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

NOTE: The marks given along with headings and sub-headings are cumulative marks for those particular headings/sub-headings.

The actual marks for each task are specified within the tasks themselves.

For example, marks given with heading *2* or sub-heading *2.1* are the cumulative marks, for your reference only.

The marks you will receive for completing tasks are given with the tasks.

Suppose the marks for two tasks are: 3 marks for 2.1.1 and 2 marks for 3.2.2, or

- 2.1.1 [3 marks]
- 3.2.2 [2 marks]

then, you will earn 3 marks for completing task 2.1.1 and 2 marks for completing task 3.2.2.

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
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

Field Name	description
	record was held in vehicle memory before sending to the vendor, aka "store and forward," because the vehicle did not have a connection to the server. 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 numpy as np
import pandas as pd
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


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

```
numpy version: 1.26.4
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('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.

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

# from google.colab import drive
# drive.mount('/content/drive')

# 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
import os


os.chdir(r'C:\Users\ADMIN\AL ML Coursesss\trip_records')

file_list = os.listdir()


df = pd.DataFrame()


for file_name in file_list:
    try:
        file_path = os.path.join(os.getcwd(), file_name)
```

```
        if file_name.endswith('.parquet'):
            sampled_data = pd.read_parquet(file_path,
engine='pyarrow')


            sampled_data = sampled_data.sample(frac=0.007,
random_state=42)


            df = pd.concat([df, sampled_data], ignore_index=True)

    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_csv('data_New.csv', index=False)
```

2 Data Cleaning

[30 marks]

Now we can load the new data directly.

```
# Load the new data file
df = pd.read_csv('data_New.csv')

df.head()

   VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
0         1  2023-01-05 07:50:08   2023-01-05 08:02:04
2.0
1         2  2023-01-17 07:47:24   2023-01-17 08:00:50
5.0
2         2  2023-01-25 21:57:59   2023-01-25 22:00:33
1.0
3         2  2023-01-09 19:36:54   2023-01-09 19:52:01
2.0
4         1  2023-01-11 22:19:13   2023-01-11 22:32:37
1.0

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID
DOLocationID  \
0           1.90         1.0                  N           239
```

```
236
1           1.86         1.0                  N            239
162
2           0.50         1.0                  N            162
170
3           2.56         1.0                  N            162
262
4           2.80         1.0                  N            164
231

   payment_type  fare_amount  extra  mta_tax  tip_amount  tolls_amount
\
0             1         13.5    2.5      0.5        2.50           0.0

1             1         14.2    0.0      0.5        3.64           0.0

2             1          5.1    1.0      0.5        2.02           0.0

3             1         17.0    2.5      0.5        4.70           0.0

4             1         14.9    3.5      0.5        3.98           0.0


   improvement_surcharge  total_amount  congestion_surcharge
airport_fee  \
0                    1.0         20.00                   2.5
0.0
1                    1.0         21.84                   2.5
0.0
2                    1.0         12.12                   2.5
0.0
3                    1.0         28.20                   2.5
0.0
4                    1.0         23.88                   2.5
0.0

   Airport_fee
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 265500 entries, 0 to 265499
Data columns (total 20 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
```

```
 0   VendorID               265500 non-null  int64
 1   tpep_pickup_datetime   265500 non-null  object
 2   tpep_dropoff_datetime  265500 non-null  object
 3   passenger_count        256470 non-null  float64
 4   trip_distance          265500 non-null  float64
 5   RatecodeID             256470 non-null  float64
 6   store_and_fwd_flag     256470 non-null  object
 7   PULocationID           265500 non-null  int64
 8   DOLocationID           265500 non-null  int64
 9   payment_type           265500 non-null  int64
 10  fare_amount            265500 non-null  float64
 11  extra                  265500 non-null  float64
 12  mta_tax                265500 non-null  float64
 13  tip_amount             265500 non-null  float64
 14  tolls_amount           265500 non-null  float64
 15  improvement_surcharge  265500 non-null  float64
 16  total_amount           265500 non-null  float64
 17  congestion_surcharge   256470 non-null  float64
 18  airport_fee            20788 non-null   float64
 19  Airport_fee            235682 non-null  float64
dtypes: float64(13), int64(4), object(3)
memory usage: 40.5+ 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
unnecessary_columns = [col for col in df.columns if "Unnamed" in col
or "index" in col.lower()]
df = df.drop(columns=unnecessary_columns, errors='ignore')
```

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

if 'airport_fee' in df.columns and 'Airport_fee' in df.columns:

    df['Combined_Airport_Fee'] = df['airport_fee'].fillna(0) +
df['Airport_fee'].fillna(0)


    df = df.drop(columns=['airport_fee', 'Airport_fee'])
```

```
df.head()

   VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
0         1  2023-01-05 07:50:08   2023-01-05 08:02:04
2.0
1         2  2023-01-17 07:47:24   2023-01-17 08:00:50
5.0
2         2  2023-01-25 21:57:59   2023-01-25 22:00:33
1.0
3         2  2023-01-09 19:36:54   2023-01-09 19:52:01
2.0
4         1  2023-01-11 22:19:13   2023-01-11 22:32:37
1.0

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID
DOLocationID  \
0           1.90         1.0                  N           239
236
1           1.86         1.0                  N           239
162
2           0.50         1.0                  N           162
170
3           2.56         1.0                  N           162
262
4           2.80         1.0                  N           164
231

   payment_type  fare_amount  extra  mta_tax  tip_amount  tolls_amount
\
0             1         13.5    2.5      0.5        2.50           0.0

1             1         14.2    0.0      0.5        3.64           0.0

2             1          5.1    1.0      0.5        2.02           0.0

3             1         17.0    2.5      0.5        4.70           0.0

4             1         14.9    3.5      0.5        3.98           0.0


   improvement_surcharge  total_amount  congestion_surcharge  \
0                    1.0         20.00                   2.5
1                    1.0         21.84                   2.5
2                    1.0         12.12                   2.5
3                    1.0         28.20                   2.5
4                    1.0         23.88                   2.5

   Combined_Airport_Fee
```

```
0                    0.0
1                    0.0
2                    0.0
3                    0.0
4                    0.0
```

2.1.4 [5 marks] Fix columns with negative (monetary) values

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

print(df["fare_amount"].min())

# No negative value

0.0
```

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

Yes we have found 99 in the record

```
# Analyse RatecodeID for the negative fare amounts
df['RatecodeID'].value_counts()

RatecodeID
1.0     241967
2.0      10228
5.0       1488
99.0      1452
3.0        829
4.0        506
Name: count, dtype: int64

# Find which columns have negative values
df = df[df["RatecodeID"] != 99.0]
print(df["RatecodeID"].value_counts())

RatecodeID
1.0    241967
2.0     10228
5.0      1488
3.0       829
4.0       506
Name: count, dtype: int64

df = df[df["RatecodeID"] != 99.0]
print(df["RatecodeID"].value_counts())

RatecodeID
1.0    241967
2.0     10228
```

```
5.0      1488
3.0       829
4.0       506
Name: count, dtype: int64
```

```python
# fix these negative values
print(df.describe())
```

```
            VendorID  passenger_count  trip_distance     RatecodeID  \
count  264048.000000    255018.000000  264048.000000  255018.000000   
mean        1.741608         1.374350       3.638318       1.075901   
std         0.442827         0.899321      80.015295       0.398203   
min         1.000000         0.000000       0.000000       1.000000   
25%         1.000000         1.000000       1.040000       1.000000   
50%         2.000000         1.000000       1.780000       1.000000   
75%         2.000000         1.000000       3.380000       1.000000   
max         6.000000         8.000000   37523.740000       5.000000   

        PULocationID   DOLocationID   payment_type    fare_amount  \
count  264048.000000  264048.000000  264048.000000  264048.000000   
mean      165.313057     164.200104       1.165224      19.779402   
std        63.884569      69.654169       0.510045      18.308347   
min         1.000000       1.000000       0.000000       0.000000   
25%       132.000000     114.000000       1.000000       9.300000   
50%       162.000000     162.000000       1.000000      13.500000   
75%       234.000000     234.000000       1.000000      21.900000   
max       265.000000     265.000000       4.000000     750.000000   

               extra        mta_tax     tip_amount   tolls_amount  \
count  264048.000000  264048.000000  264048.000000  264048.000000   
mean        1.586107       0.495319       3.577066       0.591301   
std         1.828371       0.048421       4.085435       2.170130   
min        -1.000000      -0.500000       0.000000       0.000000   
25%         0.000000       0.500000       1.000000       0.000000   
50%         1.000000       0.500000       2.860000       0.000000   
75%         2.500000       0.500000       4.450000       0.000000   
max        13.000000       0.800000     288.000000     132.040000   

       improvement_surcharge   total_amount  congestion_surcharge  \
count          264048.000000  264048.000000         255018.000000   
mean                0.999140      28.890812              2.323542   
std                 0.029012      22.919192              0.640664   
min                -1.000000      -5.000000             -2.500000   
25%                 1.000000      15.960000              2.500000   
50%                 1.000000      21.000000              2.500000   
75%                 1.000000      30.720000              2.500000   
max                 1.000000     757.940000              2.500000   

       Combined_Airport_Fee  
count         264048.000000  
```

```
mean                 0.140314
std                  0.461338
min                 -1.250000
25%                  0.000000
50%                  0.000000
75%                  0.000000
max                  1.750000
```

```
# fix these negative values
monetary_columns = ["extra", "mta_tax", "improvement_surcharge",
                    "total_amount", "congestion_surcharge",
"Combined_Airport_Fee"]

df[monetary_columns] = df[monetary_columns].abs()

print(df.describe())
```

```
            VendorID  passenger_count  trip_distance     RatecodeID  \
count  264048.000000    255018.000000  264048.000000  255018.000000
mean        1.741608         1.374350       3.638318       1.075901
std         0.442827         0.899321      80.015295       0.398203
min         1.000000         0.000000       0.000000       1.000000
25%         1.000000         1.000000       1.040000       1.000000
50%         2.000000         1.000000       1.780000       1.000000
75%         2.000000         1.000000       3.380000       1.000000
max         6.000000         8.000000   37523.740000       5.000000

        PULocationID   DOLocationID   payment_type    fare_amount  \
count  264048.000000  264048.000000  264048.000000  264048.000000
mean      165.313057     164.200104       1.165224      19.779402
std        63.884569      69.654169       0.510045      18.308347
min         1.000000       1.000000       0.000000       0.000000
25%       132.000000     114.000000       1.000000       9.300000
50%       162.000000     162.000000       1.000000      13.500000
75%       234.000000     234.000000       1.000000      21.900000
max       265.000000     265.000000       4.000000     750.000000

               extra        mta_tax     tip_amount   tolls_amount  \
count  264048.000000  264048.000000  264048.000000  264048.000000
mean        1.586114       0.495364       3.577066       0.591301
std         1.828364       0.047954       4.085435       2.170130
min         0.000000       0.000000       0.000000       0.000000
25%         0.000000       0.500000       1.000000       0.000000
50%         1.000000       0.500000       2.860000       0.000000
75%         2.500000       0.500000       4.450000       0.000000
max        13.000000       0.800000     288.000000     132.040000

       improvement_surcharge   total_amount  congestion_surcharge  \
count          264048.000000  264048.000000         255018.000000
mean                0.999246      28.891161              2.323718
```

```
std                        0.025095      22.918753                  0.640024
min                        0.000000       0.000000                  0.000000
25%                        1.000000      15.960000                  2.500000
50%                        1.000000      21.000000                  2.500000
75%                        1.000000      30.720000                  2.500000
max                        1.000000     757.940000                  2.500000

       Combined_Airport_Fee
count         264048.000000
mean               0.140333
std                0.461332
min                0.000000
25%                0.000000
50%                0.000000
75%                0.000000
max                1.750000
```

2.2 Handling Missing Values

[10 marks]

2.2.1 [2 marks] Find the proportion of missing values in each column

```python
# Find the proportion of missing values in each column
df.isnull().sum()
```

```
VendorID                    0
tpep_pickup_datetime        0
tpep_dropoff_datetime       0
passenger_count          9030
trip_distance               0
RatecodeID               9030
store_and_fwd_flag       9030
PULocationID                0
DOLocationID                0
payment_type                0
fare_amount                 0
extra                       0
mta_tax                     0
tip_amount                  0
tolls_amount                0
improvement_surcharge       0
total_amount                0
congestion_surcharge     9030
Combined_Airport_Fee        0
dtype: int64
```

2.2.2 [3 marks] Handling missing values in `passenger_count`

```
# Display the rows with null values

df[df['passenger_count'].isnull()]

# Impute NaN values in 'passenger_count'

        VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
155            2  2023-01-20 15:15:17   2023-01-20 15:27:06
NaN
157            2  2023-01-21 15:17:59   2023-01-21 15:39:36
NaN
303            2  2023-01-28 23:58:47   2023-01-29 00:11:44
NaN
309            1  2023-01-21 02:51:57   2023-01-21 03:15:22
NaN
356            2  2023-01-17 21:22:28   2023-01-17 21:36:25
NaN
...          ...                  ...                   ...
...
265403         1  2023-09-13 22:16:27   2023-09-13 22:29:21
NaN
265421         1  2023-09-11 21:23:50   2023-09-11 21:35:41
NaN
265445         2  2023-09-19 21:24:31   2023-09-19 21:41:47
NaN
265448         1  2023-09-08 22:38:30   2023-09-08 22:50:37
NaN
265477         2  2023-09-12 22:04:59   2023-09-12 22:21:59
NaN

        trip_distance  RatecodeID store_and_fwd_flag  PULocationID  \
155              1.35         NaN                NaN           142
157              6.67         NaN                NaN           170
303              1.73         NaN                NaN           211
309              0.00         NaN                NaN            79
356              2.43         NaN                NaN           143
...               ...         ...                ...           ...
265403           0.00         NaN                NaN           233
265421           0.00         NaN                NaN           141
265445           4.00         NaN                NaN           148
265448           0.00         NaN                NaN           249
265477           4.23         NaN                NaN           161

        DOLocationID  payment_type  fare_amount  extra  mta_tax
tip_amount  \
155              239             0        13.65    0.0      0.5
3.53
157               41             0        27.71    0.0      0.5
7.34
```

```
303                    4              0        15.55    0.0      0.5
3.91
309                  166              0        22.83    0.0      0.5
0.00
356                  263              0        15.21    0.0      0.5
2.88
...                  ...            ...          ...    ...      ...
...
265403                90              0        17.11    0.0      0.5
0.00
265421               107              0        17.50    0.0      0.5
0.00
265445               181              0        21.23    0.0      0.5
5.05
265448               164              0        42.92    0.0      0.5
0.00
265477                74              0        23.54    0.0      0.5
0.00

        tolls_amount  improvement_surcharge  total_amount  \
155              0.0                    1.0         21.18
157              0.0                    1.0         39.05
303              0.0                    1.0         23.46
309              0.0                    1.0         26.83
356              0.0                    1.0         22.09
...              ...                    ...           ...
265403           0.0                    1.0         21.11
265421           0.0                    1.0         21.50
265445           0.0                    1.0         30.28
265448           0.0                    1.0         46.92
265477           0.0                    1.0         27.54

        congestion_surcharge  Combined_Airport_Fee
155                      NaN                   0.0
157                      NaN                   0.0
303                      NaN                   0.0
309                      NaN                   0.0
356                      NaN                   0.0
...                      ...                   ...
265403                   NaN                   0.0
265421                   NaN                   0.0
265445                   NaN                   0.0
265448                   NaN                   0.0
265477                   NaN                   0.0

[9030 rows x 19 columns]

# Impute NaN values in 'passenger_count'

df["passenger_count"] =
```

```
df["passenger_count"].fillna(df["passenger_count"].median())
df['passenger_count'].isnull().value_counts()
```

```
passenger_count
False    264048
Name: count, dtype: int64
```

Did you find zeroes in passenger_count? Handle these.

Yes first i could zero value in Passenger_count now i have removed it.

```
df['passenger_count'].value_counts()

df = df[df["passenger_count"] > 0]

df['passenger_count'].value_counts()
```

```
passenger_count
1.0    200322
2.0     38901
3.0      9625
4.0      5426
5.0      3459
6.0      2250
8.0         2
Name: count, dtype: int64
```

2.2.3 [2 marks] Handle missing values in `RatecodeID`

```
# Fix missing values in 'RatecodeID'
df['RatecodeID'].isnull().value_counts()

df["RatecodeID"] = df["RatecodeID"].fillna(df["RatecodeID"].median())

df['RatecodeID'].isnull().value_counts()
```

```
RatecodeID
False    259985
Name: count, dtype: int64
```

2.2.4 [3 marks] Impute NaN in `congestion_surcharge`

```
# handle null values in congestion_surcharge

df["congestion_surcharge"] =
df["congestion_surcharge"].fillna(df["congestion_surcharge"].median())
df['congestion_surcharge'].isnull().value_counts()
```

```
congestion_surcharge
False    259985
Name: count, dtype: int64
```

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
df.isnull().sum()

VendorID                   0
tpep_pickup_datetime       0
tpep_dropoff_datetime      0
passenger_count            0
trip_distance              0
RatecodeID                 0
store_and_fwd_flag      9030
PULocationID               0
DOLocationID               0
payment_type               0
fare_amount                0
extra                      0
mta_tax                    0
tip_amount                 0
tolls_amount               0
improvement_surcharge      0
total_amount               0
congestion_surcharge       0
Combined_Airport_Fee       0
dtype: int64

df["store_and_fwd_flag"] = df["store_and_fwd_flag"].fillna('N')
df['store_and_fwd_flag'].isnull().value_counts()

store_and_fwd_flag
False    259985
Name: count, dtype: int64
```

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
```

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:

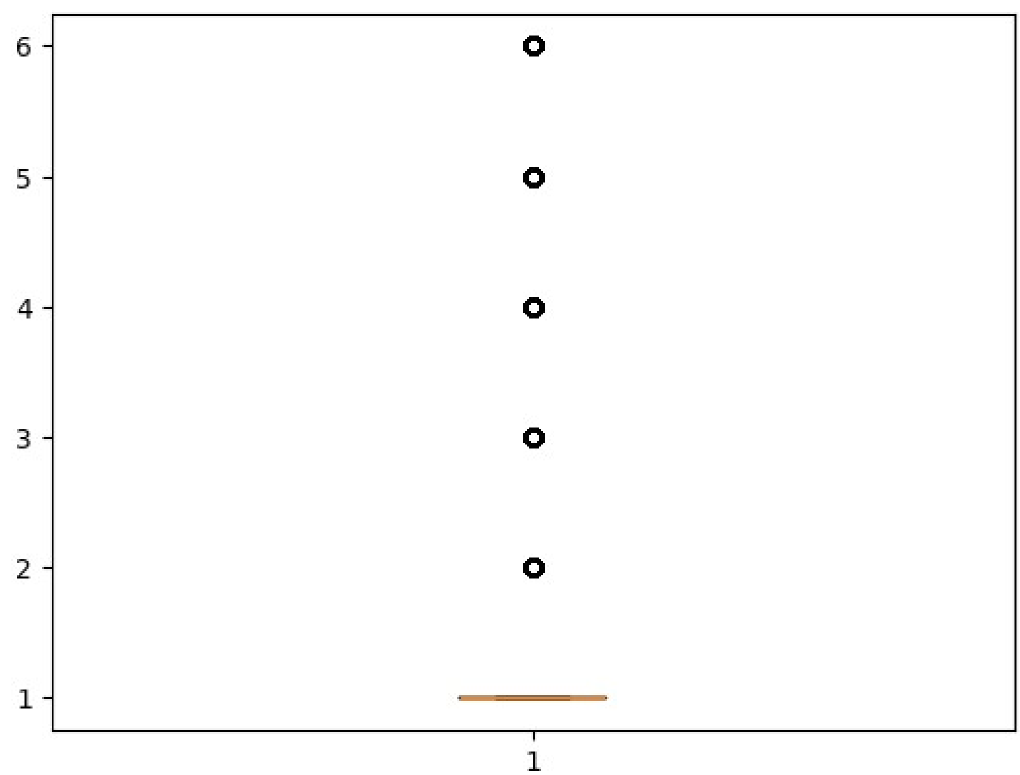
- Entries where `trip_distance` is nearly 0 and `fare_amount` is more than 300
- Entries where `trip_distance` and `fare_amount` are 0 but the pickup and dropoff zones are different (both distance and fare should not be zero for different zones)
- Entries where `trip_distance` is more than 250 miles.
- 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

df = df[df["passenger_count"] <= 7]

plt.boxplot(df['passenger_count'])
plt.show()
```



```
# Continue with outlier handling
#Entries where `trip_distance` is nearly 0 and `fare_amount` is more
than 300

filtered_df = df[(df['trip_distance'] < 0.1) & (df['fare_amount'] >
300)]
print(filtered_df)

df = df[~((df["trip_distance"] < 0.1) & (df["fare_amount"] > 300))]

filtered_df = df[(df['trip_distance'] < 0.1) & (df['fare_amount'] >
300)]
print(filtered_df)
```

```
        VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
47758          2  2023-11-26 16:04:06   2023-11-26 16:04:12
1.0
109684         2  2023-03-19 23:12:29   2023-03-19 23:12:35
1.0
210866         2  2023-05-16 19:12:48   2023-05-16 19:12:51
1.0
226046         2  2023-07-24 21:43:56   2023-07-24 21:57:57
1.0
235276         1  2023-07-20 16:17:03   2023-07-20 16:17:23
1.0

        trip_distance  RatecodeID store_and_fwd_flag  PULocationID  \
47758             0.0         5.0                  N           265
109684            0.0         5.0                  N           265
210866            0.0         5.0                  N           265
226046            0.0         5.0                  N           265
235276            0.0         5.0                  N           193

        DOLocationID  payment_type  fare_amount  extra  mta_tax
tip_amount  \
47758            265             1       305.14    0.0      0.0
61.23
109684           265             2       533.00    0.0      0.0
0.00
210866           265             1       396.00    0.0      0.5
0.80
226046           265             2       500.00    0.0      0.0
0.00
235276           193             2       555.54    0.0      0.0
0.00

        tolls_amount  improvement_surcharge  total_amount  \
47758            0.0                    1.0        367.37
109684           0.0                    1.0        534.00
```

