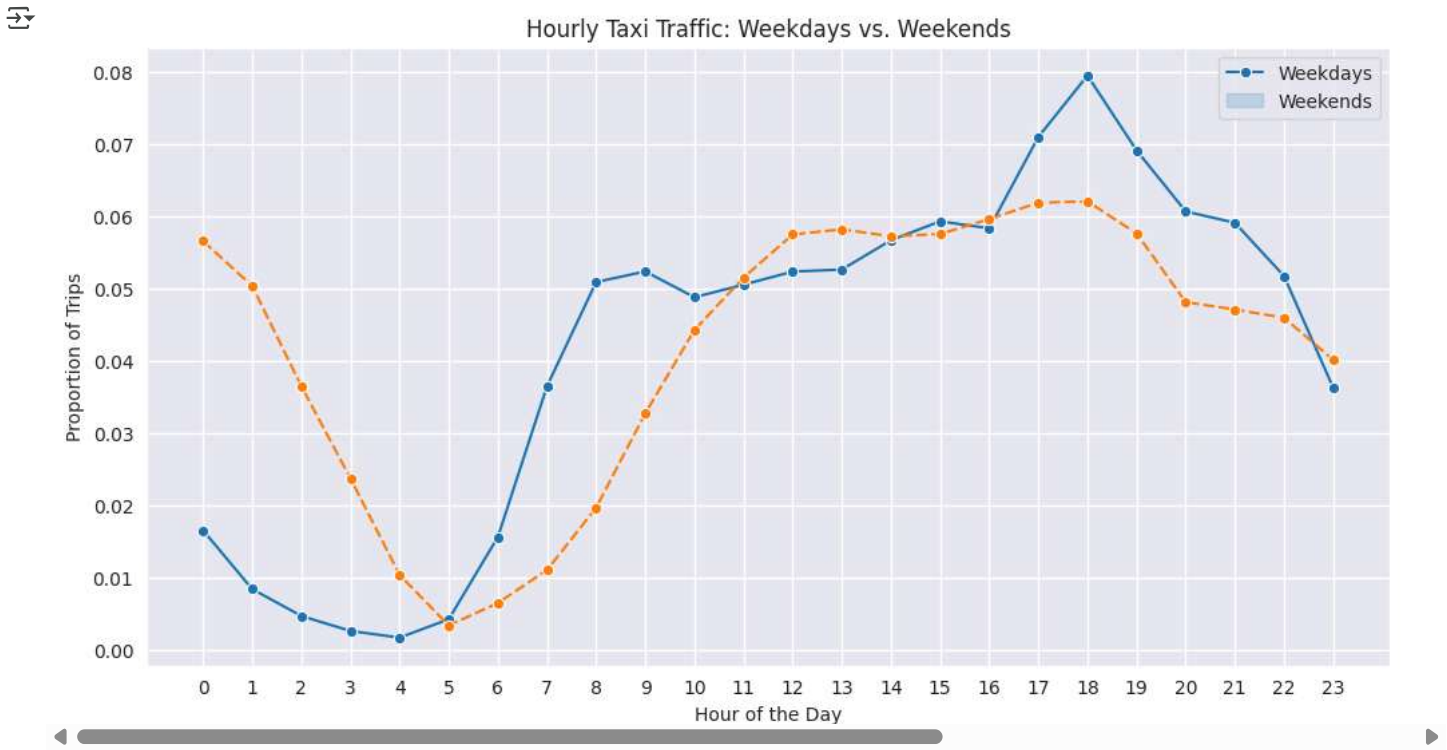


3.2.4 [3 marks]

Compare hourly traffic pattern on weekdays. Also compare for weekend.

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

# Extract hour and day of the week
df["pickup_hour"] = pd.to_datetime(df["tpep_pickup_datetime"]).dt.hour
df["day_of_week"] = pd.to_datetime(df["tpep_pickup_datetime"]).dt.weekday  # 0=Monday, 6=Sunday

# Separate into weekdays (0-4) and weekends (5-6)
```

```
df["is_weekend"] = df["day_of_week"] >= 5

# Group by hour and weekday/weekend
hourly_traffic = df.groupby(["pickup_hour", "is_weekend"]).size().reset_index(name="num_trips")

# Pivot for plotting
hourly_pivot = hourly_traffic.pivot(index="pickup_hour", columns="is_weekend", values="num_trips")

# Rename columns
hourly_pivot.columns = ["Weekdays", "Weekends"]

# Normalize to compare trends (optional)
hourly_pivot = hourly_pivot / hourly_pivot.sum()

# Plot the hourly traffic pattern
plt.figure(figsize=(12, 6))
sns.lineplot(data=hourly_pivot, marker="o")

# Labels and title
plt.xlabel("Hour of the Day")
plt.ylabel("Proportion of Trips")
plt.title("Hourly Taxi Traffic: Weekdays vs. Weekends")
plt.xticks(range(0, 24))  # Show all hours
plt.legend(["Weekdays", "Weekends"])

# Show plot
plt.show()
```

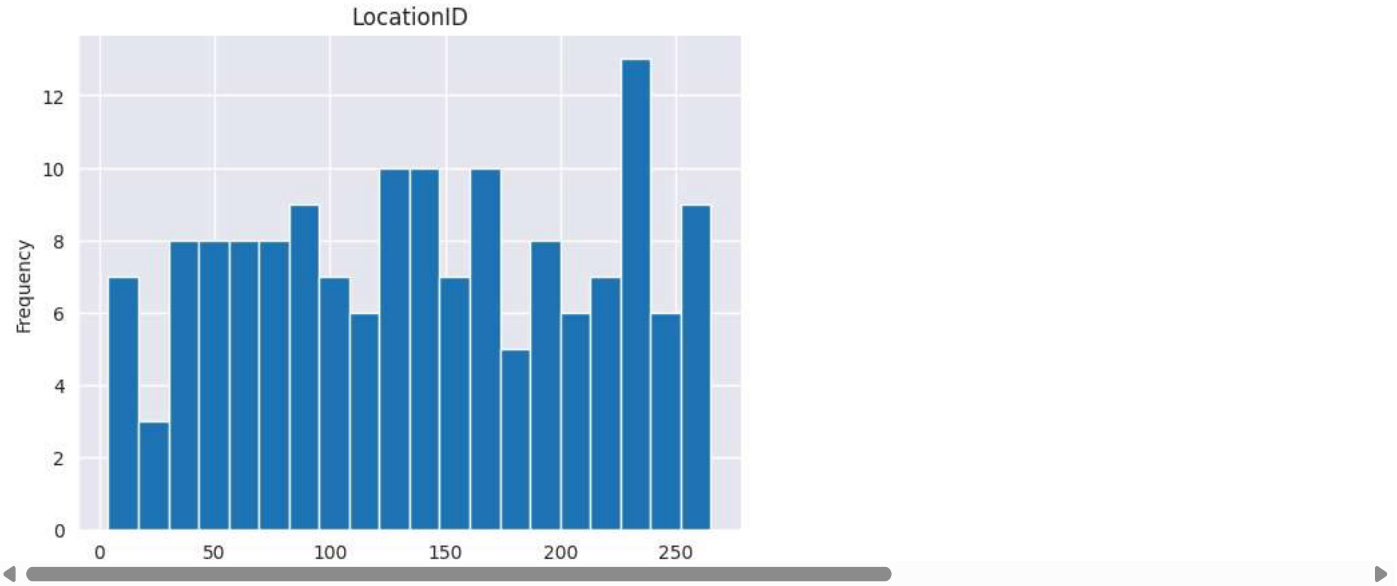


What can you infer from the above patterns? How will finding busy and quiet hours for each day help us?