```
210866          0.0                    1.0        398.30
226046          0.0                    1.0        501.00
235276          0.0                    1.0        556.54

        congestion_surcharge  Combined_Airport_Fee
47758                    0.0                   0.0
109684                   0.0                   0.0
210866                   0.0                   0.0
226046                   0.0                   0.0
235276                   0.0                   0.0
Empty DataFrame
Columns: [VendorID, tpep_pickup_datetime, tpep_dropoff_datetime,
passenger_count, trip_distance, RatecodeID, store_and_fwd_flag,
PULocationID, DOLocationID, payment_type, fare_amount, extra, mta_tax,
tip_amount, tolls_amount, improvement_surcharge, total_amount,
congestion_surcharge, Combined_Airport_Fee]
Index: []

#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

filtered_df1 = df[(df['trip_distance'] < 0.1) & (df['fare_amount'] <
0.01)]
print(filtered_df1)

df=  df[~((df['trip_distance'] < 0.1) & (df['fare_amount'] < 0.01))]

filtered_df1 = df[(df['trip_distance'] < 0.1) & (df['fare_amount'] <
0.01)]
print(filtered_df1)

        VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
3835           1  2023-01-24 18:31:57   2023-01-24 18:33:16
2.0
6585           2  2023-01-13 06:19:23   2023-01-13 06:19:27
1.0
9570           2  2023-01-14 12:03:42   2023-01-14 12:03:49
2.0
9649           1  2023-01-29 09:52:41   2023-01-29 09:53:54
1.0
11054          2  2023-01-19 13:28:25   2023-01-19 13:28:51
1.0
15245          2  2023-01-12 12:34:27   2023-01-12 12:35:28
1.0
19126          1  2023-01-04 13:09:40   2023-01-04 13:09:56
1.0
23206          1  2023-10-26 17:20:38   2023-10-26 17:21:18
3.0
```

```
23897           2  2023-10-07 11:51:21   2023-10-07 11:51:30
2.0
31754           2  2023-10-19 12:31:08   2023-10-19 12:31:24
1.0
37179           2  2023-10-01 15:43:56   2023-10-01 15:45:51
1.0
57266           2  2023-11-18 23:45:16   2023-11-18 23:45:38
2.0
61078           2  2023-11-04 18:07:19   2023-11-04 18:08:02
1.0
76548           2  2023-12-12 07:22:35   2023-12-12 07:22:51
1.0
84134           1  2023-12-16 16:35:24   2023-12-16 16:35:24
1.0
85711           2  2023-12-15 10:48:42   2023-12-15 10:52:12
1.0
101617          1  2023-03-01 15:51:07   2023-03-01 15:51:27
1.0
103635          1  2023-03-21 08:10:43   2023-03-21 08:11:05
1.0
110247          2  2023-03-08 10:48:46   2023-03-08 10:51:24
1.0
120079          1  2023-06-01 14:21:00   2023-06-01 14:21:49
1.0
122049          2  2023-06-04 09:46:55   2023-06-04 09:47:05
4.0
140003          2  2023-08-11 16:19:41   2023-08-11 16:20:14
1.0
140992          1  2023-08-21 05:47:49   2023-08-21 05:47:49
2.0
141216          2  2023-08-31 20:27:23   2023-08-31 20:28:20
1.0
160098          1  2023-02-18 17:49:17   2023-02-18 17:49:30
2.0
161037          1  2023-02-10 00:42:04   2023-02-10 00:42:04
1.0
162406          1  2023-02-22 10:33:21   2023-02-22 10:33:40
1.0
169469          2  2023-02-25 10:49:04   2023-02-25 10:49:12
1.0
183715          1  2023-04-17 06:32:54   2023-04-17 06:33:35
1.0
189099          2  2023-04-24 19:16:53   2023-04-24 19:16:58
1.0
189345          1  2023-04-14 00:36:54   2023-04-14 00:36:54
2.0
189954          1  2023-04-17 12:45:58   2023-04-17 12:46:29
1.0
194062          2  2023-04-29 10:24:56   2023-04-29 10:26:26
```

```
1.0
206373         2  2023-05-04 11:56:27   2023-05-04 11:56:37
1.0
209872         2  2023-05-04 18:40:47   2023-05-04 18:40:55
1.0
212354         2  2023-05-29 16:38:17   2023-05-29 16:38:27
2.0
224069         1  2023-05-29 06:59:22   2023-05-29 06:59:22
1.0
224660         2  2023-05-23 13:49:46   2023-05-23 13:50:01
1.0
234318         1  2023-07-13 17:40:21   2023-07-13 17:41:47
1.0
246157         2  2023-09-27 17:02:30   2023-09-27 17:08:36
1.0
250671         1  2023-09-06 00:23:52   2023-09-06 00:25:30
1.0

        trip_distance  RatecodeID store_and_fwd_flag  PULocationID  \
3835             0.00         2.0                  Y           237
6585             0.00         1.0                  N           132
9570             0.00         2.0                  N            13
9649             0.00         5.0                  N           188
11054            0.00         1.0                  N           193
15245            0.00         1.0                  N           264
19126            0.00         5.0                  N           132
23206            0.00         4.0                  N             7
23897            0.01         2.0                  N           238
31754            0.00         1.0                  N           264
37179            0.00         1.0                  N           193
57266            0.00         1.0                  N           162
61078            0.00         1.0                  N           246
76548            0.00         1.0                  N           239
84134            0.00         5.0                  Y           163
85711            0.04         2.0                  N           261
101617           0.00         5.0                  N           132
103635           0.00         1.0                  N           237
110247           0.00         1.0                  N           193
120079           0.00         1.0                  N           100
122049           0.00         5.0                  N           265
140003           0.00         1.0                  N           264
140992           0.00         3.0                  Y           162
141216           0.01         1.0                  N           264
160098           0.00         1.0                  N            50
161037           0.00         1.0                  N            26
162406           0.00         5.0                  N            68
169469           0.00         1.0                  N           264
183715           0.00         1.0                  N            89
189099           0.00         1.0                  N           132
```

```
189345          0.00         5.0                    Y            90
189954          0.00         5.0                    N           145
194062          0.03         1.0                    N           230
206373          0.00         1.0                    N           193
209872          0.00         2.0                    N           170
212354          0.00         1.0                    N            68
224069          0.00         2.0                    Y           132
224660          0.00         2.0                    N           239
234318          0.00         5.0                    N           132
246157          0.00         5.0                    N           246
250671          0.00         5.0                    N           107

        DOLocationID  payment_type  fare_amount  extra  mta_tax
tip_amount  \
3835             237             3          0.0   0.00      0.0
0.0
6585             132             2          0.0   0.00      0.5
0.0
9570              13             2          0.0   0.00      0.5
0.0
9649             188             3          0.0   0.00      0.0
0.0
11054            193             1          0.0   0.00      0.0
0.0
15245            264             1          0.0   0.00      0.0
0.0
19126            132             2          0.0   1.25      0.0
0.0
23206              7             3          0.0   2.50      0.5
0.0
23897            238             2          0.0   0.00      0.5
0.0
31754            264             2          0.0   0.00      0.0
0.0
37179            193             1          0.0   0.00      0.0
0.0
57266            162             3          0.0   1.00      0.5
0.0
61078            246             2          0.0   0.00      0.5
0.0
76548            239             2          0.0   0.00      0.5
0.0
84134            264             2          0.0   0.00      0.0
0.0
85711            261             2          0.0   0.00      0.5
0.0
101617           132             3          0.0   1.25      0.0
0.0
103635           237             3          0.0   0.00      0.0
```

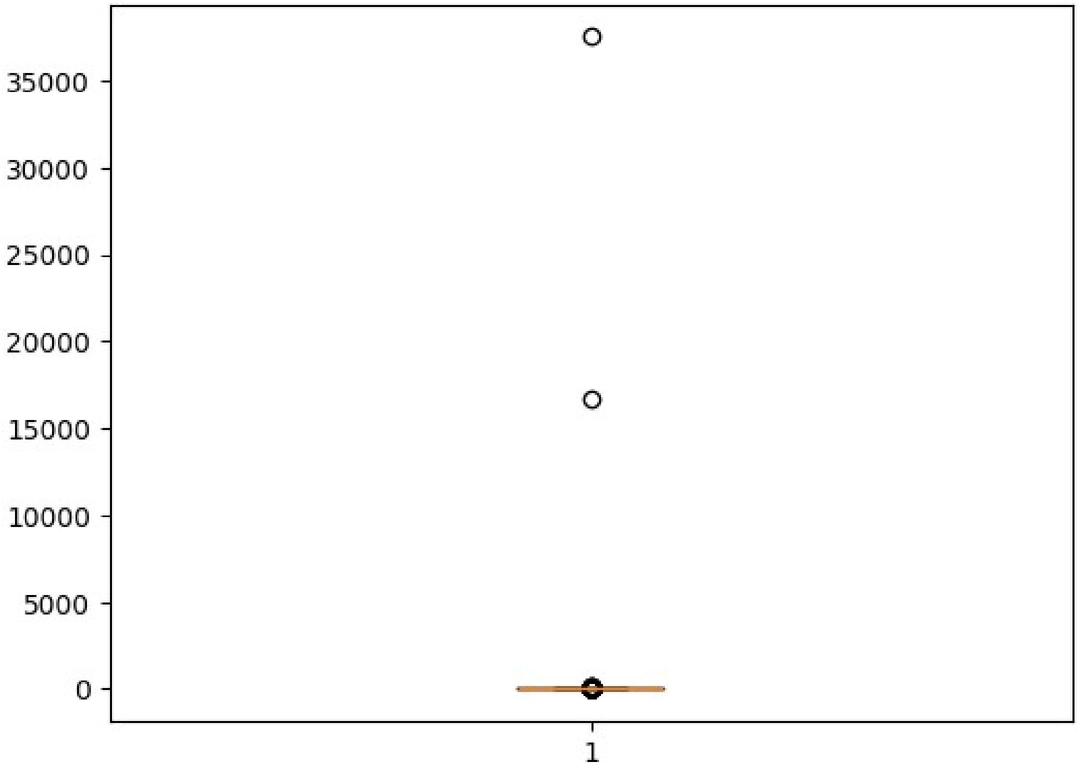
```
0.0
110247         193            1         0.0   0.00       0.0
0.0
120079         100            3         0.0   0.00       0.0
0.0
122049         265            2         0.0   0.00       0.0
0.0
140003         264            1         0.0   0.00       0.0
0.0
140992         264            3         0.0   0.00       0.0
0.0
141216         264            2         0.0   0.00       0.5
0.0
160098          50            4         0.0   0.00       0.0
0.0
161037          26            1         0.0   0.00       0.0
0.0
162406          68            2         0.0   0.00       0.0
0.0
169469         264            1         0.0   0.00       0.0
0.9
183715          89            1         0.0   0.00       0.0
0.0
189099         132            2         0.0   0.00       0.5
0.0
189345         264            2         0.0   0.00       0.0
0.0
189954         145            1         0.0   0.00       0.0
0.0
194062         230            2         0.0   0.00       0.5
0.0
206373         193            1         0.0   0.00       0.0
0.0
209872         170            2         0.0   0.00       0.5
0.0
212354          68            2         0.0   0.00       0.5
0.0
224069         264            3         0.0   0.00       0.0
0.0
224660         239            2         0.0   0.00       0.5
0.0
234318         132            2         0.0   0.00       0.0
0.0
246157         246            2         0.0   0.00       0.0
0.0
250671         107            3         0.0   0.00       0.0
0.0

        tolls_amount  improvement_surcharge  total_amount  \
```

3835	0.0	0.0	0.00
6585	0.0	1.0	2.75
9570	0.0	1.0	4.00
9649	0.0	1.0	1.00
11054	0.0	0.0	0.00
15245	0.0	0.0	0.00
19126	0.0	1.0	2.25
23206	0.0	1.0	4.00
23897	0.0	1.0	4.00
31754	0.0	0.0	0.00
37179	0.0	0.0	0.00
57266	0.0	1.0	5.00
61078	0.0	1.0	4.00
76548	0.0	1.0	4.00
84134	0.0	0.0	0.00
85711	0.0	1.0	4.00
101617	0.0	1.0	2.25
103635	0.0	0.0	0.00
110247	0.0	0.0	0.00
120079	0.0	0.0	0.00
122049	0.0	1.0	1.00
140003	0.0	0.0	0.00
140992	0.0	0.0	0.00
141216	0.0	1.0	4.00
160098	0.0	0.0	0.00
161037	0.0	0.0	0.00
162406	0.0	1.0	1.00
169469	0.0	0.0	0.90
183715	0.0	0.0	0.00
189099	0.0	1.0	1.50
189345	0.0	0.0	0.00
189954	0.0	0.0	0.00
194062	0.0	1.0	4.00
206373	0.0	0.0	0.00
209872	0.0	1.0	4.00
212354	0.0	1.0	4.00
224069	0.0	0.0	0.00
224660	0.0	1.0	4.00
234318	0.0	1.0	1.00
246157	0.0	1.0	3.50
250671	0.0	1.0	1.00

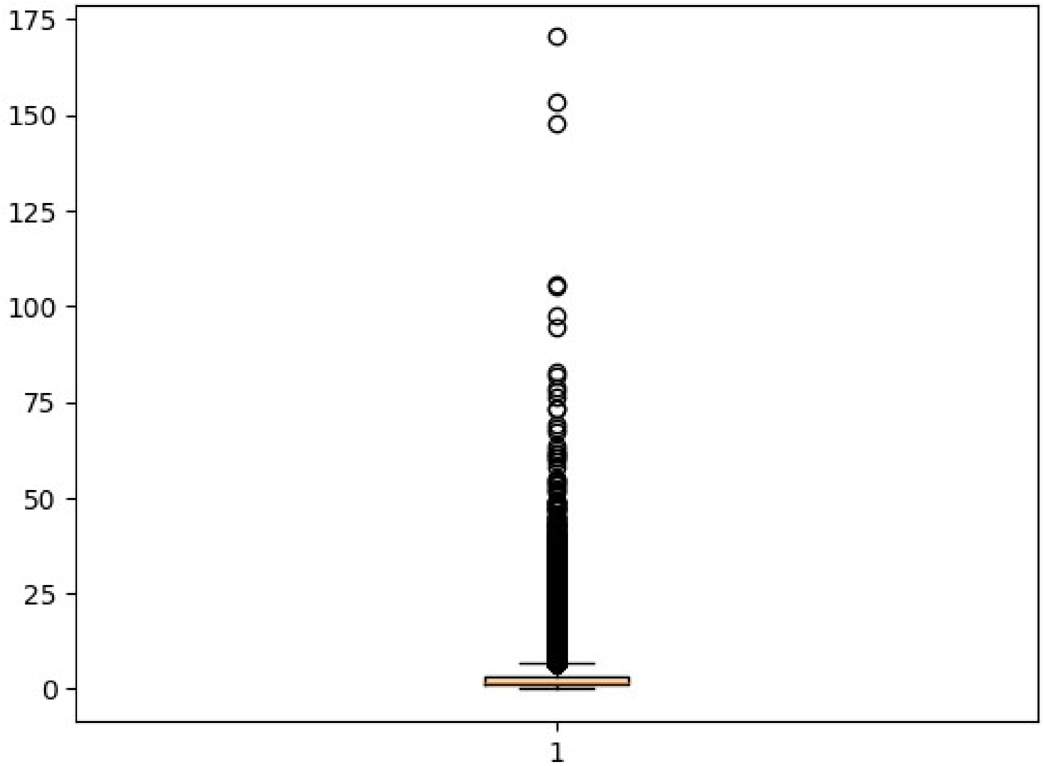
	congestion_surcharge	Combined_Airport_Fee
3835	0.0	0.00
6585	0.0	1.25
9570	2.5	0.00
9649	0.0	0.00
11054	0.0	0.00
15245	0.0	0.00

```
19126                    0.0                   1.25
23206                    0.0                   0.00
23897                    2.5                   0.00
31754                    0.0                   0.00
37179                    0.0                   0.00
57266                    2.5                   0.00
61078                    2.5                   0.00
76548                    2.5                   0.00
84134                    0.0                   0.00
85711                    2.5                   0.00
101617                   0.0                   1.25
103635                   0.0                   0.00
110247                   0.0                   0.00
120079                   0.0                   0.00
122049                   0.0                   0.00
140003                   0.0                   0.00
140992                   0.0                   0.00
141216                   2.5                   0.00
160098                   0.0                   0.00
161037                   0.0                   0.00
162406                   0.0                   0.00
169469                   0.0                   0.00
183715                   0.0                   0.00
189099                   0.0                   0.00
189345                   0.0                   0.00
189954                   0.0                   0.00
194062                   2.5                   0.00
206373                   0.0                   0.00
209872                   2.5                   0.00
212354                   2.5                   0.00
224069                   0.0                   0.00
224660                   2.5                   0.00
234318                   0.0                   0.00
246157                   2.5                   0.00
250671                   0.0                   0.00
Empty DataFrame
Columns: [VendorID, tpep_pickup_datetime, tpep_dropoff_datetime,
passenger_count, trip_distance, RatecodeID, store_and_fwd_flag,
PULocationID, DOLocationID, payment_type, fare_amount, extra, mta_tax,
tip_amount, tolls_amount, improvement_surcharge, total_amount,
congestion_surcharge, Combined_Airport_Fee]
Index: []
```

```
 #Entries where `trip_distance` is more than 250  miles.

plt.boxplot(df['trip_distance'])
plt.show()
```




```
df = df[~(df["trip_distance"] > 250)]

plt.boxplot(df['trip_distance'])
plt.show()
```




```
#Entries where `payment_type` is 0 (there is no payment_type 0 defined
in the data dictionary

df['payment_type'].value_counts()

payment_type
1    204312
2     43539
0      9028
4      1930
3      1126
Name: count, dtype: int64

#Entries where `payment_type` is 0 (there is no payment_type 0 defined
in the data dictionary

df = df[df["payment_type"] != 0]

df['payment_type'].value_counts()

payment_type
1    204312
2     43539
4      1930
3      1126
Name: count, dtype: int64
```

```python
# Do any columns need standardising?

print(df.describe())
```

```
            VendorID  passenger_count  trip_distance     RatecodeID  \
count  250907.000000    250907.000000  250907.000000  250907.000000   
mean        1.757583         1.396569       3.444610       1.075733   
std         0.428546         0.889117       4.586306       0.396638   
min         1.000000         1.000000       0.000000       1.000000   
25%         2.000000         1.000000       1.050000       1.000000   
50%         2.000000         1.000000       1.780000       1.000000   
75%         2.000000         1.000000       3.350000       1.000000   
max         2.000000         6.000000     170.300000       5.000000   

        PULocationID   DOLocationID   payment_type    fare_amount  \
count  250907.000000  250907.000000  250907.000000  250907.000000   
mean      165.403125     164.414807       1.205578      19.721341   
std        63.575002      69.610189       0.467381      18.367818   
min         1.000000       1.000000       1.000000       0.000000   
25%       132.000000     114.000000       1.000000       9.300000   
50%       162.000000     162.000000       1.000000      13.500000   
75%       234.000000     234.000000       1.000000      21.900000   
max       265.000000     265.000000       4.000000     750.000000   

               extra        mta_tax     tip_amount   tolls_amount  \
count  250907.000000  250907.000000  250907.000000  250907.000000   
mean        1.609131       0.495418       3.613894       0.594723   
std         1.826910       0.047678       4.112515       2.176181   
min         0.000000       0.000000       0.000000       0.000000   
25%         0.000000       0.500000       1.000000       0.000000   
50%         1.000000       0.500000       2.860000       0.000000   
75%         2.500000       0.500000       4.480000       0.000000   
max        11.750000       0.800000     288.000000     132.040000   

       improvement_surcharge   total_amount  congestion_surcharge  \
count          250907.000000  250907.000000         250907.000000   
mean                0.999546      28.911278              2.323819   
std                 0.019357      23.054619              0.639855   
min                 0.000000       0.000000              0.000000   
25%                 1.000000      15.960000              2.500000   
50%                 1.000000      21.000000              2.500000   
75%                 1.000000      30.620000              2.500000   
max                 1.000000     757.940000              2.500000   

       Combined_Airport_Fee  
count         250907.000000  
mean               0.146118  
std                0.469900  
min                0.000000  
25%                0.000000  
```

```
50%                  0.000000
75%                  0.000000
max                  1.750000
```

3 Exploratory Data Analysis

[90 marks]

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

```
['VendorID',
 'tpep_pickup_datetime',
 'tpep_dropoff_datetime',
 'passenger_count',
 'trip_distance',
 'RatecodeID',
 'store_and_fwd_flag',
 'PULocationID',
 'DOLocationID',
 'payment_type',
 'fare_amount',
 'extra',
 'mta_tax',
 'tip_amount',
 'tolls_amount',
 'improvement_surcharge',
 'total_amount',
 'congestion_surcharge',
 'Combined_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`:

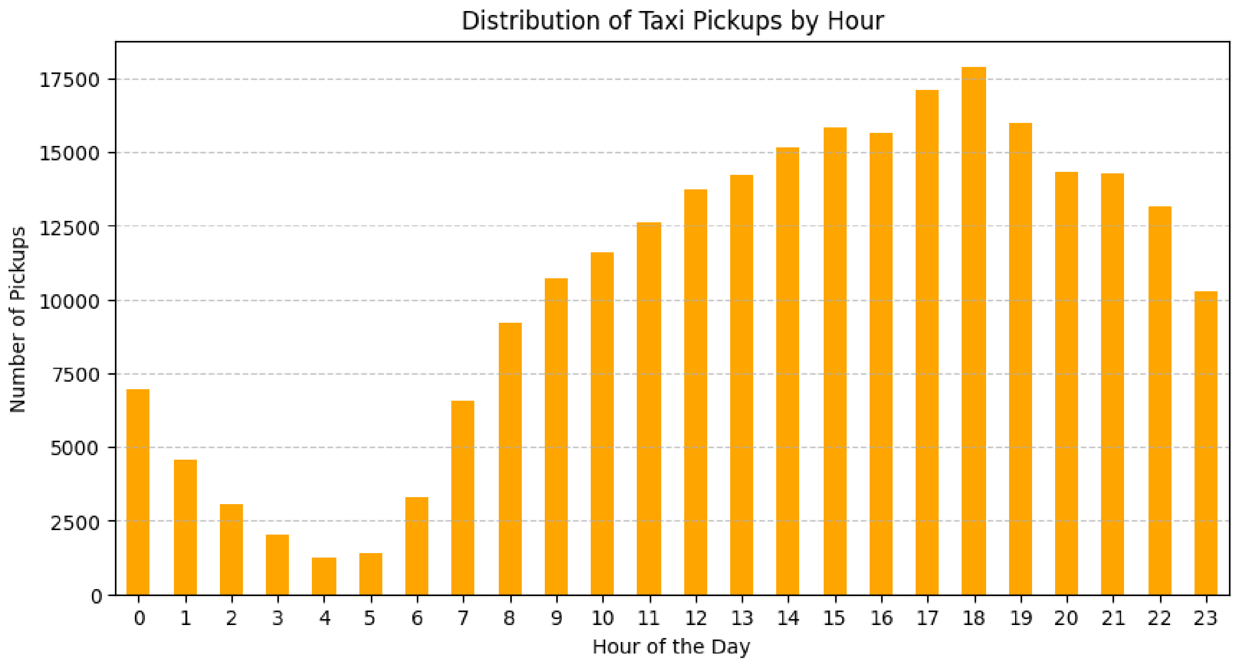
The following monetary parameters belong in the same category, is it categorical or numerical?

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

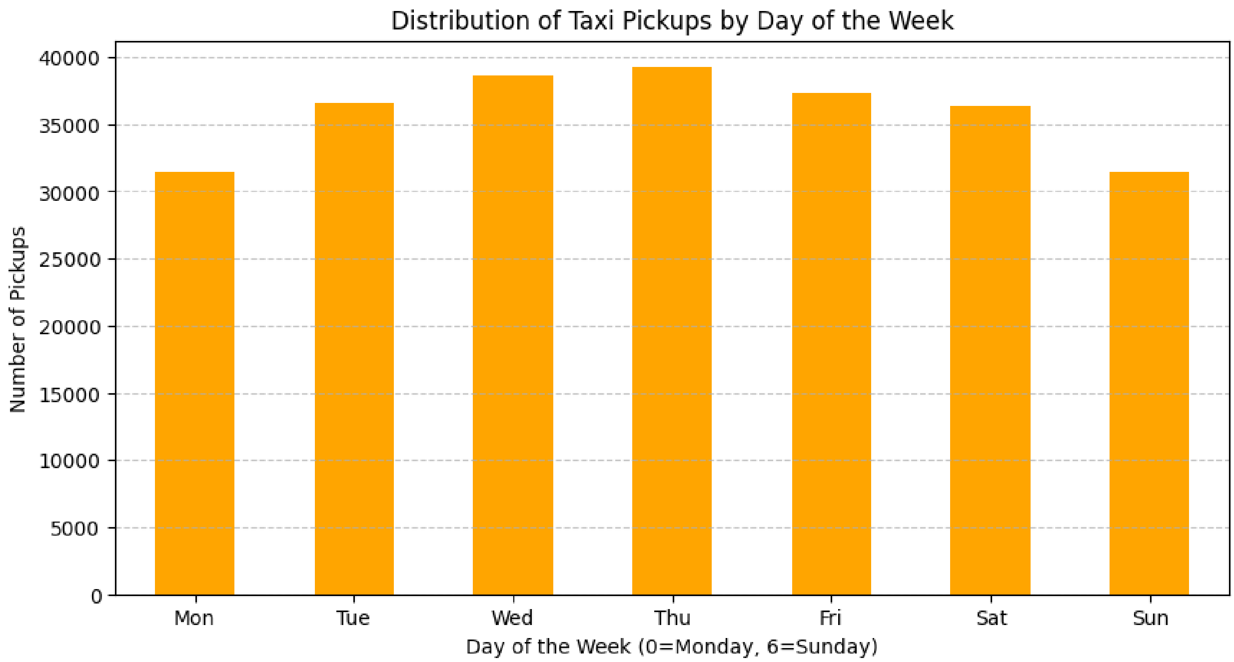
Temporal Analysis

3.1.2 [5 marks] Analyse the distribution of taxi pickups by hours, days of the week, and months.

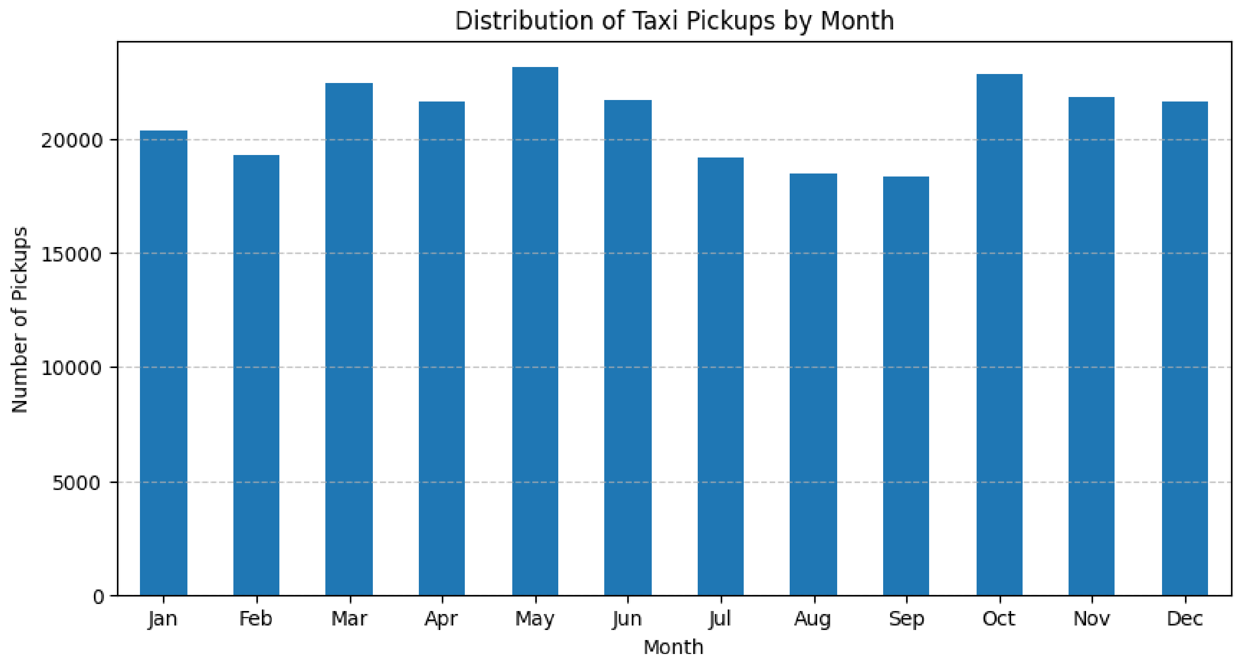
```
# Find and show the hourly trends in taxi pickups

df["tpep_pickup_datetime"] =
pd.to_datetime(df["tpep_pickup_datetime"])


df["pickup_hour"] = df["tpep_pickup_datetime"].dt.hour

plt.figure(figsize=(10, 5))
df["pickup_hour"].value_counts().sort_index().plot(kind="bar",
color="orange")
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Pickups")
plt.title("Distribution of Taxi Pickups by Hour")
plt.xticks(rotation=0)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```




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

df["pickup_day"] = df["tpep_pickup_datetime"].dt.dayofweek
plt.figure(figsize=(10, 5))
df["pickup_day"].value_counts().sort_index().plot(kind="bar",color="or
ange")
plt.xlabel("Day of the Week (0=Monday, 6=Sunday)")
plt.ylabel("Number of Pickups")
plt.title("Distribution of Taxi Pickups by Day of the Week")
plt.xticks(ticks=range(7), labels=["Mon", "Tue", "Wed", "Thu", "Fri",
"Sat", "Sun"], rotation=0)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()
```




```
# Show the monthly trends in pickups


df["tpep_pickup_datetime"] =
pd.to_datetime(df["tpep_pickup_datetime"])


df["pickup_month"] = df["tpep_pickup_datetime"].dt.month


month_order = list(range(1, 13))


plt.figure(figsize=(10, 5))
df["pickup_month"].value_counts().reindex(month_order).plot(kind="bar"
)
plt.xlabel("Month")
plt.ylabel("Number of Pickups")
plt.title("Distribution of Taxi Pickups by Month")
plt.xticks(ticks=range(12), labels=["Jan", "Feb", "Mar", "Apr", "May",
"Jun",
                                     "Jul", "Aug", "Sep", "Oct",
"Nov", "Dec"], rotation=0)
plt.grid(axis="y", linestyle="--", alpha=0.7)
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

(df['fare_amount'] < 0.01).value_counts()

fare_amount
False    250879
True         28
Name: count, dtype: int64

df = df[df['fare_amount'] != 0]
(df['fare_amount'] < 0.01).value_counts()

fare_amount
False    250879
Name: count, dtype: int64

(df['total_amount'] < 0.01).value_counts()

total_amount
False    250879
Name: count, dtype: int64

(df['trip_distance'] < 0.01).value_counts()

trip_distance
False    248098
```

```
True       2781
Name: count, dtype: int64
```

```
df = df[df['trip_distance'] != 0]
(df['trip_distance'] < 0.01).value_counts()
```

```
trip_distance
False    248098
Name: count, dtype: int64
```

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

Yes removed the Zero values from fare_amount, total_amount, and also trip_distance (Top_Amount not removed as it is optional amount from the customer it can be zero too)

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?

```
df["PU_DO_Diff"] = df["PULocationID"] - df["DOLocationID"]


pu_do_diff_counts = df["PU_DO_Diff"].value_counts().reset_index()
pu_do_diff_counts.columns = ["PU_DO_Diff", "count"]


zero_diff_counts = pu_do_diff_counts[pu_do_diff_counts["PU_DO_Diff"]
== 0]

print(zero_diff_counts)
```

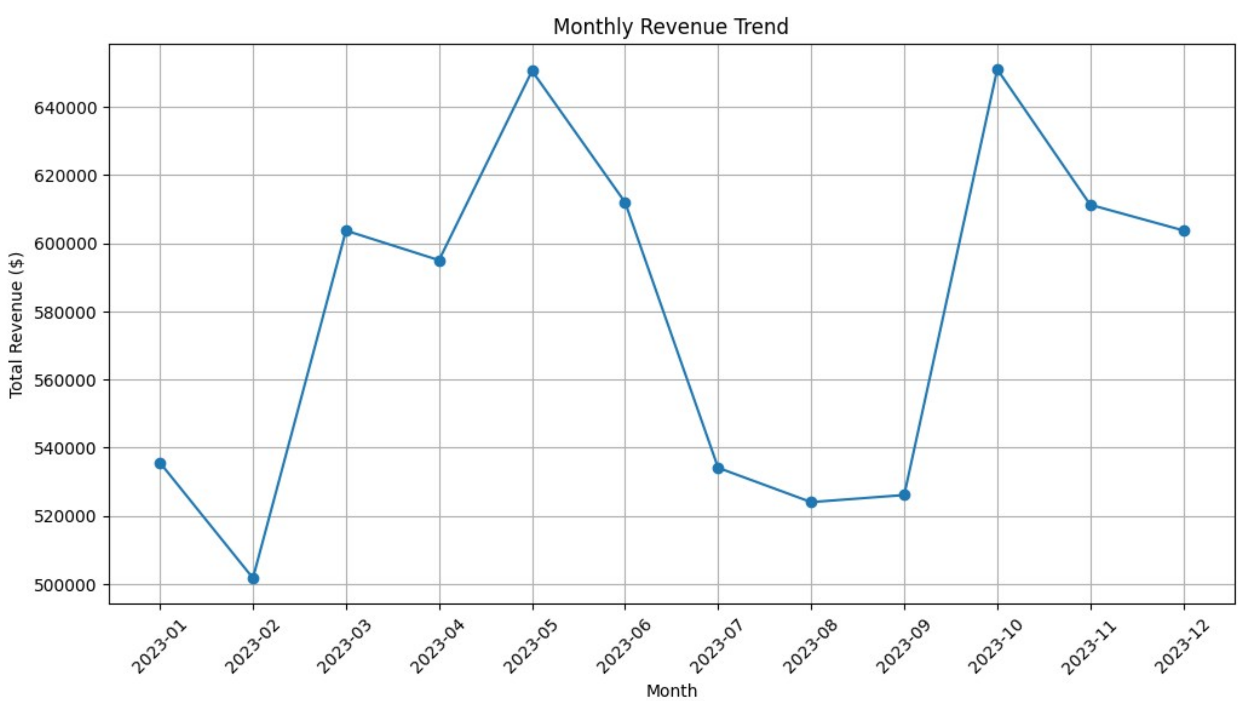
```
   PU_DO_Diff  count
0           0  11634
```

```
# Create a df with non zero entries for the selected parameters.
df = df[df["PU_DO_Diff"] != 0]

pu_do_diff_counts = df["PU_DO_Diff"].value_counts().reset_index()
pu_do_diff_counts.columns = ["PU_DO_Diff", "count"]


zero_diff_counts = pu_do_diff_counts[pu_do_diff_counts["PU_DO_Diff"]
== 0]
print(zero_diff_counts)

columns_to_remove = ["pickup_hour", "pickup_day", "pickup_month",
"PU_DO_Diff"]
```

```
Empty DataFrame
Columns: [PU_DO_Diff, count]
Index: []
```

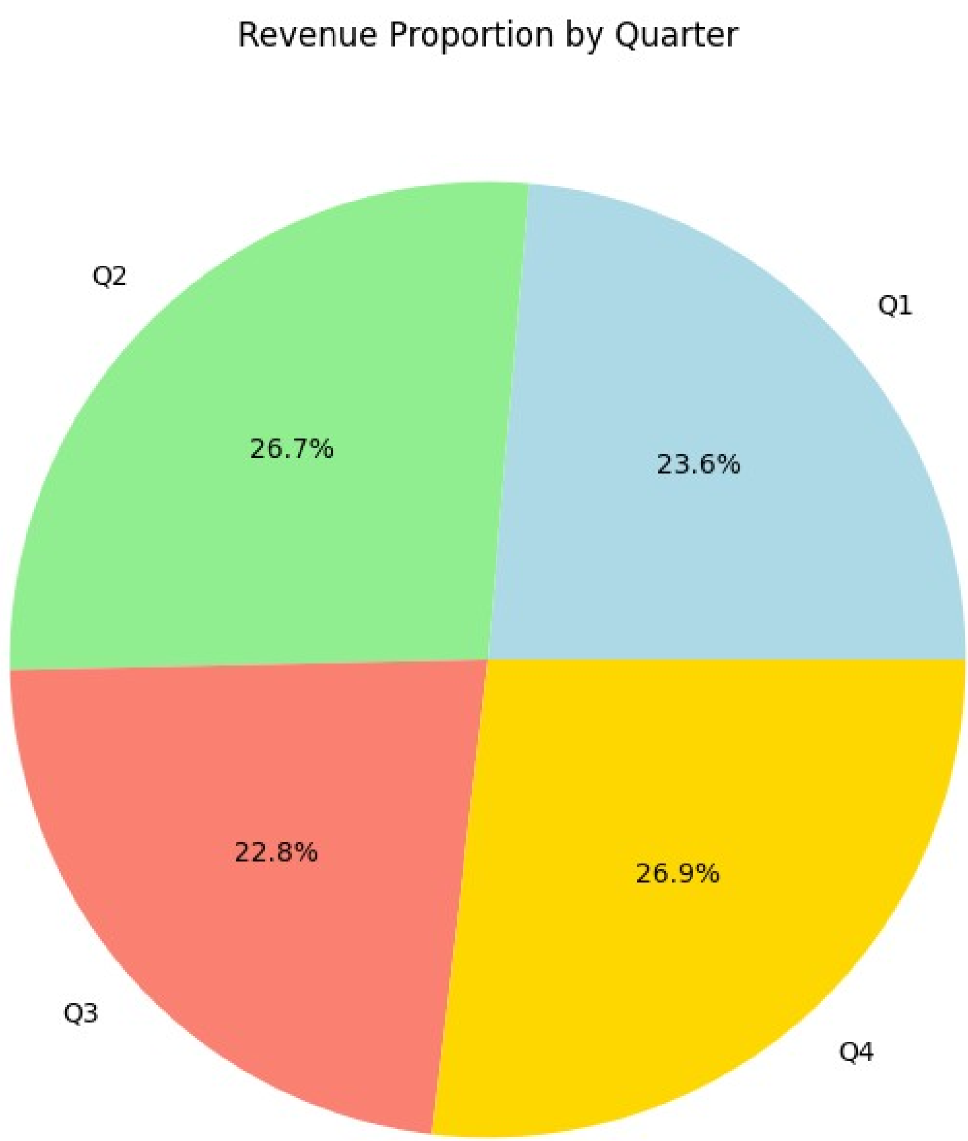
3.1.4 [3 marks] Analyse the monthly revenue (`total_amount`) trend

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

df['tpep_pickup_datetime'] =
pd.to_datetime(df['tpep_pickup_datetime'])

df['Year-Month'] = df['tpep_pickup_datetime'].dt.to_period('M')

monthly_revenue = df.groupby('Year-Month')
['total_amount'].sum().reset_index()

monthly_revenue['Year-Month'] = monthly_revenue['Year-
Month'].astype(str)

plt.figure(figsize=(12, 6))
plt.plot(monthly_revenue['Year-Month'],
monthly_revenue['total_amount'], marker='o', linestyle='-')
plt.xlabel("Month")
plt.ylabel("Total Revenue ($)")
plt.title("Monthly Revenue Trend")
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```



3.1.5 [3 marks] Show the proportion of each quarter of the year in the revenue

```
# Calculate proportion of each quarter

df["pickup_quarter"] =
pd.to_datetime(df["tpep_pickup_datetime"]).dt.quarter

quarterly_revenue = df.groupby("pickup_quarter")["total_amount"].sum()
proportion = (quarterly_revenue / quarterly_revenue.sum()) * 100

plt.figure(figsize=(8, 8))
plt.pie(proportion, labels=["Q1", "Q2", "Q3", "Q4"], autopct='%1.1f%
%', colors=["lightblue", "lightgreen", "salmon", "gold"])
plt.title("Revenue Proportion by Quarter")
plt.show()
```

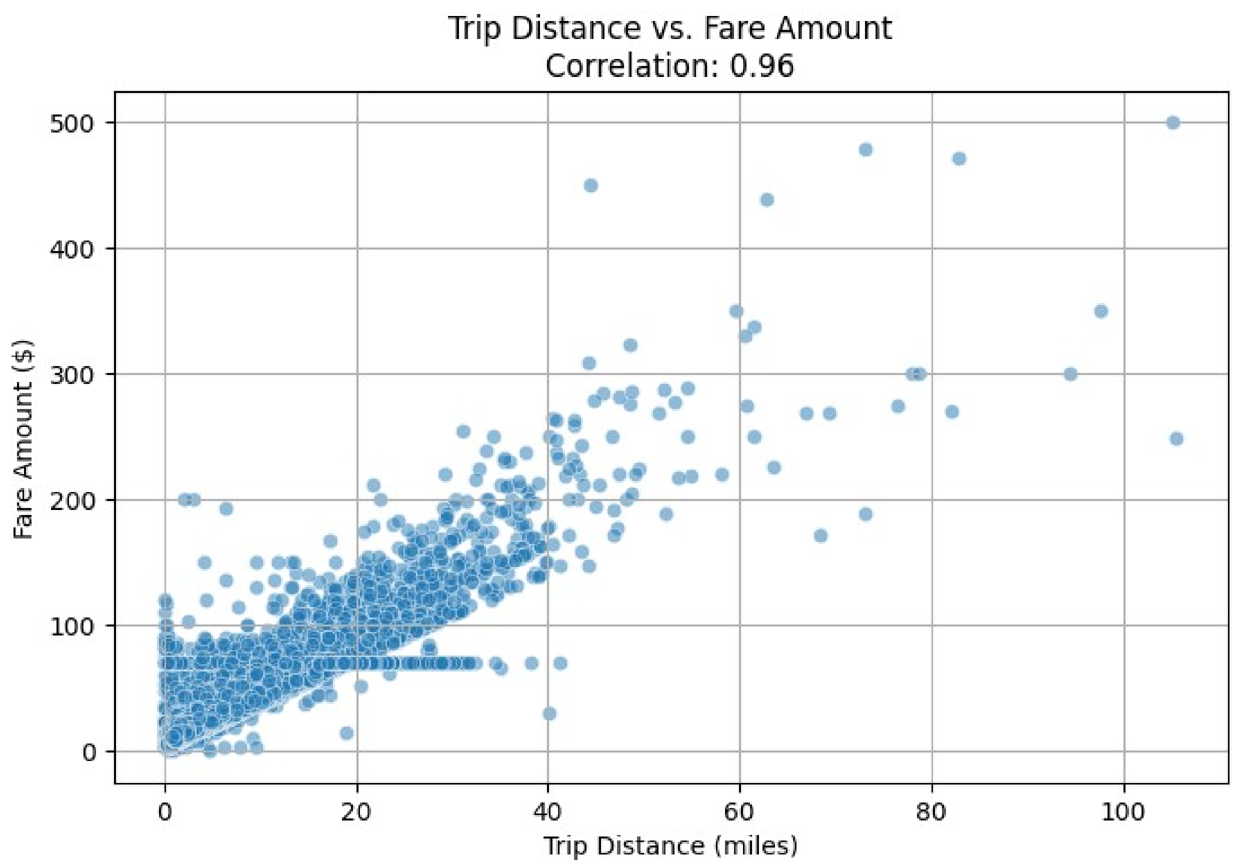


3.1.6 [3 marks] Visualise the relationship between `trip_distance` and `fare_amount`. Also find the correlation value for these two.