```
# Count pickups and dropoffs per zone
pickup_counts = df["PULocationID"].value_counts().reset_index()
pickup_counts.columns = ["LocationID", "num_pickups"]

dropoff_counts = df["DOLocationID"].value_counts().reset_index()
dropoff_counts.columns = ["LocationID", "num_dropoffs"]

# Merge the pickup and dropoff counts
zone_counts = pickup_counts.merge(dropoff_counts, on="LocationID", how="outer").fillna(0)

# Merge with zone names
zone_counts = zone_counts.merge(zones[["LocationID", "zone"]], on="LocationID", how="left")

# Calculate the ratio of pickups to dropoffs
zone_counts["pickup_dropoff_ratio"] = zone_counts["num_pickups"] / zone_counts["num_dropoffs"]
zone_counts["pickup_dropoff_ratio"].replace([float("inf"), float("nan")], 0, inplace=True)
```

```
# Display results
zone_counts.sort_values(by="pickup_dropoff_ratio", ascending=False, inplace=True)

print(zone_counts)

zone_counts['LocationID'].plot(kind='hist', bins=20, title='LocationID')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

```
     LocationID  num_pickups  num_dropoffs                              zone  \
34           70         15.0             2                     East Elmhurst
105         186      37983.0         23600       Penn Station/Madison Sq West
60          114      14489.0         10670            Greenwich Village South
17           43      16734.0         12415                       Central Park
92          162      42226.0         32196                       Midtown East
..          ...          ...           ...                                ...
128         227          0.0             3                  Sunset Park East
136         235          0.0             4  University Heights/Morris Heights
123         218          0.0             1          Springfield Gardens North
126         225          0.0            75                 Stuyvesant Heights
148         257          0.0             5                    Windsor Terrace

     pickup_dropoff_ratio
34               7.500000
105              1.609449
60               1.357919
17               1.347886
92               1.311529
..                    ...
128              0.000000
136              0.000000
123              0.000000
126              0.000000
148              0.000000

[155 rows x 5 columns]
```

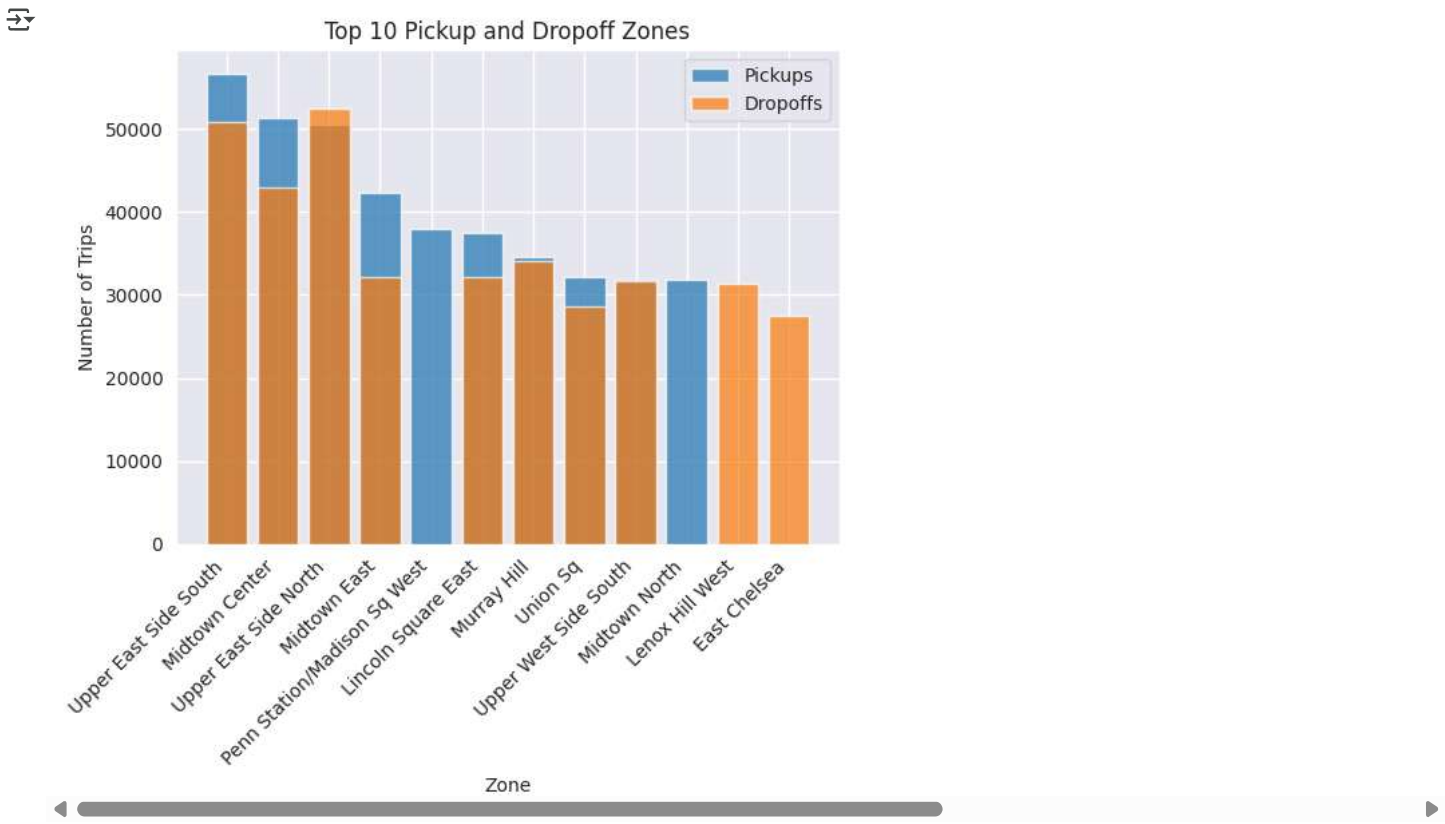


3.2.5 [3 marks]

Identify top 10 zones with high hourly pickups. Do the same for hourly dropoffs. Show pickup and dropoff trends in these zones.

```
# Find top 10 pickup and dropoff zones

import matplotlib.pyplot as plt

# Find top 10 pickup and dropoff zones
top_pickups = df["PULocationID"].value_counts().head(10).reset_index()
top_pickups.columns = ["LocationID", "num_pickups"]

top_dropoffs = df["DOLocationID"].value_counts().head(10).reset_index()
top_dropoffs.columns = ["LocationID", "num_dropoffs"]

# Merge with zones dataset to get zone names
top_pickups = top_pickups.merge(zones[["LocationID", "zone"]], on="LocationID", how="left")
top_dropoffs = top_dropoffs.merge(zones[["LocationID", "zone"]], on="LocationID", how="left")
```

```
# Plot pickups
plt.bar(top_pickups["zone"], top_pickups["num_pickups"], label="Pickups", alpha=0.7)

# Plot dropoffs
plt.bar(top_dropoffs["zone"], top_dropoffs["num_dropoffs"], label="Dropoffs", alpha=0.7)

plt.xlabel("Zone")
plt.ylabel("Number of Trips")
plt.title("Top 10 Pickup and Dropoff Zones")
plt.xticks(rotation=45, ha="right")
plt.legend()
plt.show()
```



3.2.6 [3 marks]

Find the ratio of pickups and dropoffs in each zone. Display the 10 highest (pickup/drop) and 10 lowest (pickup/drop) ratios.