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

```
df = df[df["fare_amount"] <= 700]
# Show how trip fare is affected by distance


df_filtered = df[df["trip_distance"] > 0]
```

```
correlation_value =
df_filtered["trip_distance"].corr(df_filtered["fare_amount"])
print(f"Correlation between trip distance and fare amount:
{correlation_value:.2f}")

plt.figure(figsize=(8, 5))
sns.scatterplot(x=df_filtered["trip_distance"],
y=df_filtered["fare_amount"], alpha=0.5)
plt.xlabel("Trip Distance (miles)")
plt.ylabel("Fare Amount ($)")
plt.title(f"Trip Distance vs. Fare Amount\nCorrelation:
{correlation_value:.2f}")
plt.grid(True)
plt.show()

Correlation between trip distance and fare amount: 0.96
```

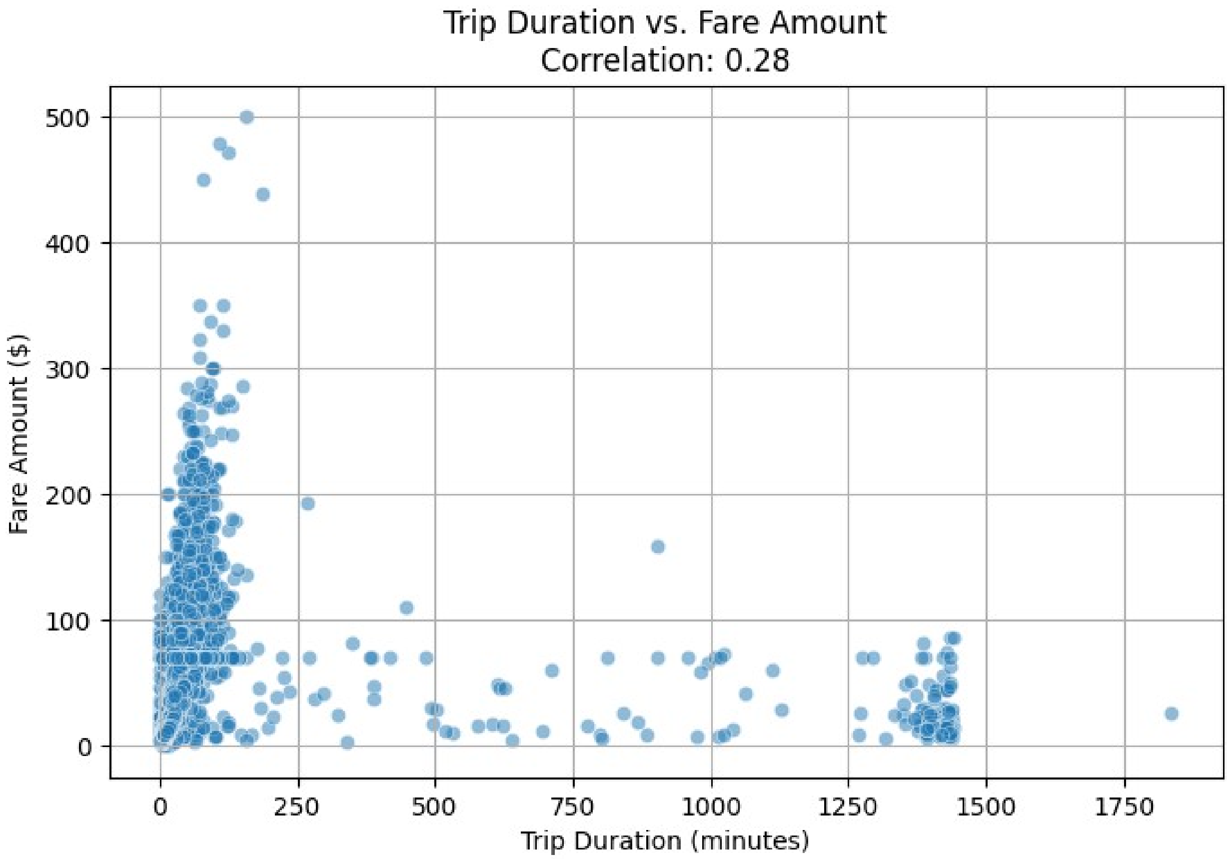


3.1.7 [5 marks] Find and visualise the correlation between:

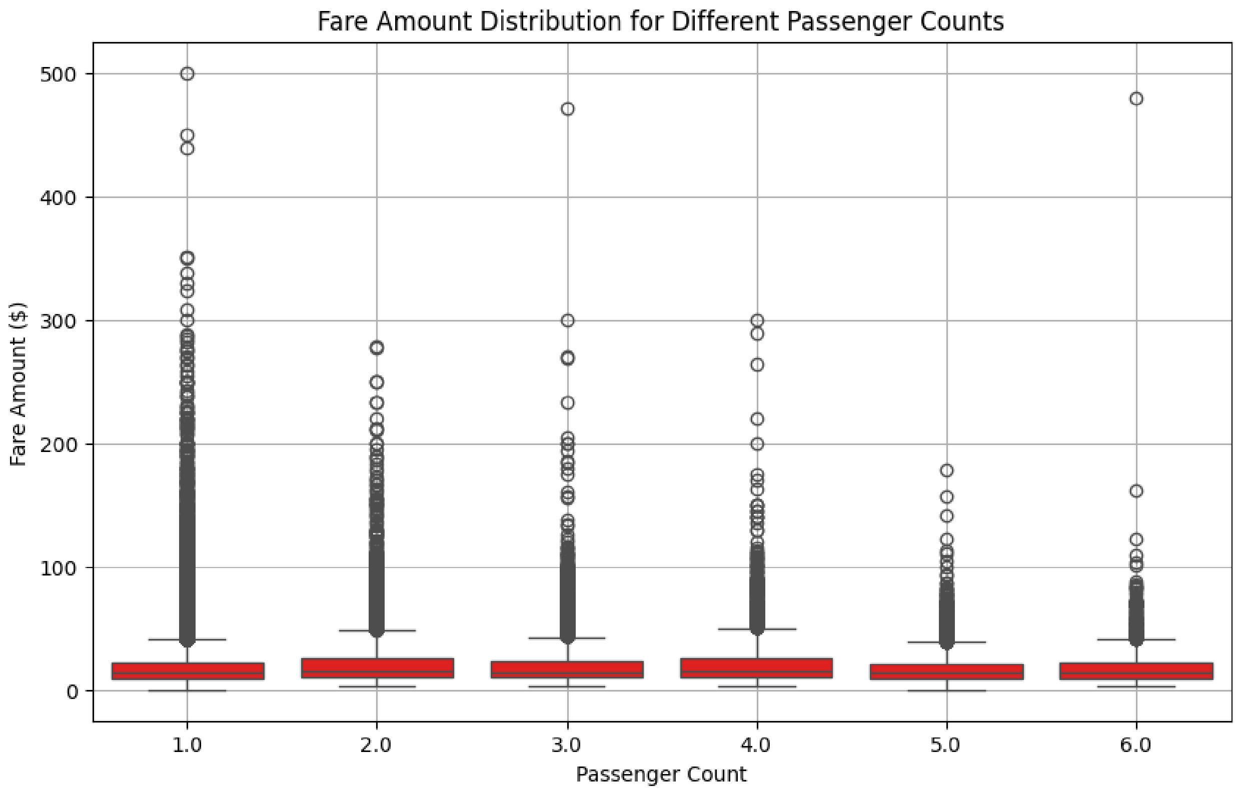
1. `fare_amount` and trip duration (pickup time to dropoff time)
2. `fare_amount` and `passenger_count`
3. `tip_amount` and `trip_distance`

```python
# Show relationship between fare and trip duration
df["tpep_pickup_datetime"] =
pd.to_datetime(df["tpep_pickup_datetime"])
df["tpep_dropoff_datetime"] =
pd.to_datetime(df["tpep_dropoff_datetime"])


df["trip_duration"] = (df["tpep_dropoff_datetime"] -
df["tpep_pickup_datetime"]).dt.total_seconds() / 60


df_filtered = df[df["trip_duration"] > 0]


correlation_value =
df_filtered["fare_amount"].corr(df_filtered["trip_duration"])
print(f"Correlation between fare amount and trip duration:
{correlation_value:.2f}")

plt.figure(figsize=(8, 5))
sns.scatterplot(x=df_filtered["trip_duration"],
y=df_filtered["fare_amount"], alpha=0.5)
plt.xlabel("Trip Duration (minutes)")
plt.ylabel("Fare Amount ($)")
plt.title(f"Trip Duration vs. Fare Amount\nCorrelation:
{correlation_value:.2f}")
plt.grid(True)
plt.show()
```

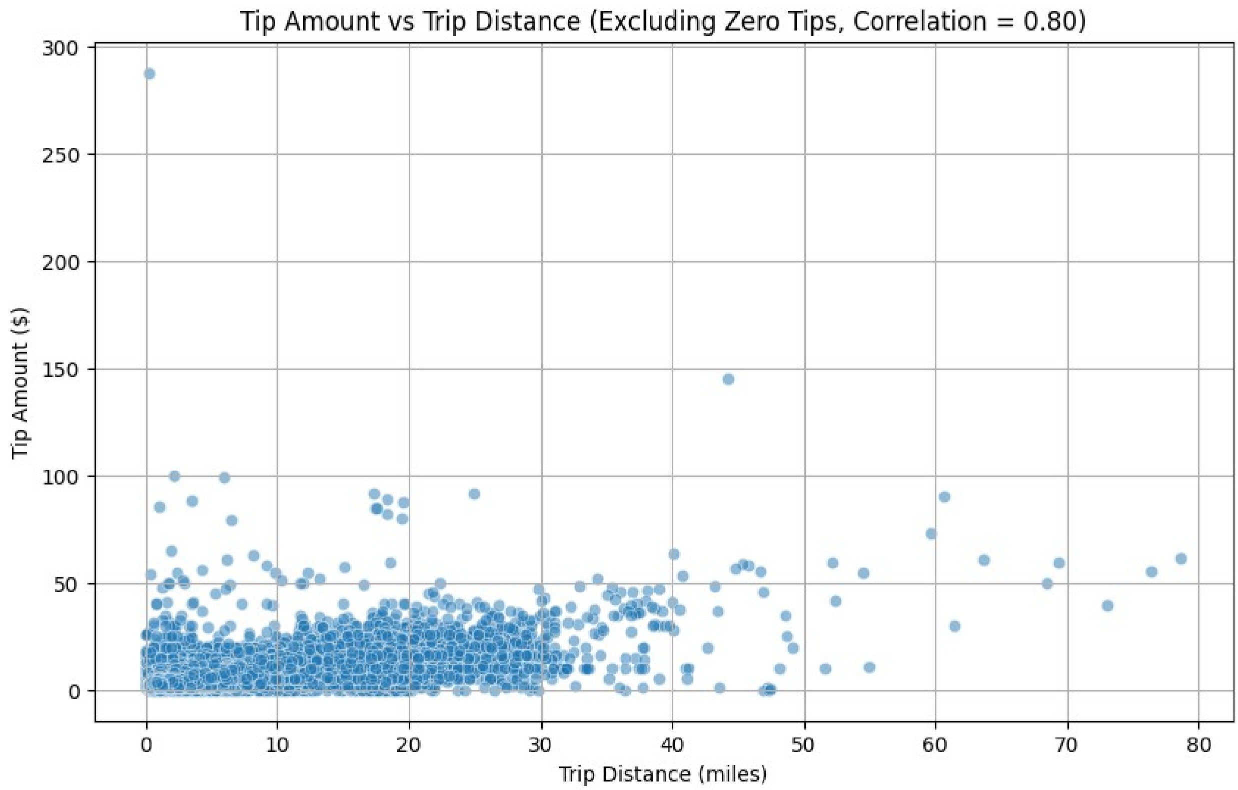
```
Correlation between fare amount and trip duration: 0.28
```




```
# Show relationship between fare and number of passengers
plt.figure(figsize=(10, 6))
sns.boxplot(x=df_filtered["passenger_count"],
y=df_filtered["fare_amount"], color='red')  # Corrected spelling
plt.xlabel("Passenger Count")
plt.ylabel("Fare Amount ($)")
plt.title("Fare Amount Distribution for Different Passenger Counts")
plt.grid(True)
plt.show()
```

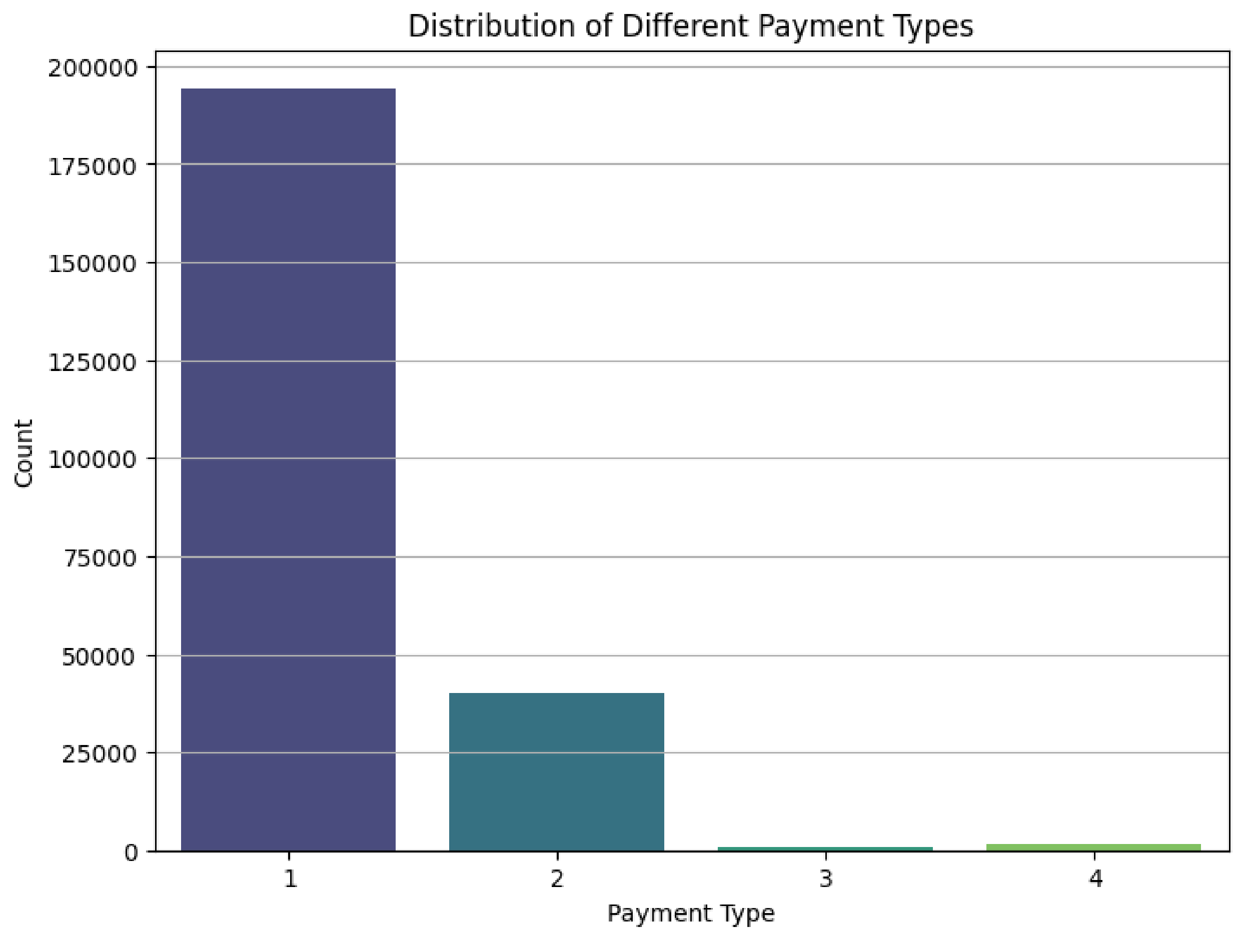



```python
# Show relationship between tip and trip distance


filtered_df = df[df['tip_amount'] > 0]


correlation_value =
filtered_df['tip_amount'].corr(filtered_df['trip_distance'])

plt.figure(figsize=(10, 6))
sns.scatterplot(x=filtered_df['trip_distance'],
y=filtered_df['tip_amount'], alpha=0.5)
plt.xlabel("Trip Distance (miles)")
plt.ylabel("Tip Amount ($)")
plt.title(f"Tip Amount vs Trip Distance (Excluding Zero Tips,
Correlation = {correlation_value:.2f})")
plt.grid(True)
plt.show()
```



3.1.8 [3 marks] Analyse the distribution of different payment types (`payment_type`)

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

payment_distribution = df['payment_type'].value_counts().reset_index()
payment_distribution.columns = ['Payment Type', 'Count']

plt.figure(figsize=(8, 6))
sns.barplot(x=payment_distribution['Payment Type'],
y=payment_distribution['Count'], palette='viridis')

plt.xlabel("Payment Type")
plt.ylabel("Count")
plt.title("Distribution of Different Payment Types")
plt.grid(axis='y')
plt.show()
```



- 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

Requirement already satisfied: geopandas in c:\users\admin\anaconda3\
lib\site-packages (1.0.1)
Requirement already satisfied: numpy>=1.22 in c:\users\admin\
anaconda3\lib\site-packages (from geopandas) (1.26.4)
Requirement already satisfied: pyogrio>=0.7.2 in c:\users\admin\
anaconda3\lib\site-packages (from geopandas) (0.10.0)
Requirement already satisfied: packaging in c:\users\admin\anaconda3\
lib\site-packages (from geopandas) (24.1)
Requirement already satisfied: pandas>=1.4.0 in c:\users\admin\
anaconda3\lib\site-packages (from geopandas) (2.2.2)
Requirement already satisfied: pyproj>=3.3.0 in c:\users\admin\
anaconda3\lib\site-packages (from geopandas) (3.7.1)
Requirement already satisfied: shapely>=2.0.0 in c:\users\admin\
anaconda3\lib\site-packages (from geopandas) (2.0.7)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\
admin\anaconda3\lib\site-packages (from pandas>=1.4.0->geopandas)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\admin\
anaconda3\lib\site-packages (from pandas>=1.4.0->geopandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\admin\
anaconda3\lib\site-packages (from pandas>=1.4.0->geopandas) (2023.3)
Requirement already satisfied: certifi in c:\users\admin\anaconda3\
lib\site-packages (from pyogrio>=0.7.2->geopandas) (2024.8.30)
Requirement already satisfied: six>=1.5 in c:\users\admin\anaconda3\
lib\site-packages (from python-dateutil>=2.8.2->pandas>=1.4.0-
>geopandas) (1.16.0)
```

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

```
# import geopandas as gpd
import geopandas as gpd
```

```
shapefile_path = r"C:\Users\ADMIN\AL ML Coursesss\taxi_zones\
taxi_zones.shp"

zones = gpd.read_file(shapefile_path)

zones.head()
```

```
   OBJECTID  Shape_Leng  Shape_Area                     zone
LocationID  \
0         1    0.116357    0.000782           Newark Airport
1
1         2    0.433470    0.004866              Jamaica Bay
2
2         3    0.084341    0.000314  Allerton/Pelham Gardens
3
3         4    0.043567    0.000112            Alphabet City
4
4         5    0.092146    0.000498            Arden Heights
5

         borough                                          geometry
0            EWR  POLYGON ((933100.918 192536.086, 933091.011 19...
1         Queens  MULTIPOLYGON (((1033269.244 172126.008, 103343...
2          Bronx  POLYGON ((1026308.77 256767.698, 1026495.593 2...
3      Manhattan  POLYGON ((992073.467 203714.076, 992068.667 20...
4  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.

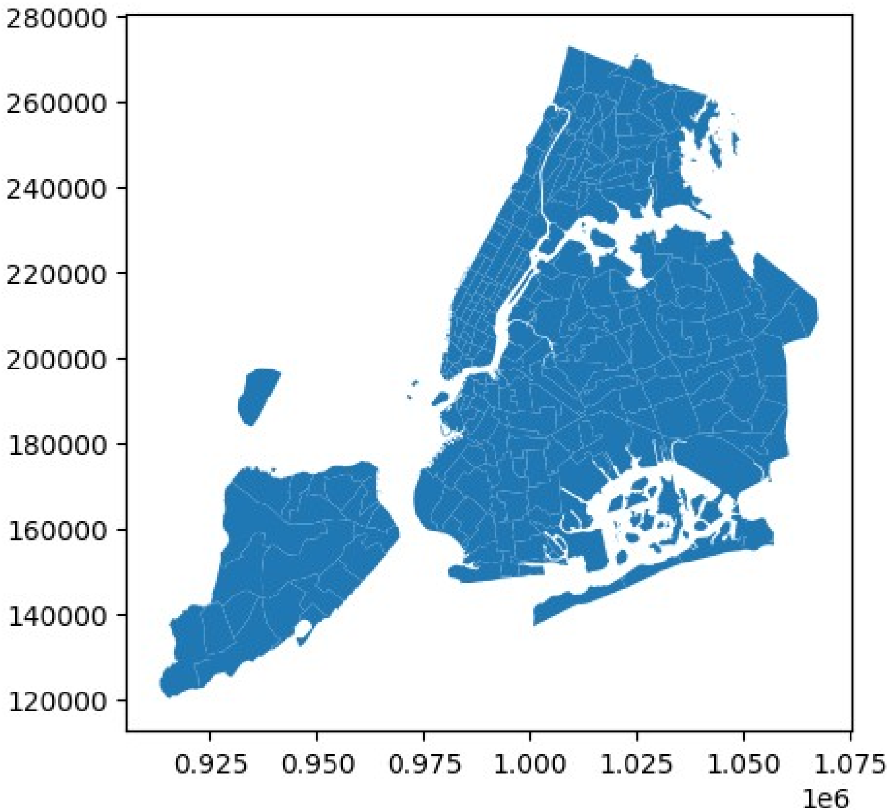
The geometric parameters like shape length, shape area and geometry are used to plot the zones on a map.

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

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

```
<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.

```
df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 236463 entries, 0 to 265499
Data columns (total 26 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   VendorID               236463 non-null  int64
 1   tpep_pickup_datetime   236463 non-null  datetime64[ns]
 2   tpep_dropoff_datetime  236463 non-null  datetime64[ns]
```

```
 3   passenger_count        236463 non-null  float64
 4   trip_distance          236463 non-null  float64
 5   RatecodeID             236463 non-null  float64
 6   store_and_fwd_flag     236463 non-null  object
 7   PULocationID           236463 non-null  int64
 8   DOLocationID           236463 non-null  int64
 9   payment_type           236463 non-null  int64
 10  fare_amount            236463 non-null  float64
 11  extra                  236463 non-null  float64
 12  mta_tax                236463 non-null  float64
 13  tip_amount             236463 non-null  float64
 14  tolls_amount           236463 non-null  float64
 15  improvement_surcharge  236463 non-null  float64
 16  total_amount           236463 non-null  float64
 17  congestion_surcharge   236463 non-null  float64
 18  Combined_Airport_Fee   236463 non-null  float64
 19  pickup_hour            236463 non-null  int32
 20  pickup_day             236463 non-null  int32
 21  pickup_month           236463 non-null  int32
 22  PU_DO_Diff             236463 non-null  int64
 23  Year-Month             236463 non-null  period[M]
 24  pickup_quarter         236463 non-null  int32
 25  trip_duration          236463 non-null  float64
dtypes: datetime64[ns](2), float64(13), int32(4), int64(5), object(1),
period[M](1)
memory usage: 45.1+ MB

#") Merge zones and trip records using locationID and PULocationID

zones["LocationID"] = zones["LocationID"].astype(int)
df["PULocationID"] = df["PULocationID"].astype(int)

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              1  2023-01-05 07:50:08   2023-01-05 08:02:04
2.0
1              2  2023-01-17 07:47:24   2023-01-17 08:00:50
5.0
2              2  2023-01-25 21:57:59   2023-01-25 22:00:33
1.0
3              2  2023-01-09 19:36:54   2023-01-09 19:52:01
2.0
4              1  2023-01-11 22:19:13   2023-01-11 22:32:37
1.0
...          ...                  ...                   ...
```

```
...
236465         1  2023-09-25 23:11:46   2023-09-25 23:19:38
1.0
236466         2  2023-09-11 07:42:12   2023-09-11 07:49:04
1.0
236467         2  2023-09-20 19:10:57   2023-09-20 19:38:54
1.0
236468         1  2023-09-30 13:48:26   2023-09-30 14:03:59
1.0
236469         2  2023-09-11 18:13:38   2023-09-11 18:27:04
1.0

        trip_distance  RatecodeID store_and_fwd_flag  PULocationID  \
0                1.90         1.0                  N           239
1                1.86         1.0                  N           239
2                0.50         1.0                  N           162
3                2.56         1.0                  N           162
4                2.80         1.0                  N           164
...               ...         ...                ...           ...
236465           1.10         1.0                  N           140
236466           1.15         1.0                  N           141
236467           9.49         1.0                  N           234
236468           1.20         1.0                  N           113
236469           0.77         1.0                  N           107

        DOLocationID  payment_type  ...  Year-Month  pickup_quarter  \
0                236             1  ...     2023-01               1
1                162             1  ...     2023-01               1
2                170             1  ...     2023-01               1
3                262             1  ...     2023-01               1
4                231             1  ...     2023-01               1
...              ...           ...  ...         ...             ...
236465           263             2  ...     2023-09               3
236466           262             1  ...     2023-09               3
236467           138             1  ...     2023-09               3
236468           211             1  ...     2023-09               3
236469           186             1  ...     2023-09               3

        trip_duration  OBJECTID  Shape_Leng  Shape_Area  \
0           11.933333     239.0    0.063626    0.000205
1           13.433333     239.0    0.063626    0.000205
2            2.566667     162.0    0.035270    0.000048
3           15.116667     162.0    0.035270    0.000048
4           13.400000     164.0    0.035772    0.000056
...               ...       ...         ...         ...
236465       7.866667     140.0    0.047584    0.000114
236466       6.866667     141.0    0.041514    0.000077
236467      27.950000     234.0    0.036072    0.000073
236468      15.550000     113.0    0.032745    0.000058
236469      13.433333     107.0    0.038041    0.000075
```

```
                           zone  LocationID    borough  \
0         Upper West Side South       239.0  Manhattan
1         Upper West Side South       239.0  Manhattan
2                  Midtown East       162.0  Manhattan
3                  Midtown East       162.0  Manhattan
4                 Midtown South       164.0  Manhattan
...                         ...         ...        ...
236465           Lenox Hill East       140.0  Manhattan
236466           Lenox Hill West       141.0  Manhattan
236467                  Union Sq       234.0  Manhattan
236468  Greenwich Village North       113.0  Manhattan
236469                  Gramercy       107.0  Manhattan

                                                 geometry
0       POLYGON ((991168.979 226252.992, 991955.565 22...
1       POLYGON ((991168.979 226252.992, 991955.565 22...
2       POLYGON ((992224.354 214415.293, 992096.999 21...
3       POLYGON ((992224.354 214415.293, 992096.999 21...
4       POLYGON ((988787.425 210315.593, 988662.868 21...
...                                                   ...
236465  POLYGON ((995735.062 215619.835, 995670.105 21...
236466  POLYGON ((994839.073 216123.698, 994786.74 216...
236467  POLYGON ((987029.847 207022.299, 987048.27 206...
236468  POLYGON ((986643.64 204346.324, 986592.535 204...
236469  POLYGON ((989131.643 205749.904, 989084.531 20...

[236470 rows x 33 columns]
```

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

trip_counts_by_location =
merged_df.groupby("zone").size().reset_index(name="Number of Trips")

print(trip_counts_by_location)
```

```
                          zone  Number of Trips
0                Alphabet City              237
1                      Astoria               80
2                 Baisley Park               71
3                   Bath Beach                2
4                 Battery Park              118
..                         ...              ...
183  Williamsburg (South Side)               39
184                   Woodside               31
185         World Trade Center             1249
186              Yorkville East             3137
187              Yorkville West             4543
```

```
[188 rows x 2 columns]
```

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
df['trip_duration']=(df['tpep_dropoff_datetime']-
df['tpep_pickup_datetime']).dt.total_seconds()/60

df_filtered = df[df["trip_duration"] > 0]

correlation_far_duration = df['fare_amount'].corr(df['trip_duration'])
correlation_far_passenger=df
['fare_amount'].corr(df['passenger_count'])

print(f"correlation_far_duration:{correlation_far_duration}")
print(f"correlation_far_passenger:{correlation_far_pass}")

# Function to safely convert WKT strings to Shapely geometries
def convert_geometry(geom):
    if isinstance(geom, str):  # Convert only if it's a string (WKT
format)
        return wkt.loads(geom)
    return geom  # Keep existing Polygon/MultiPolygon objects
unchanged


zones['geometry'] = zones['geometry'].apply(convert_geometry)

geometry_types = zones['geometry'].apply(lambda x: x.geom_type if x
else "Invalid").value_counts()
geometry_types

geometry
Polygon         240
MultiPolygon     23
Name: count, dtype: int64

from shapely.geometry import Polygon, MultiPolygon


def convert_to_polygon(geom):
    if isinstance(geom, MultiPolygon):
        return max(geom.geoms, key=lambda g: g.area)  # Keep the
largest polygon
    return geom  # Keep existing Polygons unchanged


zones['geometry'] = zones['geometry'].apply(convert_to_polygon)
```

```
zones = zones.dropna(subset=['geometry'])


gdf = gpd.GeoDataFrame(zones, geometry='geometry', crs="EPSG:4326")


geometry_types_after = gdf['geometry'].apply(lambda x: x.geom_type if
x else "Invalid").value_counts()
geometry_types_after
```

```
geometry
Polygon    263
Name: count, dtype: int64
```

```
import geopandas as gpd
gdf = gpd.GeoDataFrame(zones, geometry='geometry', crs="EPSG:4326")


gdf = gdf.merge(trip_counts_by_location, on="zone", how="left")

gdf["Number of Trips"] = gdf["Number of Trips"].fillna(0)
```

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.

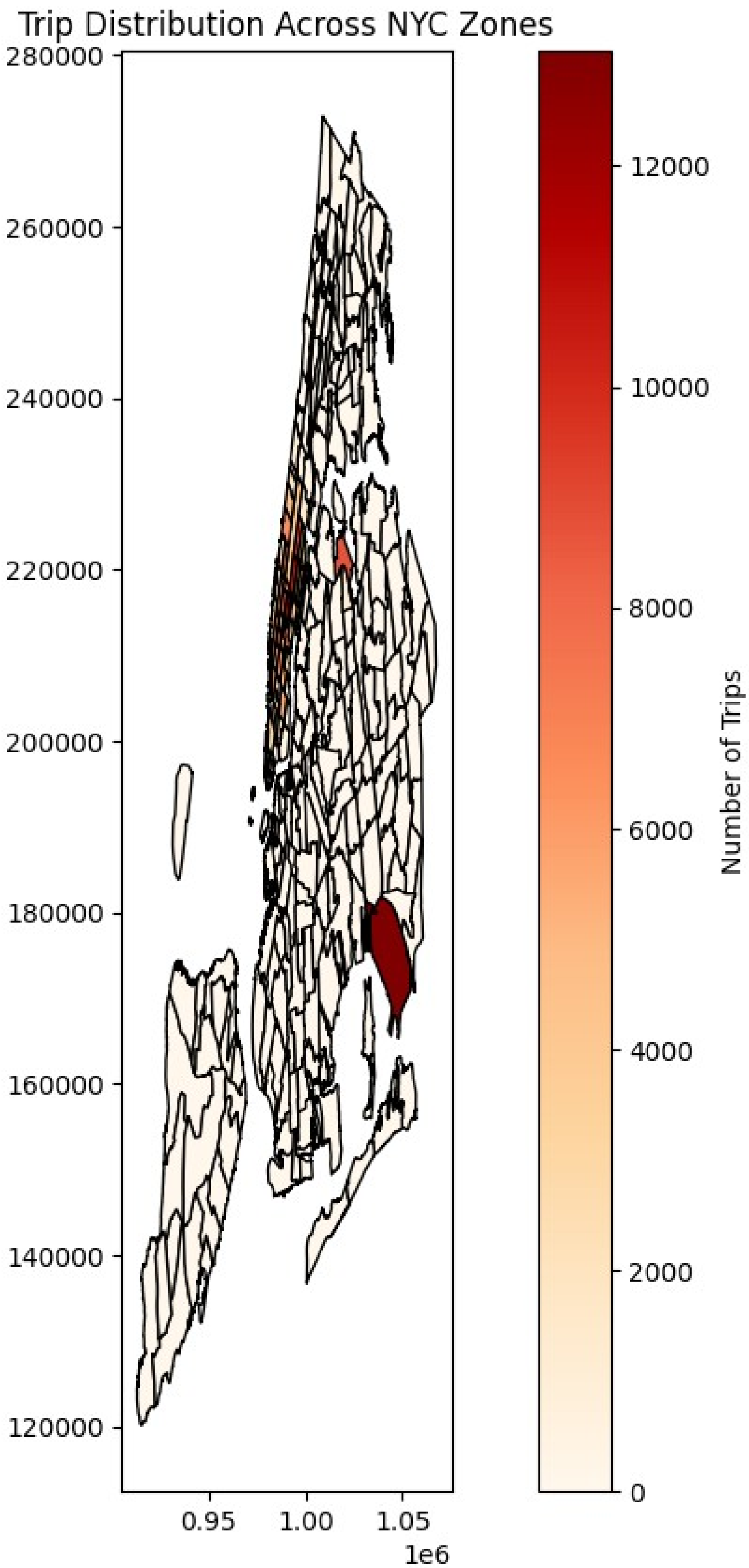
You can define the following parameters in the `zones.plot()` method:

```
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

```python
# Define figure and axis

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



gdf.plot(column="Number of Trips", ax=ax, legend=True, cmap="OrRd",
         legend_kwds={'label': "Number of Trips", 'orientation':
"vertical"},
         edgecolor="black")

ax.set_title("Trip Distribution Across NYC Zones")
plt.show()
```



Trip Distribution Across NYC Zones
280000
260000
240000
220000
200000
180000
160000
140000
120000
0.95
1.00
1.05
1e6
12000
10000
8000
6000
4000
2000
0
Number of Trips

```
# can you try displaying the zones DF sorted by the number of trips?
```

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*)

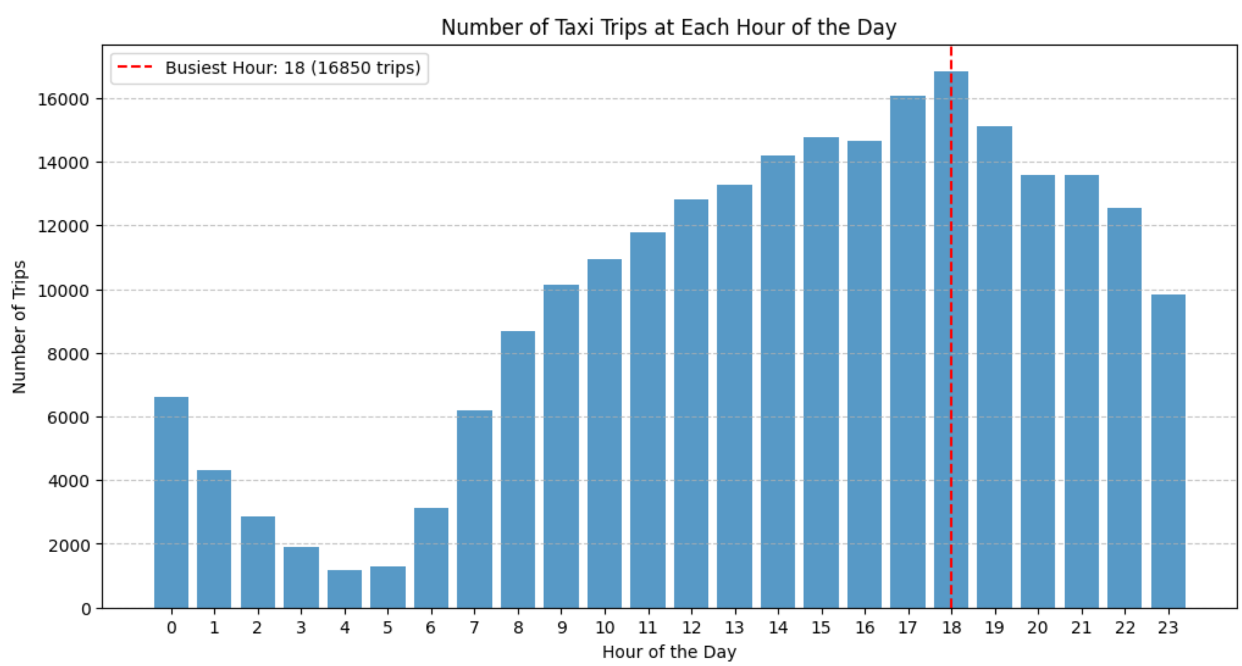
```
# Find routes which have the slowest speeds at different times of the
day
```

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

df["tpep_pickup_datetime"] =
pd.to_datetime(df["tpep_pickup_datetime"])


df["hour_of_day"] = df["tpep_pickup_datetime"].dt.hour
```

```
trips_per_hour =
df.groupby("hour_of_day").size().reset_index(name="Total_Trips")


busiest_hour =
trips_per_hour.loc[trips_per_hour["Total_Trips"].idxmax()]


plt.figure(figsize=(12, 6))
plt.bar(trips_per_hour["hour_of_day"], trips_per_hour["Total_Trips"],
alpha=0.75)
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Trips")
plt.title("Number of Taxi Trips at Each Hour of the Day")
plt.xticks(range(24))
plt.grid(axis="y", linestyle="--", alpha=0.7)


busiest_hour_value = busiest_hour["hour_of_day"]
busiest_hour_trips = busiest_hour["Total_Trips"]
plt.axvline(busiest_hour_value, color="red", linestyle="--",
label=f"Busiest Hour: {busiest_hour_value} ({busiest_hour_trips}
trips)")
plt.legend()

plt.show()
```



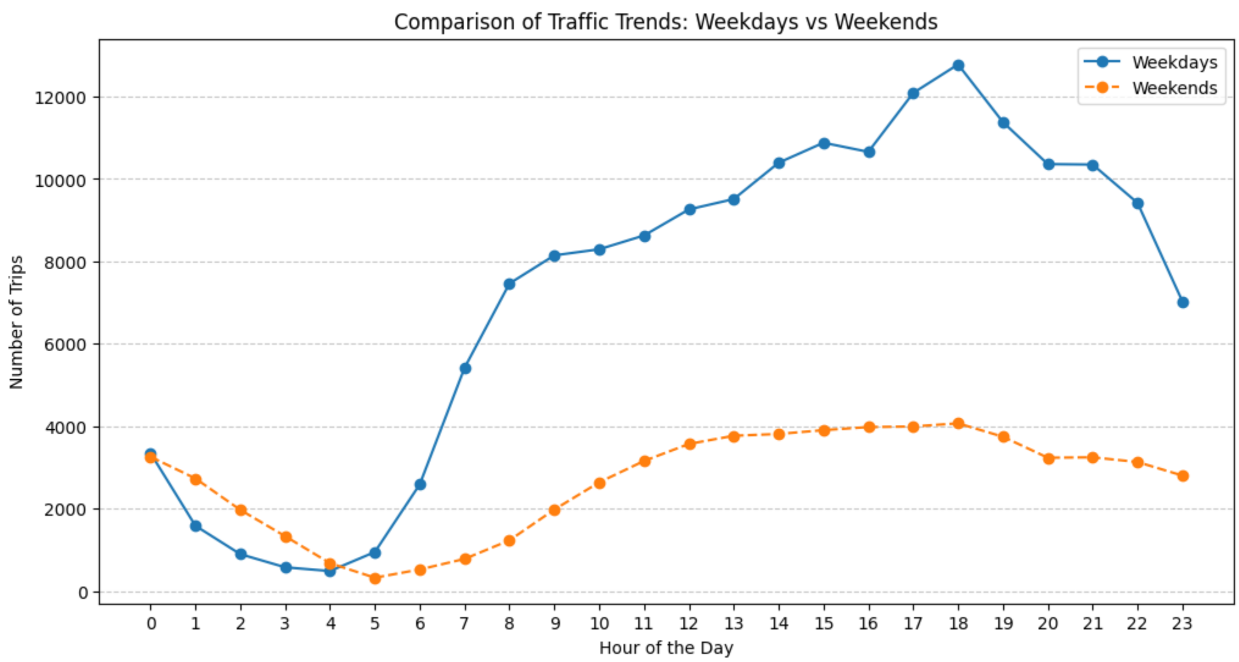
Remember, we took a fraction of trips. To find the actual number, you have to scale the number up by the sampling ratio.

3.2.3 [2 mark] Find the actual number of trips in the five busiest hours

```
# Scale up the number of trips

# Fill in the value of your sampling fraction and use that to scale up
the numbers
sample_fraction =0.1
```

3.2.4 [3 marks] Compare hourly traffic pattern on weekdays. Also compare for weekend.

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


df["day_of_week"] = df["tpep_pickup_datetime"].dt.dayofweek


df["day_type"] = df["day_of_week"].apply(lambda x: "Weekend" if x >= 5
else "Weekday")


traffic_trends = df.groupby(["day_type",
"hour_of_day"]).size().reset_index(name="Total_Trips")


plt.figure(figsize=(12, 6))


weekday_trends = traffic_trends[traffic_trends["day_type"] ==
"Weekday"]
plt.plot(weekday_trends["hour_of_day"], weekday_trends["Total_Trips"],
marker='o', label="Weekdays")


weekend_trends = traffic_trends[traffic_trends["day_type"] ==
"Weekend"]
plt.plot(weekend_trends["hour_of_day"], weekend_trends["Total_Trips"],
marker='o', linestyle="--", label="Weekends")

plt.xlabel("Hour of the Day")
plt.ylabel("Number of Trips")
plt.title("Comparison of Traffic Trends: Weekdays vs Weekends")
plt.xticks(range(24))
plt.legend()
plt.grid(axis="y", linestyle="--", alpha=0.7)


plt.show()
```



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

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

pickup_counts = df['PULocationID'].value_counts().reset_index()
pickup_counts.columns = ['LocationID', 'Pickup_Count']

dropoff_counts = df['DOLocationID'].value_counts().reset_index()
dropoff_counts.columns = ['LocationID', 'Dropoff_Count']

location_ratios = pd.merge(pickup_counts, dropoff_counts,
on='LocationID', how='outer').fillna(0)


location_ratios['Pickup_Dropoff_Ratio'] =
location_ratios['Pickup_Count'] / (location_ratios['Dropoff_Count'] +
1e-9)

top_10_ratios = location_ratios.nlargest(10, 'Pickup_Dropoff_Ratio')
bottom_10_ratios = location_ratios.nsmallest(10,
'Pickup_Dropoff_Ratio')

top_10_ratios, bottom_10_ratios
```

```
(     LocationID  Pickup_Count  Dropoff_Count  Pickup_Dropoff_Ratio
 56           59           1.0            0.0          1.000000e+09
 67           70        1173.0          126.0          9.309524e+00
 124         132       13039.0         2422.0          5.383567e+00
 130         138        8694.0         2928.0          2.969262e+00
 178         186        8634.0         5505.0          1.568392e+00
 40           43        4107.0         2868.0          1.432008e+00
 106         114        3270.0         2382.0          1.372796e+00
 240         249        5394.0         3943.0          1.367994e+00
 154         162        8748.0         6819.0          1.282886e+00
 153         161       11366.0         9455.0          1.202115e+00,
     LocationID  Pickup_Count  Dropoff_Count  Pickup_Dropoff_Ratio
 1            3           0.0           19.0                   0.0
 3            6           0.0            7.0                   0.0
 5            8           0.0            6.0                   0.0
 6            9           0.0           26.0                   0.0
 12          15           0.0           24.0                   0.0
 16          19           0.0           19.0                   0.0
 17          20           0.0           24.0                   0.0
 18          21           0.0           24.0                   0.0
 19          22           0.0           38.0                   0.0
 20          23           0.0            3.0                   0.0)
```

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


pickup_dropoff_counts = df.groupby(['PULocationID',
'DOLocationID']).size().reset_index(name='Count')


pickup_counts = df['PULocationID'].value_counts().reset_index()
pickup_counts.columns = ['PULocationID', 'Total_Pickups']

dropoff_counts = df['DOLocationID'].value_counts().reset_index()
dropoff_counts.columns = ['DOLocationID', 'Total_Dropoffs']


pickup_dropoff_ratios = pickup_dropoff_counts.merge(pickup_counts,
on='PULocationID')
pickup_dropoff_ratios = pickup_dropoff_ratios.merge(dropoff_counts,
on='DOLocationID')


pickup_dropoff_ratios['Pickup_Dropoff_Ratio'] =
pickup_dropoff_ratios['Total_Pickups'] /
(pickup_dropoff_ratios['Total_Dropoffs'] + 1e-9)
```