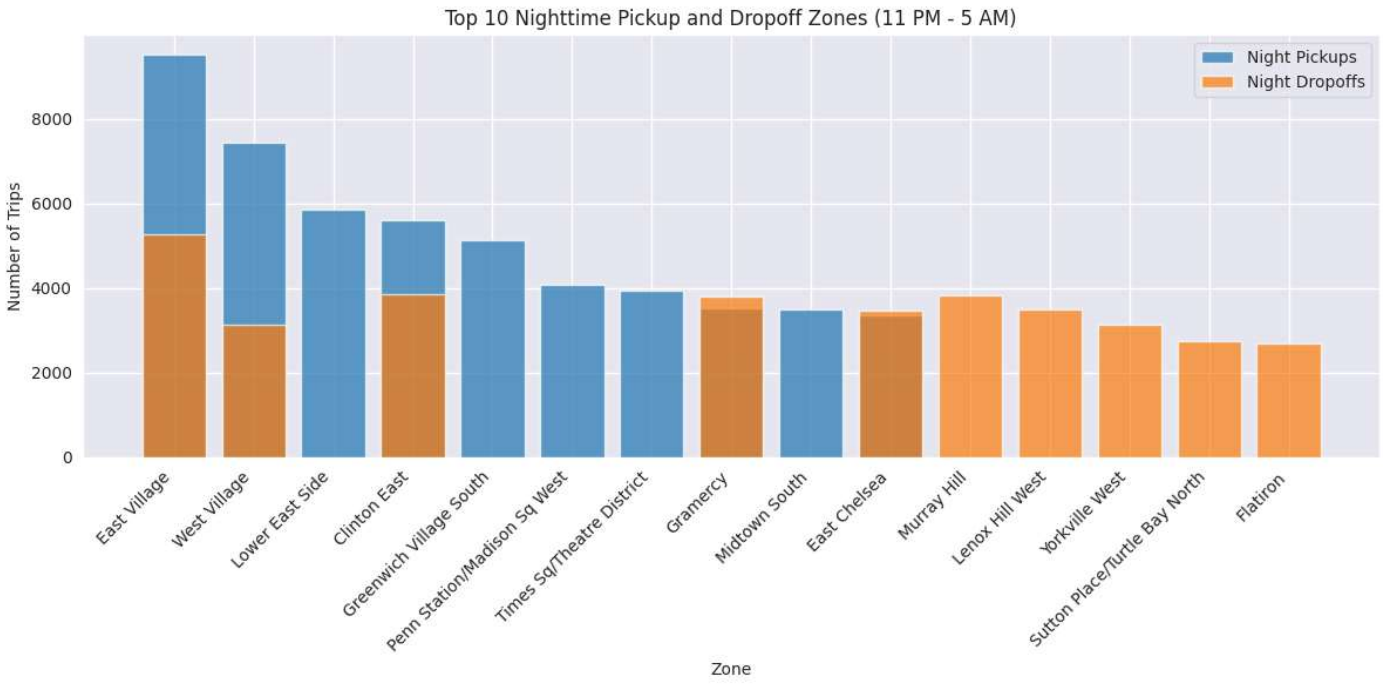
```
# Find the top 10 and bottom 10 pickup/dropoff ratios

# Count trips for pickup and dropoff
pickup_counts = df["PULocationID"].value_counts().rename("num_pickups")
dropoff_counts = df["DOLocationID"].value_counts().rename("num_dropoffs")

# Merge pickup and dropoff counts into a single DataFrame
pickup_dropoff_df = pd.DataFrame({"num_pickups": pickup_counts, "num_dropoffs": dropoff_counts}).fillna(1)

# Calculate the pickup/dropoff ratio
pickup_dropoff_df["pickup_dropoff_ratio"] = pickup_dropoff_df["num_pickups"] / pickup_dropoff_df["num_dropoffs"]

# Merge with taxi zones data to get names
pickup_dropoff_df = pickup_dropoff_df.reset_index().rename(columns={"index": "LocationID"})
pickup_dropoff_df = pickup_dropoff_df.merge(zones[["LocationID", "zone"]], on="LocationID", how="left")

# Get top 10 and bottom 10 ratios
top_10_ratios = pickup_dropoff_df.sort_values(by="pickup_dropoff_ratio", ascending=False).head(10)
bottom_10_ratios = pickup_dropoff_df.sort_values(by="pickup_dropoff_ratio", ascending=True).head(10)

# Display results
print("Top 10 Pickup/Dropoff Ratios:")
print(top_10_ratios[["zone", "pickup_dropoff_ratio"]])

print("\nBottom 10 Pickup/Dropoff Ratios:")
print(bottom_10_ratios[["zone", "pickup_dropoff_ratio"]])
```

```
Top 10 Pickup/Dropoff Ratios:
                            zone  pickup_dropoff_ratio
34                 East Elmhurst              7.500000
105  Penn Station/Madison Sq West              1.609449
60        Greenwich Village South              1.357919
17                  Central Park              1.347886
92                  Midtown East              1.311529
145                 West Village              1.306322
52               Garment District              1.290942
93                  Midtown North              1.228011
131     Times Sq/Theatre District              1.202219
91                  Midtown Center              1.195479

Bottom 10 Pickup/Dropoff Ratios:
                            zone  pickup_dropoff_ratio
28             Crown Heights North              0.009709
97           Mott Haven/Port Morris             0.009709
126             Stuyvesant Heights              0.013333
114               Roosevelt Island              0.014760
122             South Williamsburg              0.015152
23                 Columbia Street              0.015385
20                    Clinton Hill              0.019048
13                  Bushwick South              0.019737
106       Prospect-Lefferts Gardens              0.020833
81   Long Island City/Hunters Point              0.026820
```

3.2.7 [3 marks]

Identify zones with high pickup and dropoff traffic during night hours (11PM to 5AM)

```
# 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

# Extract hour from pickup datetime
df["pickup_hour"] = pd.to_datetime(df["tpep_pickup_datetime"]).dt.hour

# Filter for night hours (11 PM to 5 AM)
night_df = df[(df["pickup_hour"] >= 23) | (df["pickup_hour"] <= 5)]

# Count pickups and dropoffs during night hours
top_night_pickups = night_df["PULocationID"].value_counts().head(10).reset_index()
top_night_pickups.columns = ["LocationID", "num_pickups"]

top_night_dropoffs = night_df["DOLocationID"].value_counts().head(10).reset_index()
top_night_dropoffs.columns = ["LocationID", "num_dropoffs"]

# Merge with taxi zones dataset to get zone names
top_night_pickups = top_night_pickups.merge(zones[["LocationID", "zone"]], on="LocationID", how="left")
top_night_dropoffs = top_night_dropoffs.merge(zones[["LocationID", "zone"]], on="LocationID", how="left")

# Plot bar chart for night pickups and dropoffs
plt.figure(figsize=(12, 6))

# Plot pickups
plt.bar(top_night_pickups["zone"], top_night_pickups["num_pickups"], label="Night Pickups", alpha=0.7)

# Plot dropoffs
plt.bar(top_night_dropoffs["zone"], top_night_dropoffs["num_dropoffs"], label="Night Dropoffs", alpha=0.7)

plt.xlabel("Zone")
plt.ylabel("Number of Trips")
plt.title("Top 10 Nighttime Pickup and Dropoff Zones (11 PM - 5 AM)")
plt.xticks(rotation=45, ha="right")
plt.legend()
plt.tight_layout()
plt.show()
```



Now, let us find the revenue share for the night time hours and the day time hours. After this, we will move to deciding a pricing strategy.

3.2.8 [2 marks]
Find the revenue share for nighttime and daytime hours.

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

# Extract the hour from the pickup timestamp
df["pickup_hour"] = pd.to_datetime(df["tpep_pickup_datetime"]).dt.hour

# Define nighttime and daytime filters
night_df = df[(df["pickup_hour"] >= 23) | (df["pickup_hour"] <= 5)]
day_df = df[(df["pickup_hour"] > 5) & (df["pickup_hour"] < 23)]

# Calculate total revenue for each period
night_revenue = night_df["total_amount"].sum()
day_revenue = day_df["total_amount"].sum()
total_revenue = night_revenue + day_revenue

# Compute revenue share percentage
night_share = (night_revenue / total_revenue) * 100
day_share = (day_revenue / total_revenue) * 100

print(f"Night Time: {night_share:.2f}%")
print(f"Day Time: {day_share:.2f}%")
```

```
Night Time: 11.19%
Day Time: 88.81%
```

Pricing Strategy

3.2.9 [2 marks]
For the different passenger counts, find the average fare per mile per passenger.

For instance, suppose the average fare per mile for trips with 3 passengers is 3 USD/mile, then the fare per mile per passenger will be 1 USD/mile.

```
# Analyse the fare per mile per passenger for different passenger counts


# Ensure trip_distance > 0 to avoid division by zero
df = df[df["trip_distance"] > 0]
```

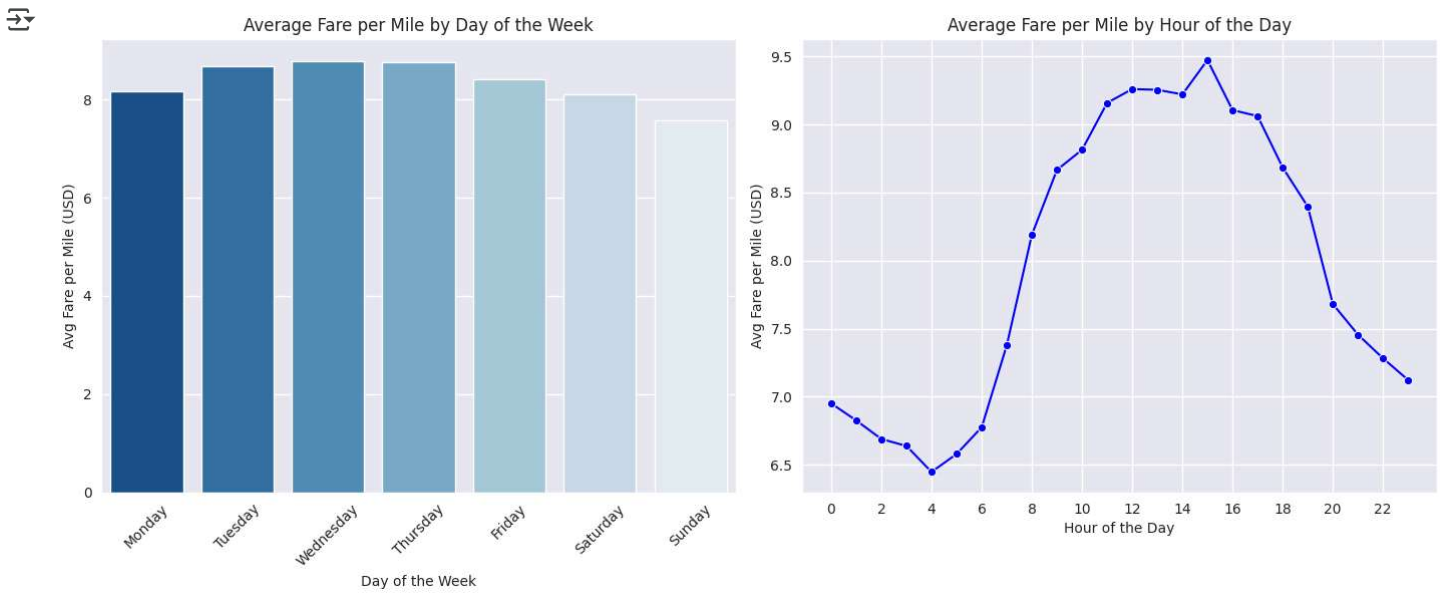
```
# Calculate fare per mile
df["fare_per_mile"] = df["fare_amount"] / df["trip_distance"]

# Ensure passenger_count > 0 to avoid division by zero
df = df[df["passenger_count"] > 0]

# Calculate fare per mile per passenger
df["fare_per_mile_per_passenger"] = df["fare_per_mile"] / df["passenger_count"]

# Compute the average fare per mile per passenger for different passenger counts
fare_analysis = df.groupby("passenger_count")["fare_per_mile_per_passenger"].mean().reset_index()

print(fare_analysis)
```

```
   passenger_count  fare_per_mile_per_passenger
0              1.0                     8.402253
```

3.2.10 [3 marks]

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

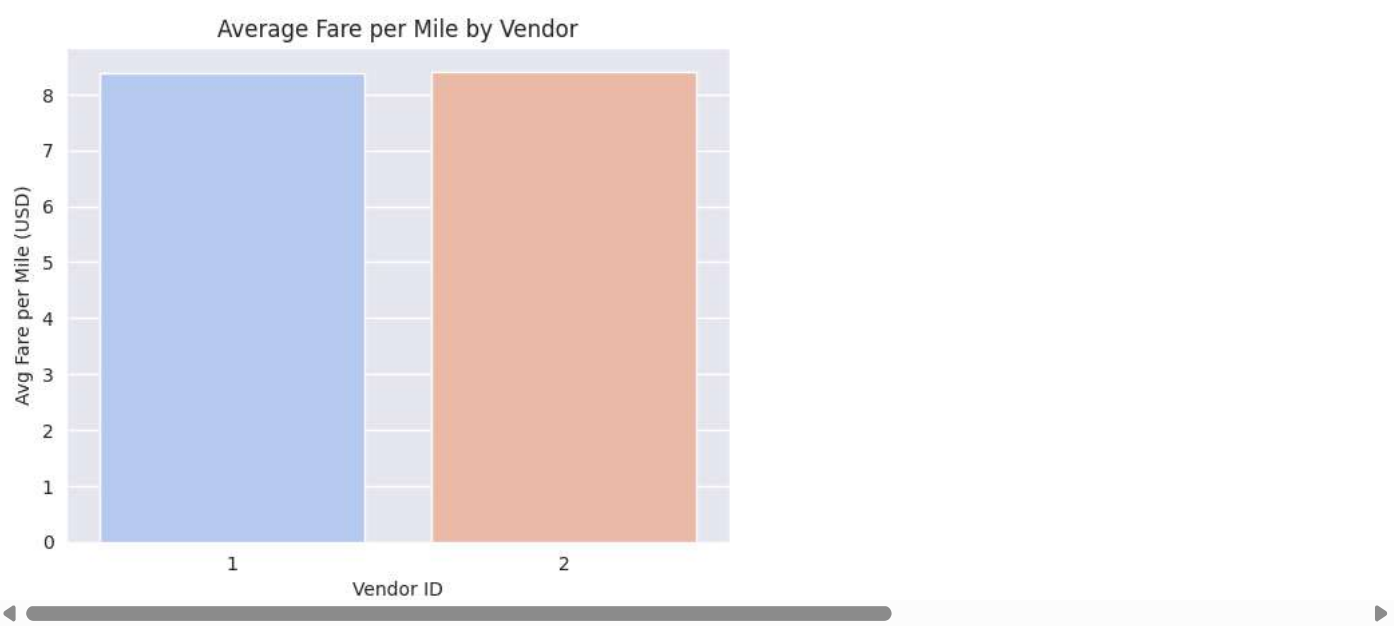
```
# Compare the average fare per mile for different days and for different times of the day