```python
top_10_ratios = pickup_dropoff_ratios.nlargest(10,
'Pickup_Dropoff_Ratio')
bottom_10_ratios = pickup_dropoff_ratios.nsmallest(10,
'Pickup_Dropoff_Ratio')

top_10_ratios, bottom_10_ratios
```

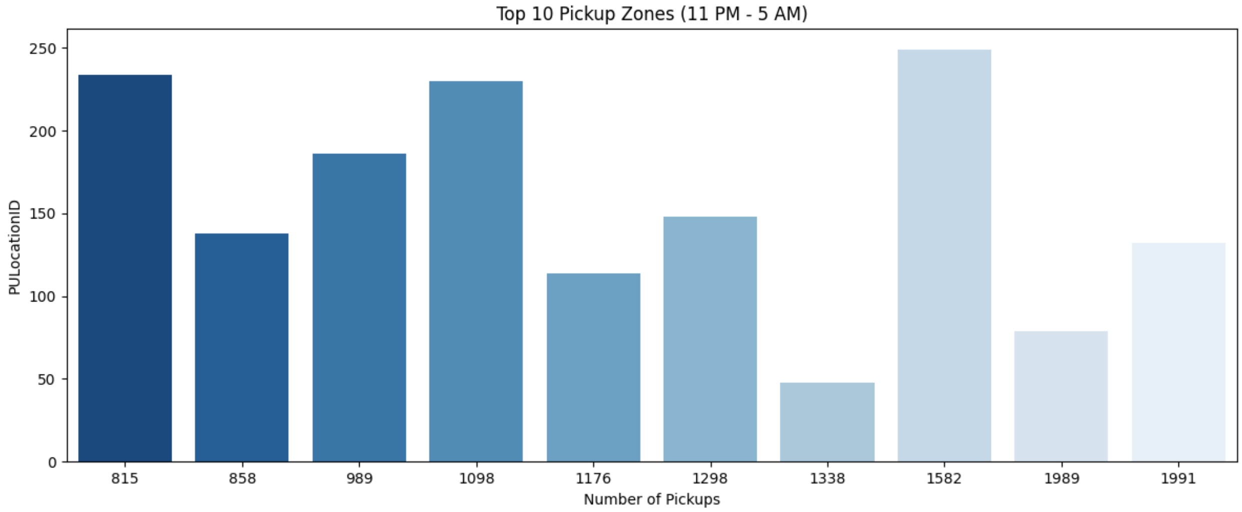
```
(      PULocationID  DOLocationID  Count  Total_Pickups
Total_Dropoffs  \
 2977           132            44      1          13039
1
 3104           132           187      1          13039
1
 3120           132           204      1          13039
1
 3539           138           253      1           8694
1
 2990           132            57      1          13039
2
 3093           132           176      1          13039
2
 3101           132           184      1          13039
2
 3366           138            57      1           8694
2
 3414           138           115      1           8694
2
 2956           132            23      3          13039
3

       Pickup_Dropoff_Ratio
 2977          13038.999987
 3104          13038.999987
 3120          13038.999987
 3539           8693.999991
 2990           6519.499997
 3093           6519.499997
 3101           6519.499997
 3366           4346.999998
 3414           4346.999998
 2956           4346.333332  ,
       PULocationID  DOLocationID  Count  Total_Pickups
Total_Dropoffs  \
 400             31           237      1              1
9462
 1059            62           234      1              1
5760
 6044           228           186      1              1
```

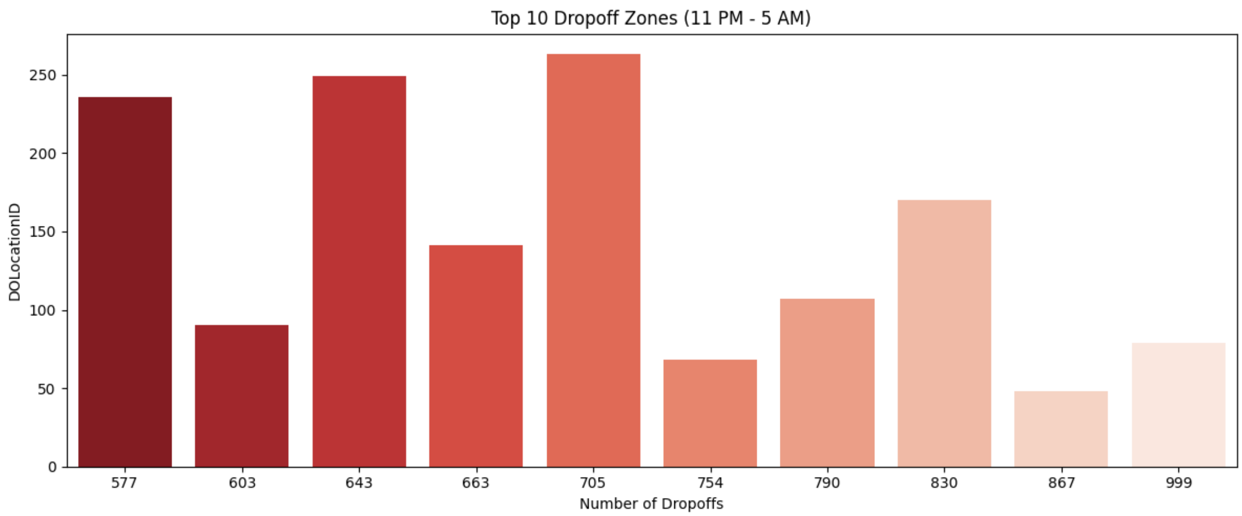
```
5505
 461             38            161      1              2
9455
 5642           200            230      1              2
7547
 5643           200            239      1              2
6579
 5351           177            141      1              2
6189
 5196           167             75      1              1
2705
 5651           207            164      1              2
5298
 446             34            229      1              2
5179

       Pickup_Dropoff_Ratio
 400               0.000106
 1059              0.000174
 6044              0.000182
 461               0.000212
 5642              0.000265
 5643              0.000304
 5351              0.000323
 5196              0.000370
 5651              0.000378
 446               0.000386  )
```

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


df['tpep_pickup_datetime'] =
pd.to_datetime(df['tpep_pickup_datetime'])


df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour

night_hours_data = df[(df['pickup_hour'] >= 23) | (df['pickup_hour']
<= 5)]


top_night_pickup_zones =
night_hours_data['PULocationID'].value_counts().head(10).reset_index()
top_night_pickup_zones.columns = ['PULocationID', 'Pickup_Count']
```

```
top_night_dropoff_zones =
night_hours_data['DOLocationID'].value_counts().head(10).reset_index()
top_night_dropoff_zones.columns = ['DOLocationID', 'Dropoff_Count']

top_night_pickup_zones, top_night_dropoff_zones


fig, axes = plt.subplots(2, 1, figsize=(12, 10))

sns.barplot(x='Pickup_Count', y='PULocationID',
data=top_night_pickup_zones, ax=axes[0], palette='Blues_r')
axes[0].set_title('Top 10 Pickup Zones (11 PM - 5 AM)')
axes[0].set_xlabel('Number of Pickups')
axes[0].set_ylabel('PULocationID')


sns.barplot(x='Dropoff_Count', y='DOLocationID',
data=top_night_dropoff_zones, ax=axes[1], palette='Reds_r')
axes[1].set_title('Top 10 Dropoff Zones (11 PM - 5 AM)')
axes[1].set_xlabel('Number of Dropoffs')
axes[1].set_ylabel('DOLocationID')

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)

night_hours_data.head()

    VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
20         2  2023-01-04 01:07:43   2023-01-04 01:27:24
1.0
41         2  2023-01-28 00:11:13   2023-01-28 00:35:17
1.0
54         1  2023-01-25 23:04:10   2023-01-25 23:37:00
2.0
60         2  2023-01-22 04:14:54   2023-01-22 04:29:23
1.0
```

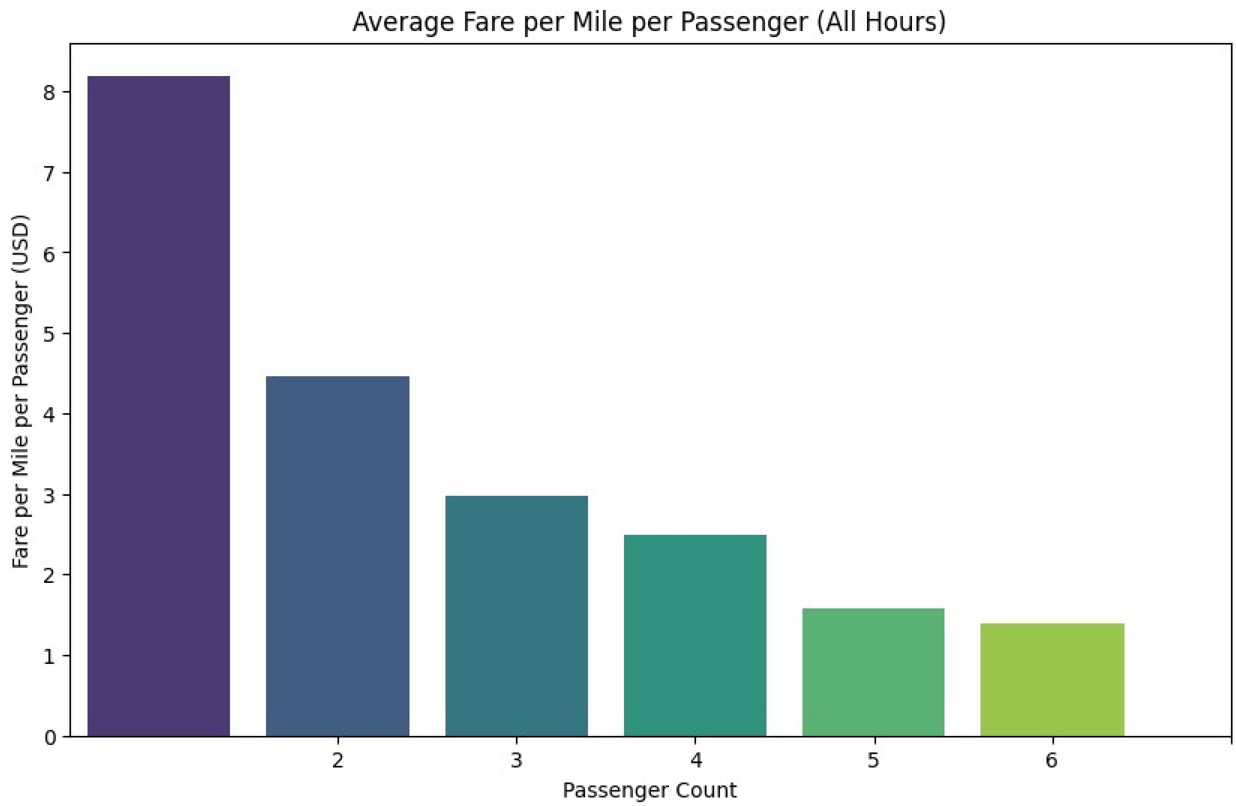
```
67         2  2023-01-30 01:30:07   2023-01-30 02:01:47
1.0

    trip_distance  RatecodeID store_and_fwd_flag  PULocationID
DOLocationID  \
20          14.90         1.0                  N           132
192
41           5.44         1.0                  N            50
232
54           8.90         1.0                  N           230
106
60           3.19         1.0                  N           232
25
67          16.02         1.0                  N            68
122

    payment_type  ...  pickup_hour  pickup_day  pickup_month
PU_DO_Diff  \
20             1  ...            1           2             1         -
60
41             1  ...            0           5             1         -
182
54             1  ...           23           2             1
124
60             2  ...            4           6             1
207
67             2  ...            1           0             1         -
54

    Year-Month  pickup_quarter  trip_duration  hour_of_day
day_of_week  \
20     2023-01               1      19.683333            1
2
41     2023-01               1      24.066667            0
5
54     2023-01               1      32.833333           23
2
60     2023-01               1      14.483333            4
6
67     2023-01               1      31.666667            1
0

    day_type
20   Weekday
41   Weekend
54   Weekday
60   Weekend
67   Weekday

[5 rows x 29 columns]
```

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.

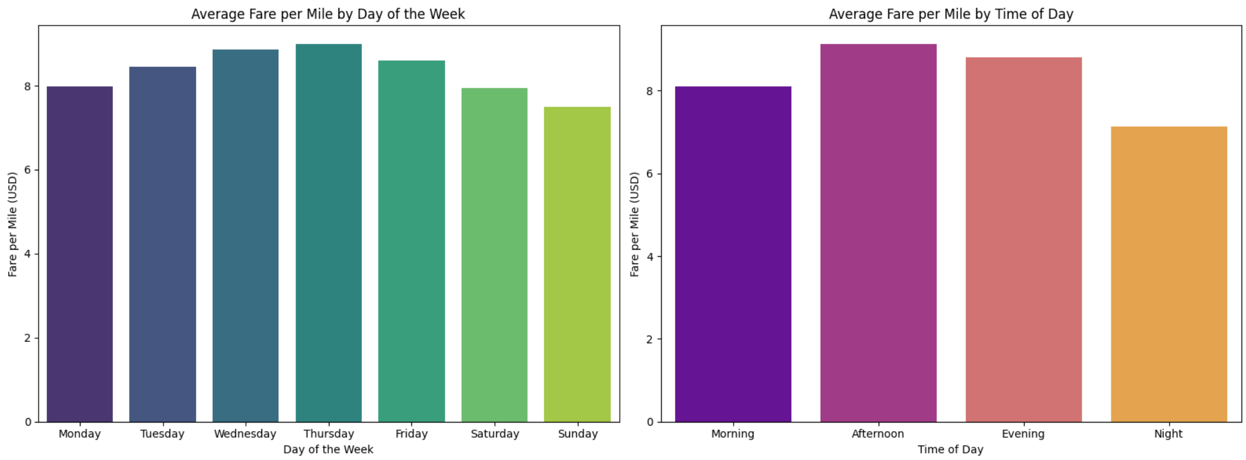
```
# Analyse the fare per mile per passenger for different passenger
counts


df = df[(df['trip_distance'] > 0) & (df['fare_amount'] > 0) &
(df['passenger_count'] > 0)]


df['fare_per_mile_per_passenger'] = df['fare_amount'] /
(df['trip_distance'] * df['passenger_count'])


avg_fare_per_mile_per_passenger['passenger_count'] =
avg_fare_per_mile_per_passenger['passenger_count'].astype(int)

plt.figure(figsize=(10, 6))
sns.barplot(x='passenger_count', y='fare_per_mile_per_passenger',
data=avg_fare_per_mile_per_passenger, palette='viridis')

plt.title('Average Fare per Mile per Passenger (All Hours)')
plt.xlabel('Passenger Count')
plt.ylabel('Fare per Mile per Passenger (USD)')
plt.xticks(ticks=range(1,
avg_fare_per_mile_per_passenger['passenger_count'].max() + 1))
plt.show()
```



3.2.10 [3 marks] Find the average fare per mile by hours of the day and by days of the week

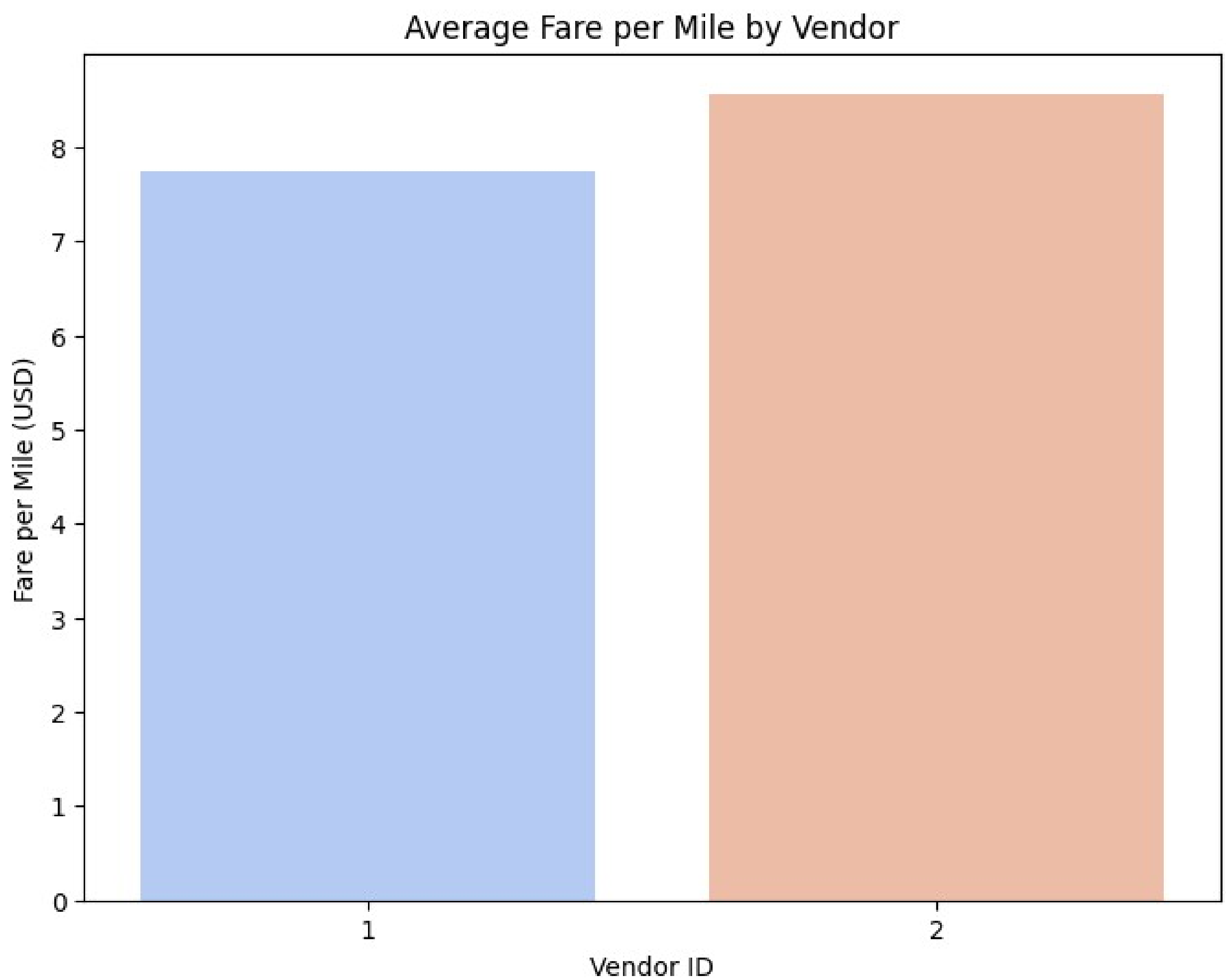
```
# Compare the average fare per mile for different days and for
different times of the day


df['tpep_pickup_datetime'] =
pd.to_datetime(df['tpep_pickup_datetime'])


df['day_of_week'] = df['tpep_pickup_datetime'].dt.day_name()


def categorize_time_of_day(hour):
    if 5 <= hour < 12:
        return 'Morning'
    elif 12 <= hour < 17:
        return 'Afternoon'
    elif 17 <= hour < 21:
        return 'Evening'
    else:
        return 'Night'

df['time_of_day'] =
df['tpep_pickup_datetime'].dt.hour.apply(categorize_time_of_day)
```

```
valid_df = df[(df['trip_distance'] > 0) & (df['fare_amount'] > 0) &
(df['passenger_count'] > 0)]


valid_df['fare_per_mile'] = valid_df['fare_amount'] /
valid_df['trip_distance']


avg_fare_per_mile_by_day = valid_df.groupby('day_of_week')
['fare_per_mile'].mean().reindex(
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']).reset_index()


avg_fare_per_mile_by_time = valid_df.groupby('time_of_day')
['fare_per_mile'].mean().reindex(
    ['Morning', 'Afternoon', 'Evening', 'Night']).reset_index()


fig, axes = plt.subplots(1, 2, figsize=(16, 6))


sns.barplot(x='day_of_week', y='fare_per_mile',
data=avg_fare_per_mile_by_day, palette='viridis', ax=axes[0])
axes[0].set_title('Average Fare per Mile by Day of the Week')
axes[0].set_xlabel('Day of the Week')
axes[0].set_ylabel('Fare per Mile (USD)')


sns.barplot(x='time_of_day', y='fare_per_mile',
data=avg_fare_per_mile_by_time, palette='plasma', ax=axes[1])
axes[1].set_title('Average Fare per Mile by Time of Day')
axes[1].set_xlabel('Time of Day')
axes[1].set_ylabel('Fare per Mile (USD)')

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

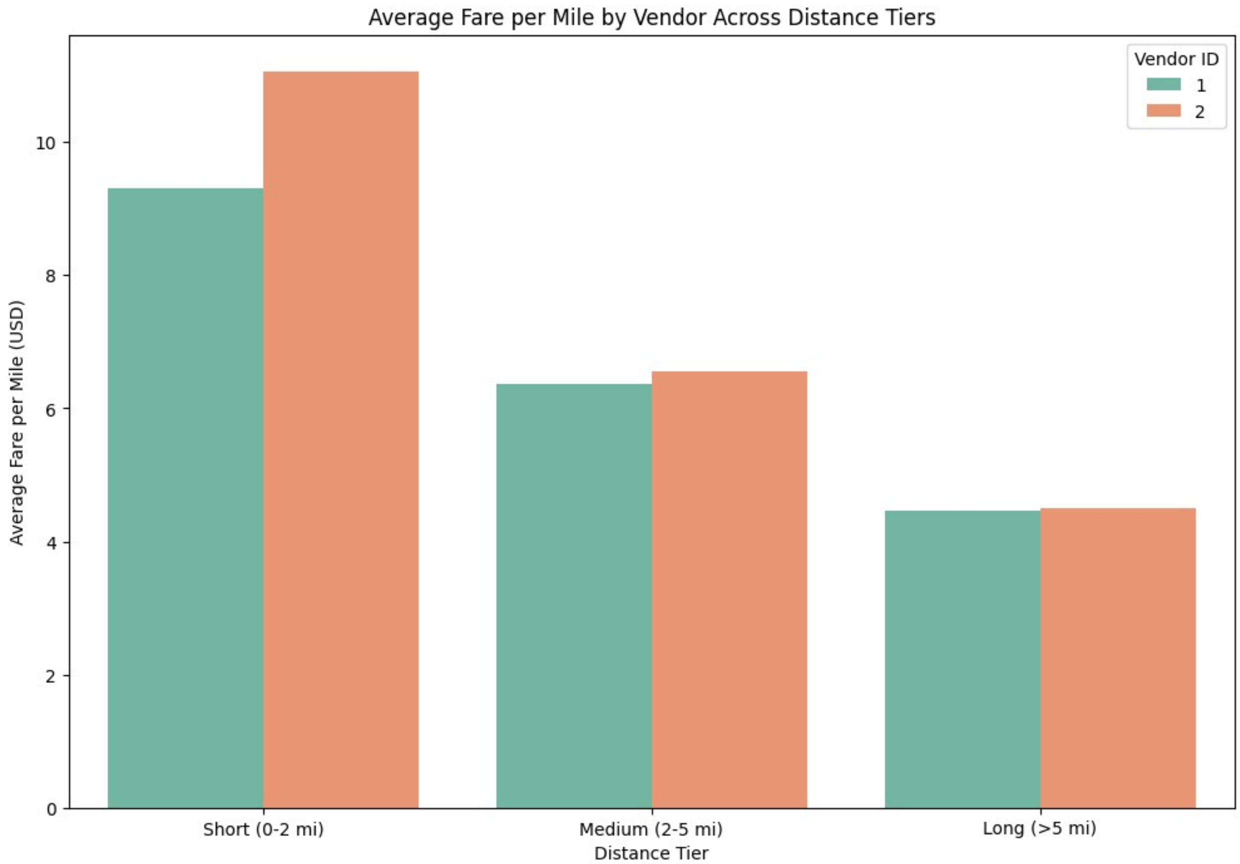
```
# Compare fare per mile for different vendors



df['tpep_pickup_datetime'] =
pd.to_datetime(df['tpep_pickup_datetime'])


valid_df = df[(df['trip_distance'] > 0) & (df['fare_amount'] > 0)]


valid_df['fare_per_mile'] = valid_df['fare_amount'] /
valid_df['trip_distance']


avg_fare_per_mile_by_vendor = valid_df.groupby('VendorID')
['fare_per_mile'].mean().reset_index()


plt.figure(figsize=(8, 6))
sns.barplot(x='VendorID', y='fare_per_mile',
data=avg_fare_per_mile_by_vendor, palette='coolwarm')

plt.title('Average Fare per Mile by Vendor')
plt.xlabel('Vendor ID')
plt.ylabel('Fare per Mile (USD)')
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



valid_df = df[(df['trip_distance'] > 0) & (df['fare_amount'] > 0)]


def distance_tier(distance):
    if distance <= 2:
        return 'Short (0-2 mi)'
    elif distance <= 5:
        return 'Medium (2-5 mi)'
    else:
        return 'Long (>5 mi)'
```