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

# Ensure trip_distance > 0 to avoid division by zero
df = df[df["trip_distance"] > 0]

# Calculate fare per mile
df["fare_per_mile"] = df["fare_amount"] / df["trip_distance"]

# Extract day of the week and hour
df["pickup_day"] = pd.to_datetime(df["tpep_pickup_datetime"]).dt.day_name()  # Get full weekday name
df["pickup_hour"] = pd.to_datetime(df["tpep_pickup_datetime"]).dt.hour  # Extract hour (0-23)

# Compute average fare per mile for each day of the week
fare_by_day = df.groupby("pickup_day")["fare_per_mile"].mean().reindex(
    ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
)

# Compute average fare per mile for each hour of the day
fare_by_hour = df.groupby("pickup_hour")["fare_per_mile"].mean()

# --- Plotting ---
plt.figure(figsize=(14, 6))

# Plot for days of the week
plt.subplot(1, 2, 1)
sns.barplot(x=fare_by_day.index, y=fare_by_day.values, palette="Blues_r")
plt.xlabel("Day of the Week")
plt.ylabel("Avg Fare per Mile (USD)")
plt.title("Average Fare per Mile by Day of the Week")
plt.xticks(rotation=45)

# Plot for hourly trend
plt.subplot(1, 2, 2)
sns.lineplot(x=fare_by_hour.index, y=fare_by_hour.values, marker="o", color="blue")
plt.xlabel("Hour of the Day")
plt.ylabel("Avg Fare per Mile (USD)")
plt.title("Average Fare per Mile by Hour of the Day")
plt.xticks(range(0, 24, 2))  # Show labels every 2 hours
plt.tight_layout()
plt.show()
```



3.2.11 [3 marks]

Analyse the average fare per mile for the different vendors for different hours of the day

```
# Compare fare per mile for different vendors

# Ensure trip_distance > 0 to avoid division by zero
df = df[df["trip_distance"] > 0]

# Calculate fare per mile
df["fare_per_mile"] = df["fare_amount"] / df["trip_distance"]

# Compute average fare per mile for each vendor
fare_by_vendor = df.groupby("VendorID", as_index=False)["fare_per_mile"].mean()

# Convert VendorID to a string for better categorical plotting
fare_by_vendor["VendorID"] = fare_by_vendor["VendorID"].astype(str)

# Plot bar chart
sns.barplot(x="VendorID", y="fare_per_mile", data=fare_by_vendor, palette="coolwarm")
plt.xlabel("Vendor ID")
plt.ylabel("Avg Fare per Mile (USD)")
plt.title("Average Fare per Mile by Vendor")
plt.show()
```



3.2.12 [5 marks]

Compare the fare rates of the different vendors in a tiered fashion. Analyse the average fare per mile for distances upto 2 miles. Analyse the fare per mile for distances from 2 to 5 miles. And then for distances more than 5 miles.

```
# Defining distance tiers

# Define distance tier boundaries
bins = [0, 2, 5, 10, 20, float("inf")]  # Adjust as needed
labels = ["Very Short (0-2 mi)", "Short (2-5 mi)", "Medium (5-10 mi)", "Long (10-20 mi)", "Very Long (20+ mi)"]

# Categorize trips into distance tiers
df["distance_tier"] = pd.cut(df["trip_distance"], bins=bins, labels=labels, right=False)

# Display distribution of trips per tier
distance_tier_counts = df["distance_tier"].value_counts().sort_index()
print(distance_tier_counts)
```

```
distance_tier
Very Short (0-2 mi)    644771
Short (2-5 mi)         308698
Medium (5-10 mi)        15166
Long (10-20 mi)             0
Very Long (20+ mi)          0
Name: count, dtype: int64
```

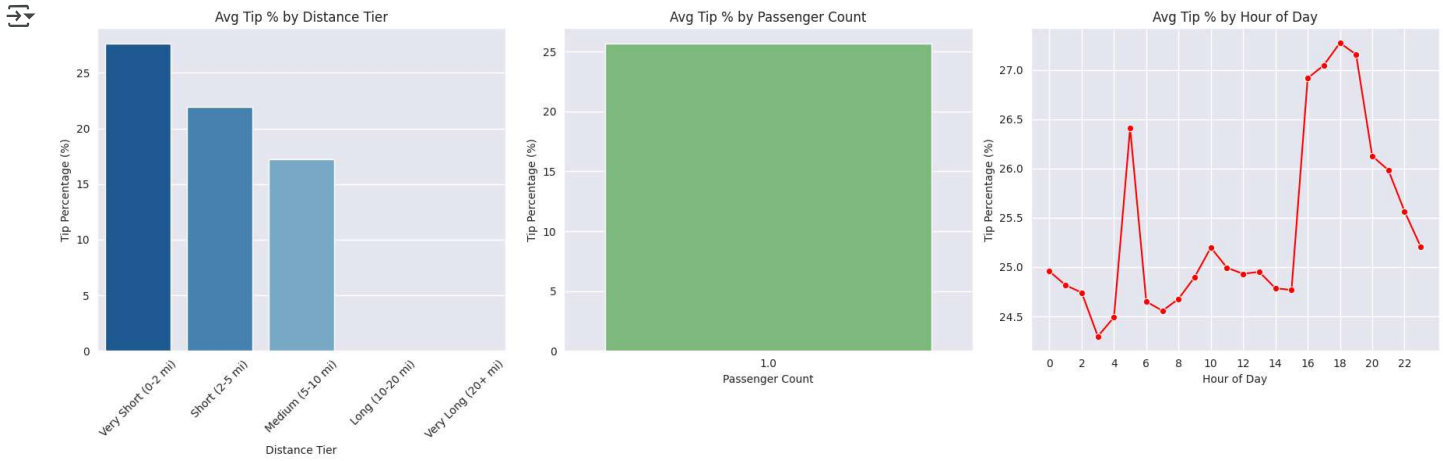
Customer Experience and Other Factors

3.2.13 [5 marks]

Analyse average tip percentages based on trip distances, passenger counts and time of pickup. What factors lead to low tip percentages?

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

# Ensure fare_amount > 0 to avoid division by zero
df = df[df["fare_amount"] > 0]

# Calculate tip percentage
df["tip_percent"] = (df["tip_amount"] / df["fare_amount"]) * 100

# --- Tip % by Distance Tier ---
tip_by_distance = df.groupby("distance_tier")["tip_percent"].mean().reset_index()

# --- Tip % by Passenger Count ---
tip_by_passengers = df.groupby("passenger_count")["tip_percent"].mean().reset_index()

# --- Tip % by Hour of the Day ---
df["pickup_hour"] = pd.to_datetime(df["tpep_pickup_datetime"]).dt.hour
tip_by_hour = df.groupby("pickup_hour")["tip_percent"].mean().reset_index()
```

```
# --- Create Subplots ---
fig, axes = plt.subplots(1, 3, figsize=(18, 6))

# Plot 1: Tip % by Distance Tier
sns.barplot(x="distance_tier", y="tip_percent", data=tip_by_distance, ax=axes[0], palette="Blues_r")
axes[0].set_title("Avg Tip % by Distance Tier")
axes[0].set_xlabel("Distance Tier")
axes[0].set_ylabel("Tip Percentage (%)")
axes[0].tick_params(axis='x', rotation=45)

# Plot 2: Tip % by Passenger Count
sns.barplot(x="passenger_count", y="tip_percent", data=tip_by_passengers, ax=axes[1], palette="Greens_r")
axes[1].set_title("Avg Tip % by Passenger Count")
axes[1].set_xlabel("Passenger Count")
axes[1].set_ylabel("Tip Percentage (%)")

# Plot 3: Tip % by Hour of the Day
sns.lineplot(x="pickup_hour", y="tip_percent", data=tip_by_hour, marker="o", ax=axes[2], color="red")
axes[2].set_title("Avg Tip % by Hour of Day")
axes[2].set_xlabel("Hour of Day")
axes[2].set_ylabel("Tip Percentage (%)")
axes[2].set_xticks(range(0, 24, 2))  # Show labels every 2 hours

# Adjust layout and show plot
plt.tight_layout()
plt.show()
```



Additional analysis [optional]: Let's try comparing cases of low tips with cases of high tips to find out if we find a clear aspect that drives up the tipping behaviours

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

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

# Ensure valid fare amounts
df = df[df["fare_amount"] > 0]

# Calculate tip percentage
df["tip_percent"] = (df["tip_amount"] / df["fare_amount"]) * 100

# Define categories
low_tip_df = df[df["tip_percent"] < 10]
high_tip_df = df[df["tip_percent"] > 25]

# --- Compare trip distances ---
fig, axes = plt.subplots(2, 3, figsize=(18, 10))

sns.histplot(low_tip_df["trip_distance"], bins=30, kde=True, ax=axes[0, 0], color="blue")
sns.histplot(high_tip_df["trip_distance"], bins=30, kde=True, ax=axes[1, 0], color="green")
axes[0, 0].set_title("Trip Distance (Low Tip < 10%)")
axes[1, 0].set_title("Trip Distance (High Tip > 25%)")
```

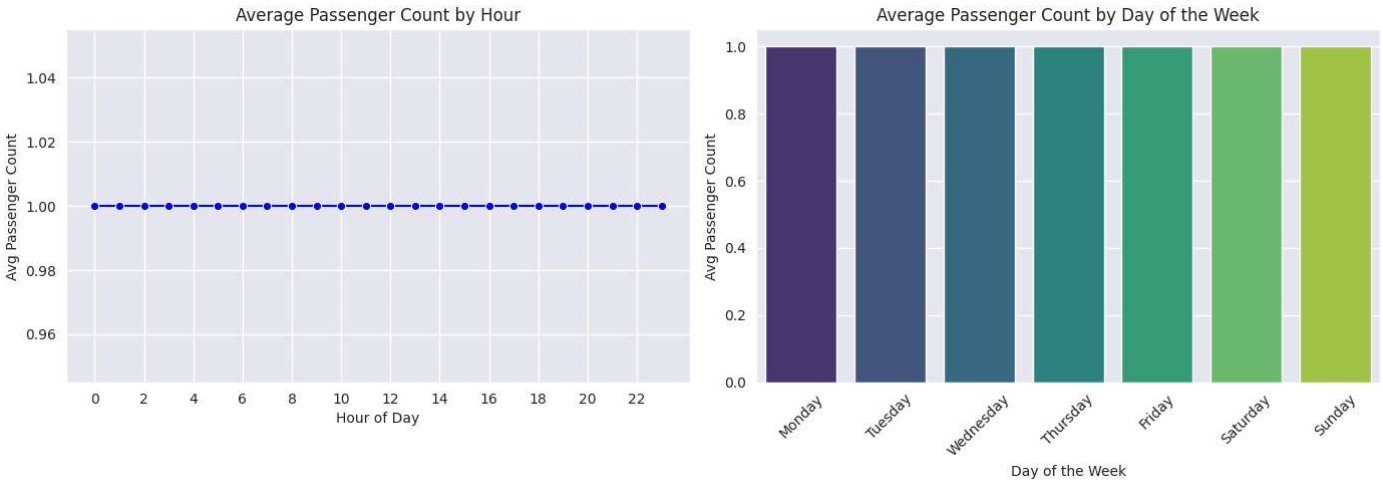
```
# --- Compare fare amounts ---
sns.histplot(low_tip_df["fare_amount"], bins=30, kde=True, ax=axes[0, 1], color="blue")
sns.histplot(high_tip_df["fare_amount"], bins=30, kde=True, ax=axes[1, 1], color="green")
axes[0, 1].set_title("Fare Amount (Low Tip < 10%)")
axes[1, 1].set_title("Fare Amount (High Tip > 25%)")

# --- Compare passenger count ---
sns.countplot(x="passenger_count", data=low_tip_df, ax=axes[0, 2], palette="Blues_r")
sns.countplot(x="passenger_count", data=high_tip_df, ax=axes[1, 2], palette="Greens_r")
axes[0, 2].set_title("Passenger Count (Low Tip < 10%)")
axes[1, 2].set_title("Passenger Count (High Tip > 25%)")

# Adjust layout and show plots
plt.tight_layout()
plt.show()
```

Show hidden output

3.2.14 [3 marks]

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

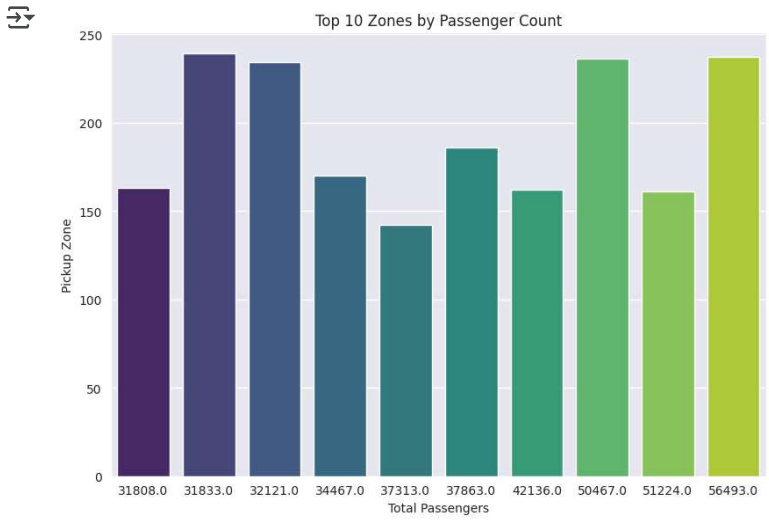
```
# See how passenger count varies across hours and days

# Extract hour and day
df["pickup_datetime"] = pd.to_datetime(df["tpep_pickup_datetime"])
df["pickup_hour"] = df["pickup_datetime"].dt.hour
df["pickup_day"] = df["pickup_datetime"].dt.day_name()