```
valid_df['distance_tier'] =
valid_df['trip_distance'].apply(distance_tier)


valid_df['fare_per_mile'] = valid_df['fare_amount'] /
valid_df['trip_distance']


avg_fare_by_vendor_tier = valid_df.groupby(['VendorID',
'distance_tier'])['fare_per_mile'].mean().reset_index()


tier_order = ['Short (0-2 mi)', 'Medium (2-5 mi)', 'Long (>5 mi)']
avg_fare_by_vendor_tier['distance_tier'] =
pd.Categorical(avg_fare_by_vendor_tier['distance_tier'],
categories=tier_order, ordered=True)


plt.figure(figsize=(12, 8))
sns.barplot(x='distance_tier', y='fare_per_mile', hue='VendorID',
data=avg_fare_by_vendor_tier, palette='Set2')

plt.title('Average Fare per Mile by Vendor Across Distance Tiers')
plt.xlabel('Distance Tier')
plt.ylabel('Average Fare per Mile (USD)')
plt.legend(title='Vendor ID')
plt.show()
```



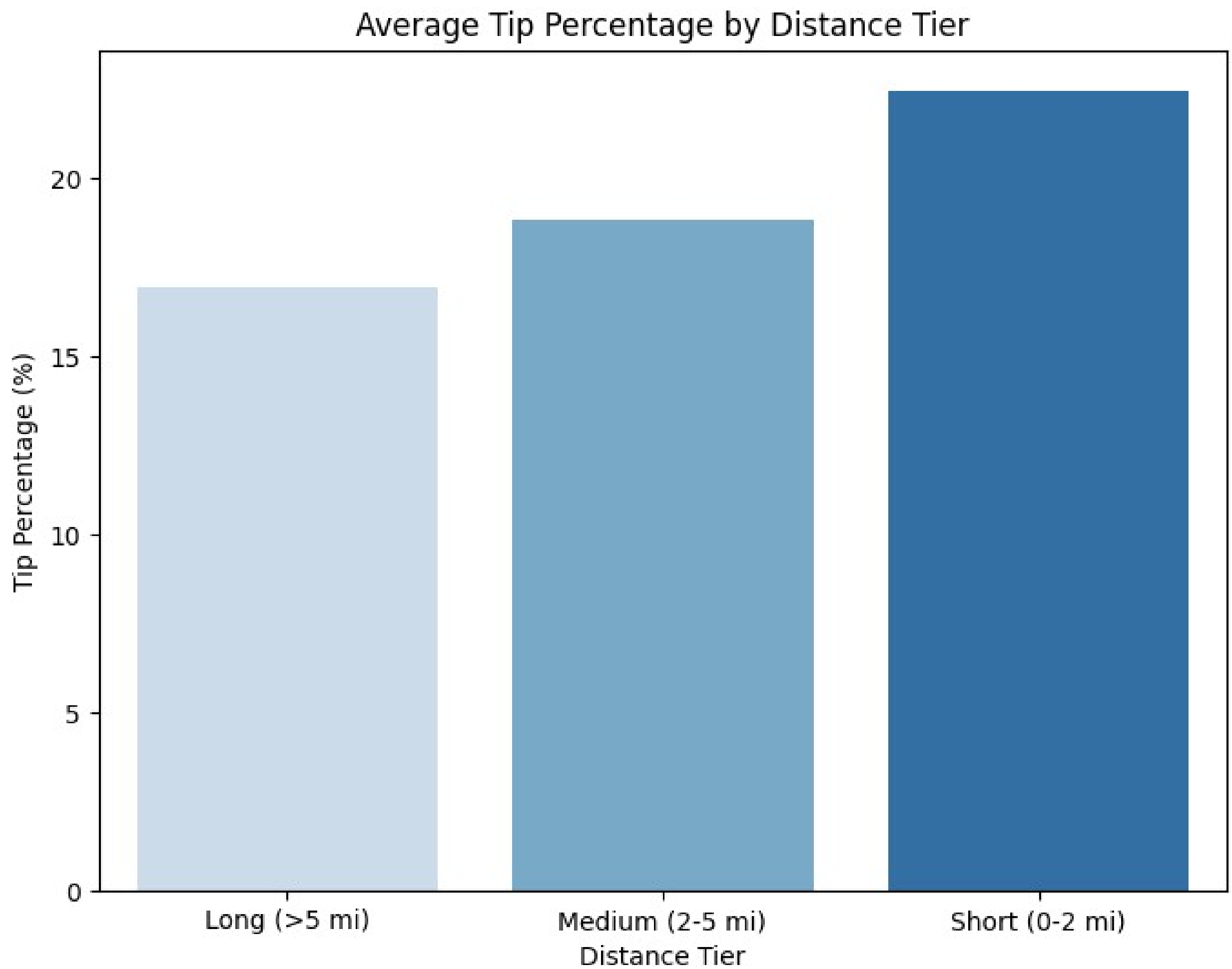
Customer Experience and Other Factors

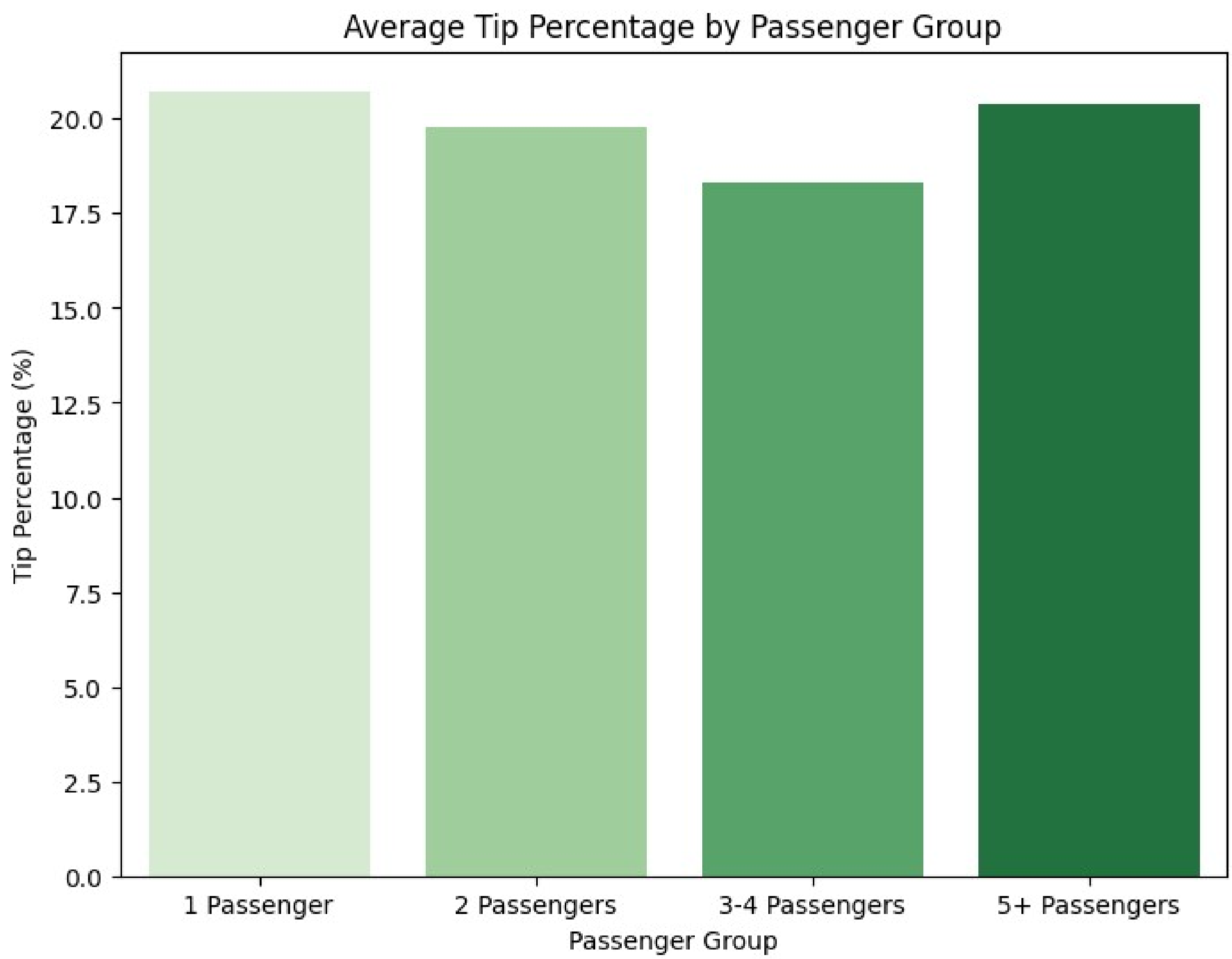
3.2.13 [5 marks] Analyse average tip percentages based on trip distances, passenger counts and time of pickup. What factors lead to low tip percentages?

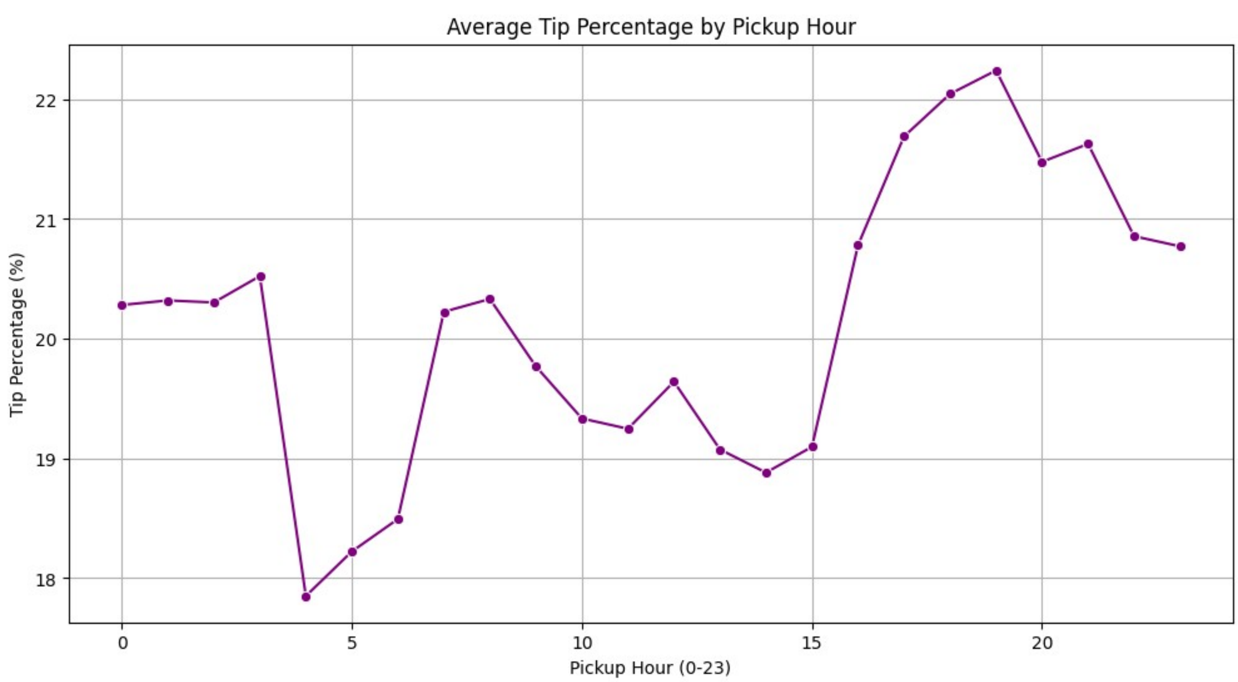
```
#  Analyze tip percentages based on distances, passenger counts and
pickup times



df['tpep_pickup_datetime'] =
pd.to_datetime(df['tpep_pickup_datetime'])


valid_df = df[(df['fare_amount'] > 0) & (df['tip_amount'] >= 0)]


valid_df['tip_percentage'] = (valid_df['tip_amount'] /
valid_df['fare_amount']) * 100

def distance_tier(distance):
    if distance <= 2:
        return 'Short (0-2 mi)'
    elif distance <= 5:
```

```
        return 'Medium (2-5 mi)'
    else:
        return 'Long (>5 mi)'

valid_df['distance_tier'] =
valid_df['trip_distance'].apply(distance_tier)


def passenger_group(count):
    if count == 1:
        return '1 Passenger'
    elif count == 2:
        return '2 Passengers'
    elif 3 <= count <= 4:
        return '3-4 Passengers'
    else:
        return '5+ Passengers'

valid_df['passenger_group'] =
valid_df['passenger_count'].apply(passenger_group)


valid_df['pickup_hour'] = valid_df['tpep_pickup_datetime'].dt.hour


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

plt.figure(figsize=(8, 6))
sns.barplot(x='distance_tier', y='tip_percentage',
data=tip_by_distance, palette='Blues')
plt.title('Average Tip Percentage by Distance Tier')
plt.xlabel('Distance Tier')
plt.ylabel('Tip Percentage (%)')
plt.show()


tip_by_passenger = valid_df.groupby('passenger_group')
['tip_percentage'].mean().reset_index()

plt.figure(figsize=(8, 6))
sns.barplot(x='passenger_group', y='tip_percentage',
data=tip_by_passenger, palette='Greens')
plt.title('Average Tip Percentage by Passenger Group')
plt.xlabel('Passenger Group')
plt.ylabel('Tip Percentage (%)')
plt.show()

tip_by_hour = valid_df.groupby('pickup_hour')
['tip_percentage'].mean().reset_index()
```

```
plt.figure(figsize=(12, 6))
sns.lineplot(x='pickup_hour', y='tip_percentage', data=tip_by_hour,
marker='o', color='purple')
plt.title('Average Tip Percentage by Pickup Hour')
plt.xlabel('Pickup Hour (0-23)')
plt.ylabel('Tip Percentage (%)')
plt.grid(True)
plt.show()
```



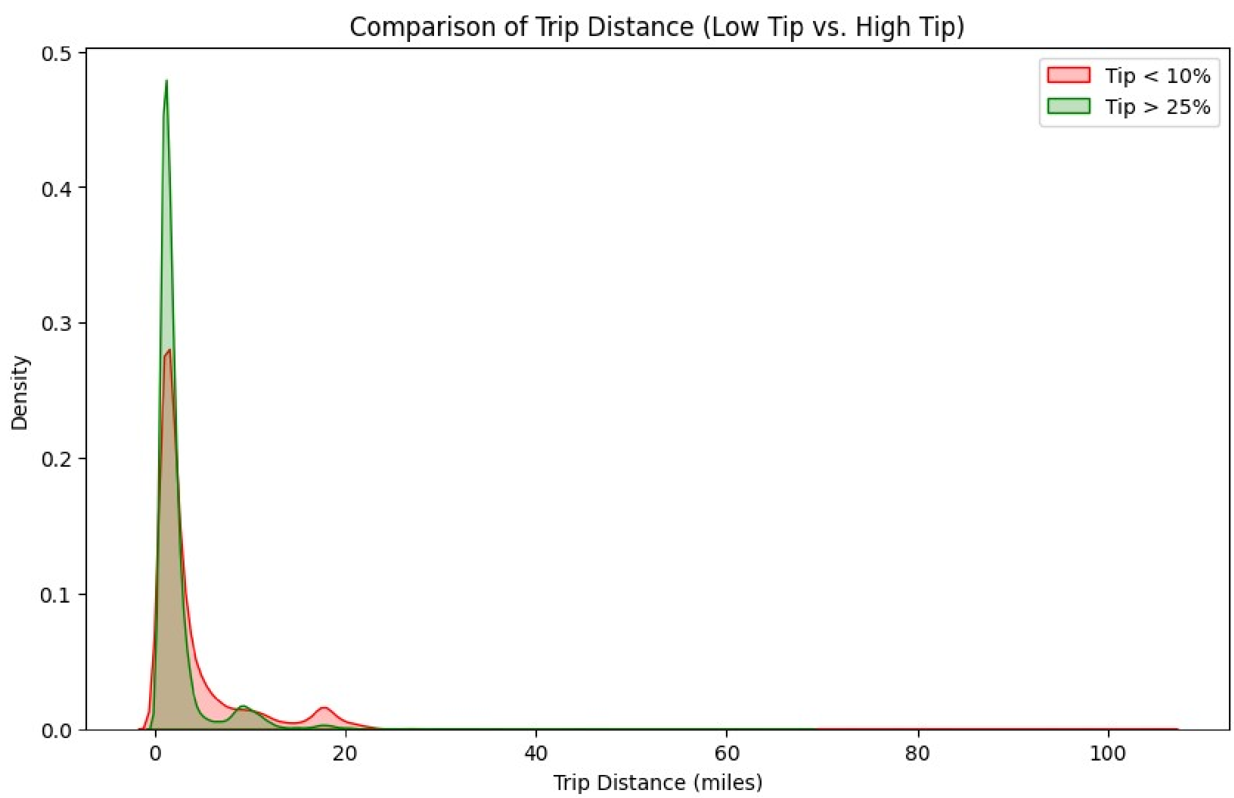

Average Tip Percentage by Passenger Group
Tip Percentage (%)
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
1 Passenger
2 Passengers
3-4 Passengers
5+ Passengers
Passenger Group

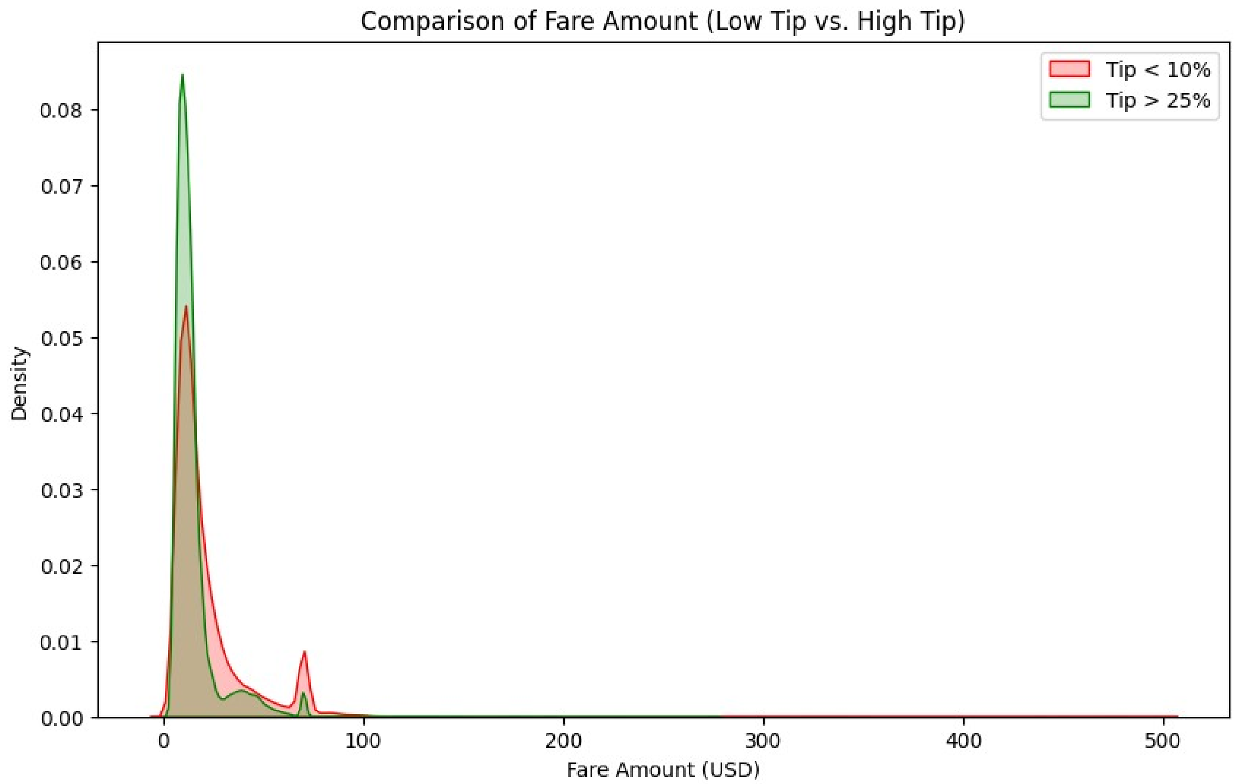

Average Tip Percentage by Pickup Hour
Tip Percentage (%)
18
19
20
21
22
0
5
10
15
20
Pickup Hour (0-23)

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

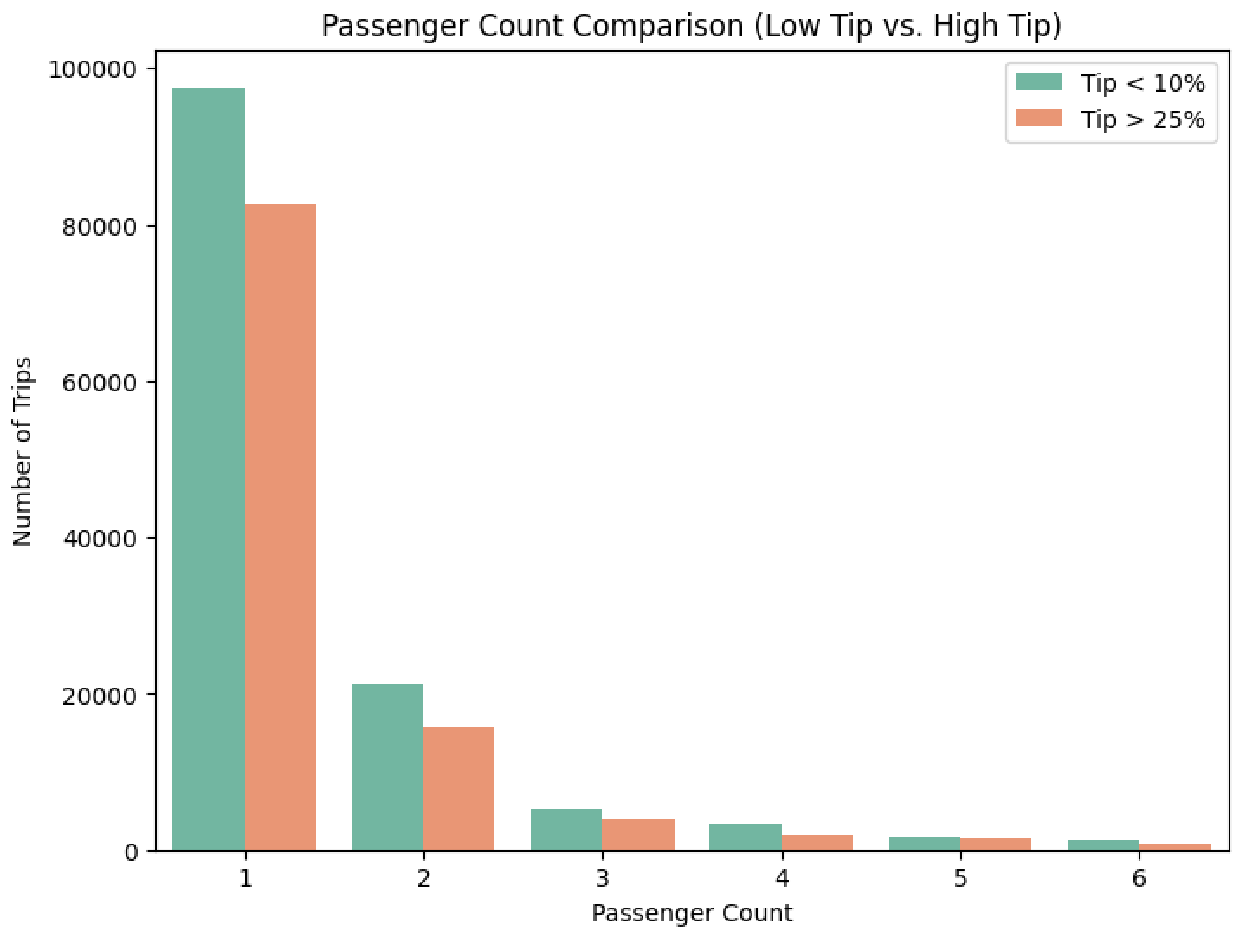
```
# Compare trips with tip percentage < 10% to trips with tip percentage
> 25%