# --- Aggregate passenger counts ---
hourly_passengers = df.groupby("pickup_hour")["passenger_count"].mean().reset_index()
daily_passengers = df.groupby("pickup_day")["passenger_count"].mean().reset_index()

# Order days of the week correctly
days_order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
daily_passengers["pickup_day"] = pd.Categorical(daily_passengers["pickup_day"], categories=days_order, ordered=True)
daily_passengers = daily_passengers.sort_values("pickup_day")

# --- Create Subplots ---
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Line plot for hourly variation
sns.lineplot(x="pickup_hour", y="passenger_count", data=hourly_passengers, marker="o", ax=axes[0], color="blue")
axes[0].set_title("Average Passenger Count by Hour")
axes[0].set_xlabel("Hour of Day")
axes[0].set_ylabel("Avg Passenger Count")
axes[0].set_xticks(range(0, 24, 2))  # Show every 2 hours

# Bar plot for daily variation
sns.barplot(x="pickup_day", y="passenger_count", data=daily_passengers, ax=axes[1], palette="viridis")
axes[1].set_title("Average Passenger Count by Day of the Week")
axes[1].set_xlabel("Day of the Week")
axes[1].set_ylabel("Avg Passenger Count")
axes[1].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```

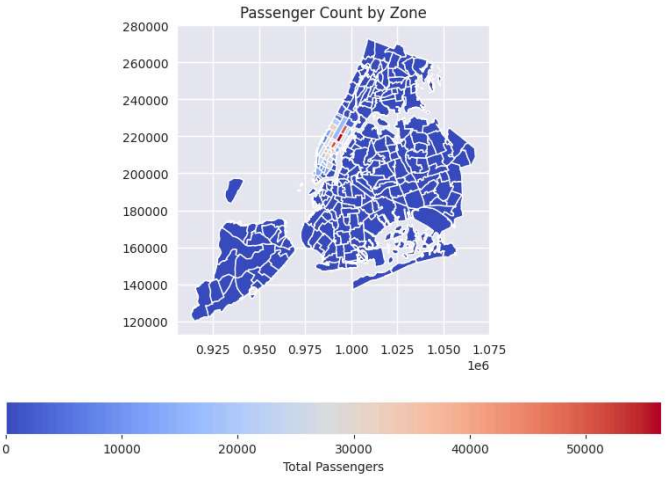


3.2.15 [2 marks]

Analyse the variation of passenger counts across zones

```
# How does passenger count vary across zones

# Aggregate passenger counts per zone
zone_passenger_counts = df.groupby("PULocationID")["passenger_count"].sum().reset_index()

# Merge with geo data (taxi zones shapefile)
zones = gpd.read_file("/content/drive/MyDrive/EDA NYC TAXI RECORDS CASE STUDY/Datasets and Dictionary-NYC/Datasets and Dictionary/taxi_zones
zones = zones.merge(zone_passenger_counts, left_on="LocationID", right_on="PULocationID", how="left")
zones["passenger_count"].fillna(0, inplace=True)  # Fill missing values

# --- Create Subplots ---
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# 1. Top 10 zones bar plot
top_zones = zone_passenger_counts.nlargest(10, "passenger_count")
sns.barplot(x="passenger_count", y="PULocationID", data=top_zones, ax=axes[0], palette="viridis")
axes[0].set_title("Top 10 Zones by Passenger Count")
axes[0].set_xlabel("Total Passengers")
axes[0].set_ylabel("Pickup Zone")

# 2. Choropleth Map of Passenger Density
zones.plot(column="passenger_count", cmap="coolwarm", legend=True, ax=axes[1],
           legend_kwds={"label": "Total Passengers", "orientation": "horizontal"})
axes[1].set_title("Passenger Count by Zone")

plt.tight_layout()
plt.show()
```






```
# For a more detailed analysis, we can use the zones GeoDataFrame
# Create a new column for the average passenger count in each zone.

# Ensure PULocationID is an integer
df["PULocationID"] = df["PULocationID"].astype(int)

# Rename the correct passenger count column
zones.rename(columns={"passenger_count": "avg_passenger_count"}, inplace=True)

# Drop unnecessary duplicate columns
zones.drop(columns=["PULocationID_y"], errors="ignore", inplace=True)

# Fill missing values with 0 (for zones with no trips)
zones["avg_passenger_count"] = zones["avg_passenger_count"].fillna(0)

# Display cleaned dataset
print(zones[["LocationID", "avg_passenger_count"]].head())
```

```
   LocationID  avg_passenger_count
0           1                  0.0
1           2                  0.0
2           3                  0.0
3           4               1073.0
4           5                  0.0
```

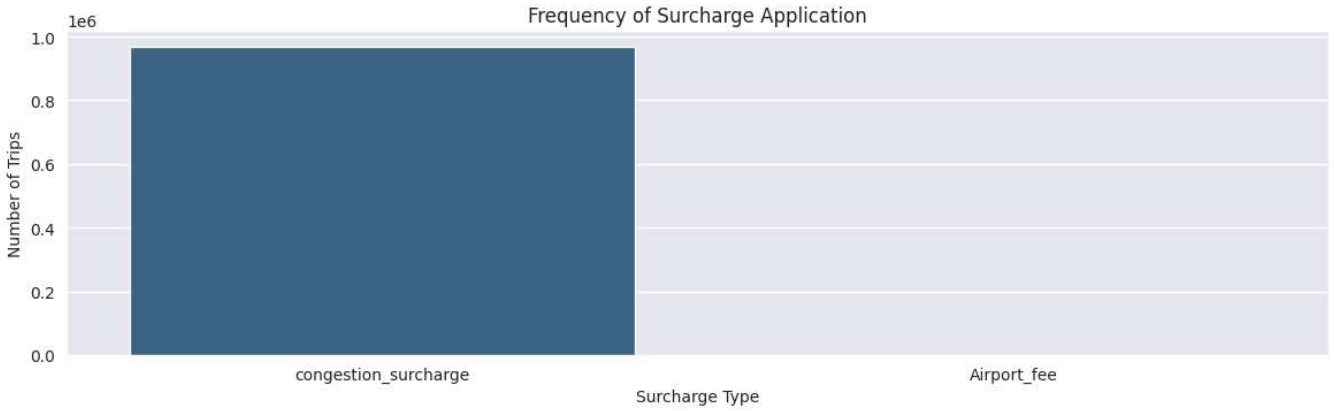
Find out how often surcharges/extra charges are applied to understand their prevalance

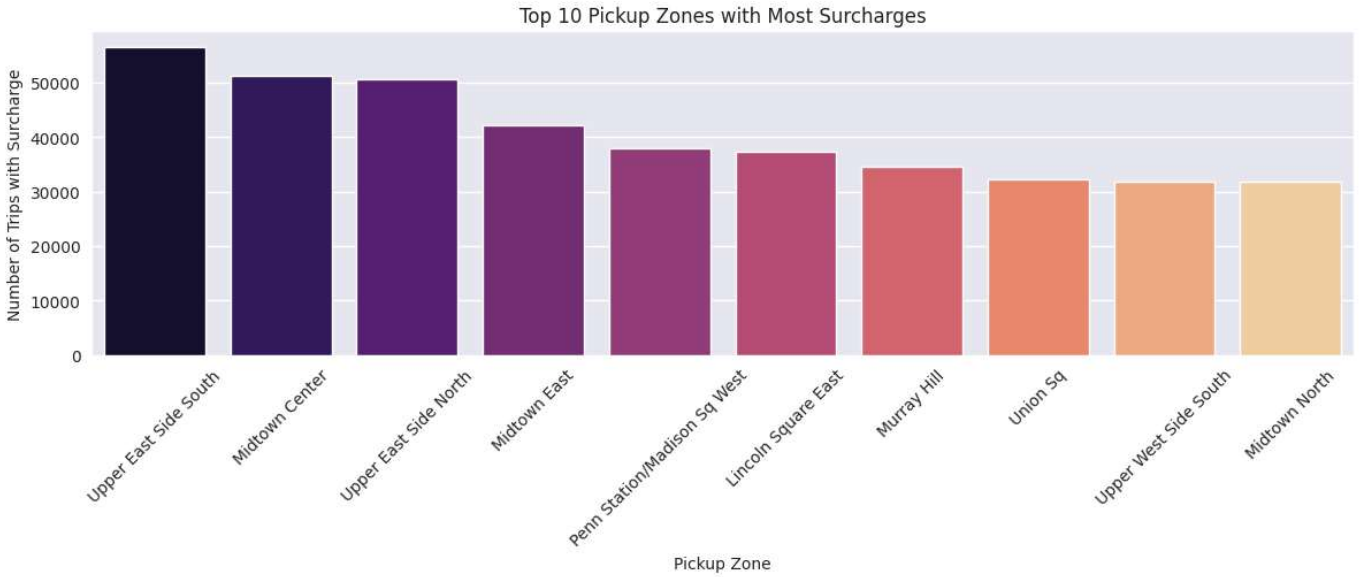
3.2.16 [5 marks]

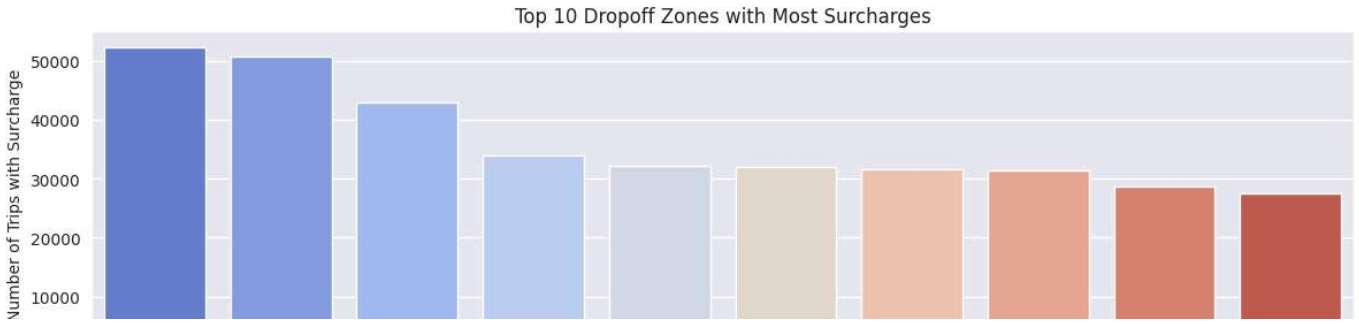
Analyse the pickup/dropoff zones or times when extra charges are applied more frequently

```
# How often is each surcharge applied?

# Define surcharge-related columns in the dataset
surcharge_columns = ["congestion_surcharge", "Airport_fee"]

# Identify trips with any surcharge applied
df["surcharge_applied"] = (df[surcharge_columns].sum(axis=1) > 0).astype(int)

# Count how often each surcharge is applied
surcharge_counts = df[surcharge_columns].apply(lambda x: (x > 0).sum())

# Convert to DataFrame for plotting
surcharge_counts_df = surcharge_counts.reset_index()
surcharge_counts_df.columns = ["Surcharge Type", "Count"]
```

```
# Extract hour from pickup datetime
df["pickup_hour"] = pd.to_datetime(df["tpep_pickup_datetime"]).dt.hour

# Count surcharge occurrences per hour
hourly_surcharge_counts = df.groupby("pickup_hour")["surcharge_applied"].sum().reset_index()

# Count surcharge occurrences per pickup and dropoff zone
pickup_surcharge_counts = df.groupby("PULocationID")["surcharge_applied"].sum().reset_index()
dropoff_surcharge_counts = df.groupby("DOLocationID")["surcharge_applied"].sum().reset_index()

# Merge with taxi zone names
pickup_surcharge_counts = pickup_surcharge_counts.merge(zones[["LocationID", "zone"]],
                                                        left_on="PULocationID", right_on="LocationID", how="left")
dropoff_surcharge_counts = dropoff_surcharge_counts.merge(zones[["LocationID", "zone"]],
                                                          left_on="DOLocationID", right_on="LocationID", how="left")

# Sort by highest surcharge occurrence
top_pickup_surcharge_zones = pickup_surcharge_counts.sort_values(by="surcharge_applied", ascending=False).head(10)
top_dropoff_surcharge_zones = dropoff_surcharge_counts.sort_values(by="surcharge_applied", ascending=False).head(10)

# Visualization: Plot everything
fig, axes = plt.subplots(3, 1, figsize=(12, 15))

# Plot surcharge frequency
sns.barplot(x="Surcharge Type", y="Count", data=surcharge_counts_df, palette="viridis", ax=axes[0])
axes[0].set_title("Frequency of Surcharge Application")
axes[0].set_xlabel("Surcharge Type")
axes[0].set_ylabel("Number of Trips")

# Plot surcharge application by pickup/dropoff zone
sns.barplot(x="zone", y="surcharge_applied", data=top_pickup_surcharge_zones, palette="magma", ax=axes[1])
axes[1].set_title("Top 10 Pickup Zones with Most Surcharges")
axes[1].set_xlabel("Pickup Zone")
axes[1].set_ylabel("Number of Trips with Surcharge")
axes[1].tick_params(axis="x", rotation=45)

sns.barplot(x="zone", y="surcharge_applied", data=top_dropoff_surcharge_zones, palette="coolwarm", ax=axes[2])
axes[2].set_title("Top 10 Dropoff Zones with Most Surcharges")
axes[2].set_xlabel("Dropoff Zone")
axes[2].set_ylabel("Number of Trips with Surcharge")
axes[2].tick_params(axis="x", rotation=45)
plt.tight_layout()
plt.show()

# Plot surcharge application by time of day
plt.figure(figsize=(10, 5))
sns.lineplot(x="pickup_hour", y="surcharge_applied", data=hourly_surcharge_counts, marker="o", color="red")
plt.xlabel("Hour of Day")
plt.ylabel("Number of Trips with Surcharge")
plt.title("Surcharge Application by Hour of Day")
plt.xticks(range(0, 24))
plt.grid(True)
plt.show()
```


Frequency of Surcharge Application
1e6
1.0
0.8
0.6
0.4
0.2
0.0
Number of Trips
congestion_surcharge
Airport_fee
Surcharge Type


Top 10 Pickup Zones with Most Surcharges
Number of Trips with Surcharge
50000
40000
30000
20000
10000
0
Upper East Side South
Midtown Center
Upper East Side North
Midtown East
Penn Station/Madison Sq West
Lincoln Square East
Murray Hill
Union Sq
Upper West Side South
Midtown North
Pickup Zone


Top 10 Dropoff Zones with Most Surcharges
Number of Trips with Surcharge
50000
40000
30000
20000
10000