df['tpep_pickup_datetime'] =
pd.to_datetime(df['tpep_pickup_datetime'])


valid_df = df[(df['fare_amount'] > 0) & (df['tip_amount'] >= 0)]


valid_df['tip_percentage'] = (valid_df['tip_amount'] /
valid_df['fare_amount']) * 100


low_tip_df = valid_df[valid_df['tip_percentage'] < 10]
high_tip_df = valid_df[valid_df['tip_percentage'] > 25]


def compare_distributions(low_df, high_df, feature, xlabel, title):
    plt.figure(figsize=(10, 6))
    sns.kdeplot(low_df[feature], fill=True, color='red', label='Tip <
10%')
    sns.kdeplot(high_df[feature], fill=True, color='green', label='Tip
> 25%')
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel('Density')
    plt.legend()
    plt.show()


compare_distributions(low_tip_df, high_tip_df, 'trip_distance',
                      'Trip Distance (miles)', 'Comparison of Trip
Distance (Low Tip vs. High Tip)')


compare_distributions(low_tip_df, high_tip_df, 'fare_amount',
                      'Fare Amount (USD)', 'Comparison of Fare Amount
(Low Tip vs. High Tip)')


plt.figure(figsize=(8, 6))
sns.countplot(x='passenger_count', hue=(valid_df['tip_percentage'] >
25), data=valid_df, palette='Set2')
plt.title('Passenger Count Comparison (Low Tip vs. High Tip)')
plt.xlabel('Passenger Count')
plt.ylabel('Number of Trips')
plt.legend(['Tip < 10%', 'Tip > 25%'])
```

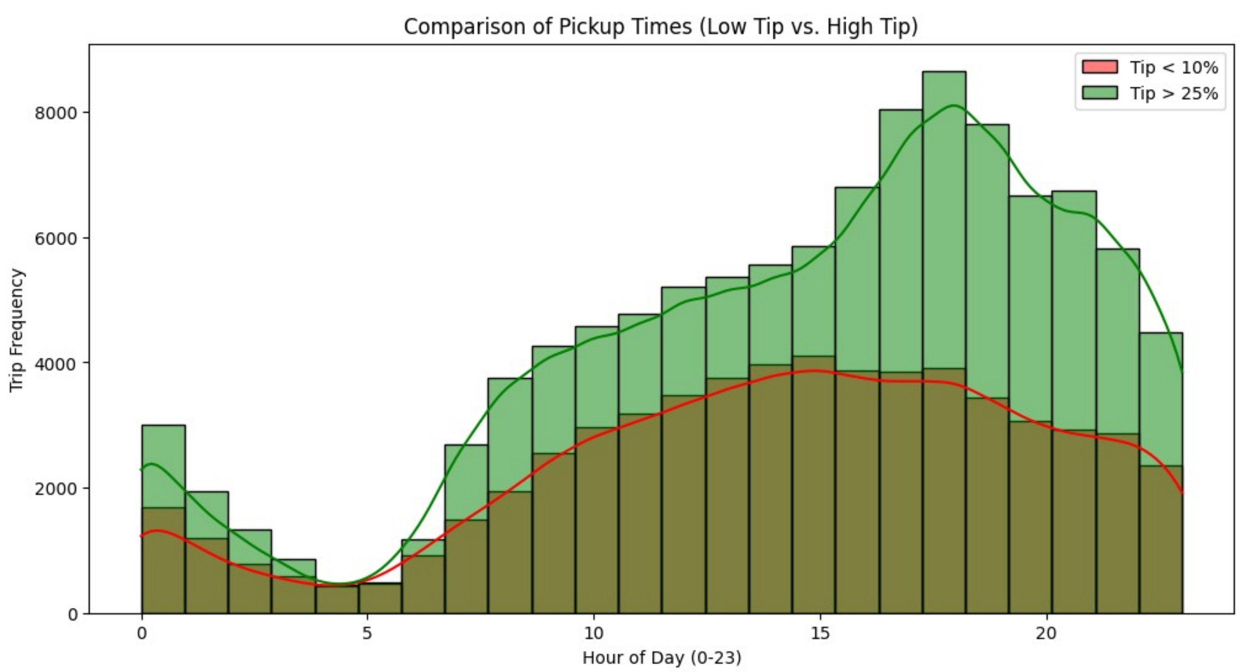
```
plt.show()


low_tip_df['pickup_hour'] = low_tip_df['tpep_pickup_datetime'].dt.hour
high_tip_df['pickup_hour'] =
high_tip_df['tpep_pickup_datetime'].dt.hour

plt.figure(figsize=(12, 6))
sns.histplot(low_tip_df['pickup_hour'], kde=True, color='red',
label='Tip < 10%', bins=24)
sns.histplot(high_tip_df['pickup_hour'], kde=True, color='green',
label='Tip > 25%', bins=24)
plt.title('Comparison of Pickup Times (Low Tip vs. High Tip)')
plt.xlabel('Hour of Day (0-23)')
plt.ylabel('Trip Frequency')
plt.legend()
plt.show()
```





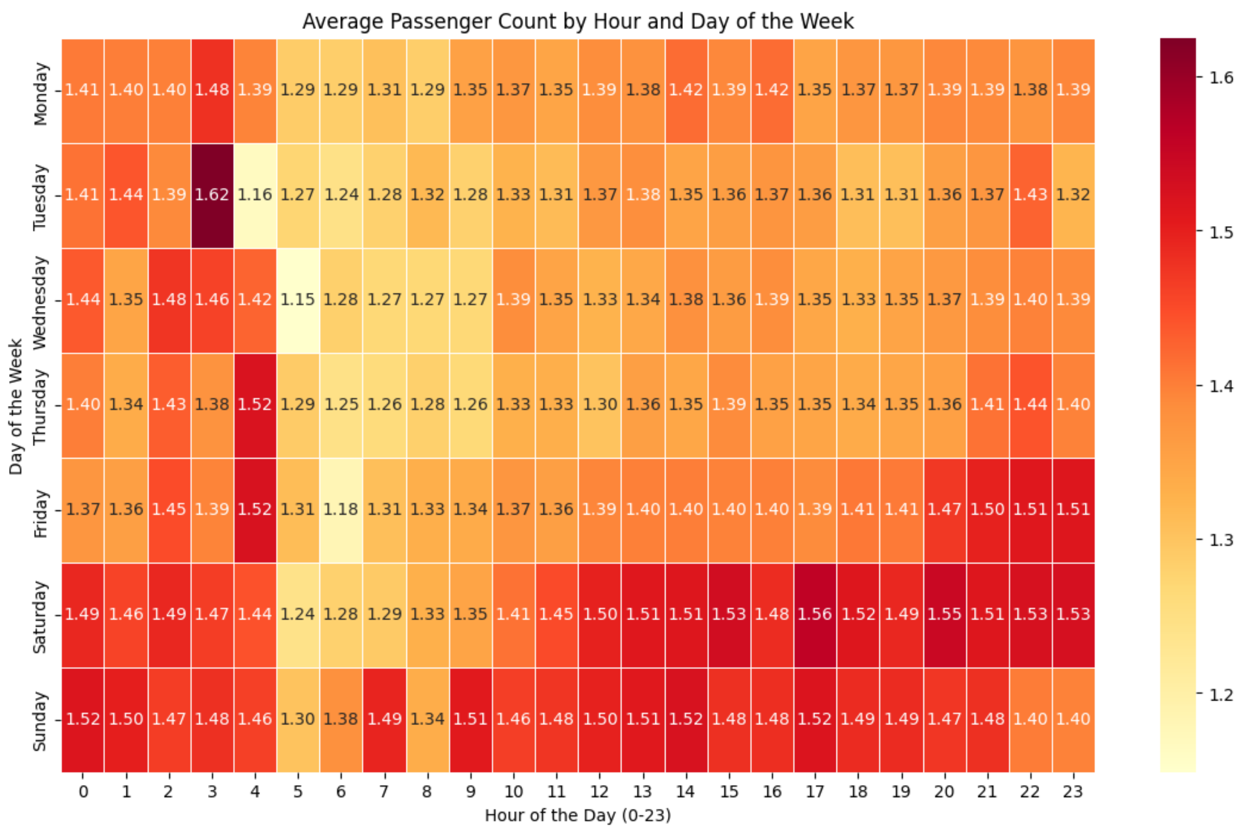
Comparison of Fare Amount (Low Tip vs. High Tip)
Tip < 10%
Tip > 25%
Density
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
0
100
200
300
400
500
Fare Amount (USD)





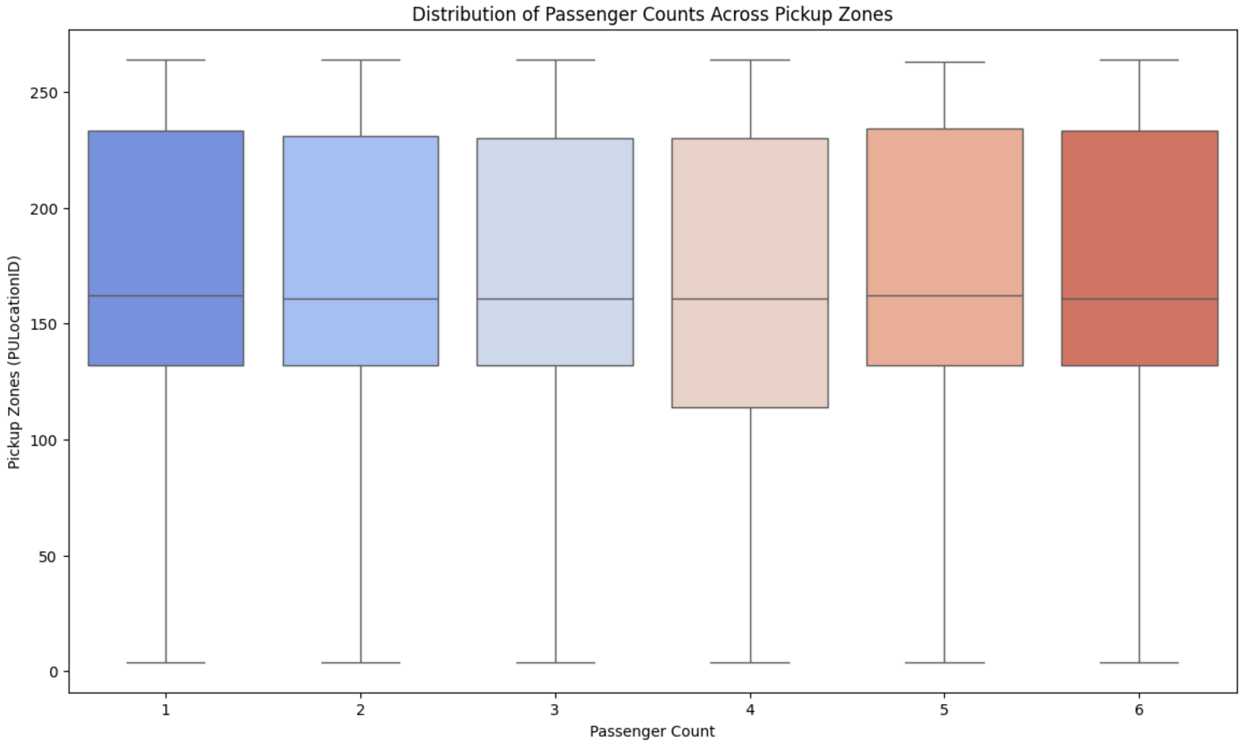
3.2.14 [3 marks] Analyse the variation of passenger count across hours and days of the week.

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



df['tpep_pickup_datetime'] =
pd.to_datetime(df['tpep_pickup_datetime'])


df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour
df['pickup_day'] = df['tpep_pickup_datetime'].dt.dayofweek


day_map = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday',
4: 'Friday', 5: 'Saturday', 6: 'Sunday'}
df['pickup_day_name'] = df['pickup_day'].map(day_map)


pivot_table = df.pivot_table(index='pickup_day_name',
columns='pickup_hour', values='passenger_count', aggfunc='mean')


pivot_table = pivot_table.loc[list(day_map.values())]


plt.figure(figsize=(14, 8))
sns.heatmap(pivot_table, cmap='YlOrRd', annot=True, fmt=".2f",
linewidths=0.5)
plt.title('Average Passenger Count by Hour and Day of the Week')
plt.xlabel('Hour of the Day (0-23)')
plt.ylabel('Day of the Week')
plt.show()
```



3.2.15 [2 marks] Analyse the variation of passenger counts across zones

```
# How does passenger count vary across zones



zone_counts = df['PULocationID'].value_counts()
top_zones = zone_counts[zone_counts > 50].index  # Filter zones with
more than 50 trips
filtered_df = df[df['PULocationID'].isin(top_zones)]


plt.figure(figsize=(14, 8))


sns.boxplot(x='passenger_count', y='PULocationID', data=filtered_df,
palette='coolwarm', showfliers=False)

plt.title('Distribution of Passenger Counts Across Pickup Zones')
plt.xlabel('Passenger Count')
plt.ylabel('Pickup Zones (PULocationID)')

plt.show()
```




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

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?

surcharge_columns = ['extra', 'mta_tax', 'tip_amount', 'tolls_amount',

                     'improvement_surcharge', 'congestion_surcharge',
'Combined_Airport_Fee']

surcharge_counts = (df[surcharge_columns] != 0).sum().reset_index()
surcharge_counts.columns = ['Surcharge', 'Frequency']

print(surcharge_counts)
```

```
               Surcharge  Frequency
0                  extra     146570
1                mta_tax     235047
```

```
2             tip_amount     186951
3           tolls_amount      19980
4  improvement_surcharge     236457
5   congestion_surcharge     221655
6   Combined_Airport_Fee      21824
```

4 Conclusion

[15 marks]

4.1 Final Insights and Recommendations

[15 marks]

Conclude your analyses here. Include all the outcomes you found based on the analysis.

Based on the insights, frame a concluding story explaining suitable parameters such as location, time of the day, day of the week etc. to be kept in mind while devising a strategy to meet customer demand and optimise supply.

4.1.1 [5 marks] Recommendations to optimize routing and dispatching based on demand patterns and operational inefficiencies

```
optimization_strategy = """
Dynamic Dispatching:
    - Prioritize high-demand zones during peak hours (morning rush,
evening, night surge).
    - Use real-time monitoring to adjust vehicle allocation.

Geofencing & Supply Management:
    - Set up geofences around busy areas (airports, business hubs).
    - Move idle vehicles to under-served regions.

Route Optimization:
    - Use algorithms for shortest, most efficient routes.
    - Minimize empty return trips by smart fleet positioning.

Driver Incentives:
    - Offer bonuses for low-demand areas and off-peak hours.
    - Reward long-distance trips to ensure broader coverage.
"""

print(optimization_strategy)
```

```
Dynamic Dispatching:
    - Prioritize high-demand zones during peak hours (morning rush,
evening, night surge).
    - Use real-time monitoring to adjust vehicle allocation.
```

```
Geofencing & Supply Management:
    - Set up geofences around busy areas (airports, business hubs).
    - Move idle vehicles to under-served regions.

Route Optimization:
    - Use algorithms for shortest, most efficient routes.
    - Minimize empty return trips by smart fleet positioning.

Driver Incentives:
    - Offer bonuses for low-demand areas and off-peak hours.
    - Reward long-distance trips to ensure broader coverage.
```

4.1.2 [5 marks]

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

```python
cab_strategy = """
Peak Hour Hotspots:
    - Morning (7 AM - 10 AM): Position cabs in residential zones for work commutes.
    - Evening (5 PM - 8 PM): Focus on business districts for return trips.

Night Demand Zones:
    - 11 PM - 3 AM: Place cabs near entertainment hubs, airports, and transport stations to capture late-night travelers.

Weekend Strategy:
    - Increase cab availability in malls, tourist spots, and event venues during weekends (especially in the afternoon and evening).

Low-Demand Redistribution:
    - Reallocate idle cabs from low-traffic areas to zones with emerging trends (e.g., newly developed regions or seasonal event areas).

Seasonal Adjustment:
    - Adjust cab supply based on monthly trends—increase near holiday destinations during vacation seasons and business hubs during work periods.

Zone-Specific Supply Balancing:
    - Ensure a balanced distribution by analyzing pickup/drop-off imbalances—position more cabs in zones with high outbound demand.
"""

print(cab_strategy)
```

```
Peak Hour Hotspots:
    - Morning (7 AM - 10 AM): Position cabs in residential zones for
work commutes.
    - Evening (5 PM - 8 PM): Focus on business districts for return
trips.

Night Demand Zones:
    - 11 PM - 3 AM: Place cabs near entertainment hubs, airports, and
transport stations to capture late-night travelers.

Weekend Strategy:
    - Increase cab availability in malls, tourist spots, and event
venues during weekends (especially in the afternoon and evening).

Low-Demand Redistribution:
    - Reallocate idle cabs from low-traffic areas to zones with
emerging trends (e.g., newly developed regions or seasonal event
areas).

Seasonal Adjustment:
    - Adjust cab supply based on monthly trends—increase near holiday
destinations during vacation seasons and business hubs during work
periods.

Zone-Specific Supply Balancing:
    - Ensure a balanced distribution by analyzing pickup/drop-off
imbalances—position more cabs in zones with high outbound demand.
```

4.1.3 [5 marks] Propose data-driven adjustments to the pricing strategy to maximize revenue while maintaining competitive rates with other vendors.

```
pricing_strategy = """
1. Dynamic Surge Pricing:
   - Increase fares during peak hours and in high-demand zones.

2. Distance-Based Tiers:
   - Lower fares for short trips, standard for medium trips, and
discounts for long trips.

3. Passenger-Based Rates:
   - Higher rates for 1-2 passengers, discounts for 3+ passengers.

4. Night & Off-Peak Discounts:
   - Offer lower fares during off-peak hours (10 PM - 6 AM) to boost
demand.

5. Loyalty Programs:
   - Introduce discounts for frequent riders and subscription plans
```

```python
for regular commuters.

6. Competitor Matching:
   - Adjust fares based on competitor pricing to stay competitive.
"""

print(pricing_strategy)
```

```
1. Dynamic Surge Pricing:
   - Increase fares during peak hours and in high-demand zones.

2. Distance-Based Tiers:
   - Lower fares for short trips, standard for medium trips, and
discounts for long trips.

3. Passenger-Based Rates:
   - Higher rates for 1-2 passengers, discounts for 3+ passengers.

4. Night & Off-Peak Discounts:
   - Offer lower fares during off-peak hours (10 PM - 6 AM) to boost
demand.

5. Loyalty Programs:
   - Introduce discounts for frequent riders and subscription plans
for regular commuters.

6. Competitor Matching:
   - Adjust fares based on competitor pricing to stay competitive.
```