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

pip install seaborn==0.13.2
```

```
Requirement already satisfied: seaborn==0.13.2 in
/opt/anaconda3/lib/python3.11/site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in
/opt/anaconda3/lib/python3.11/site-packages (from seaborn==0.13.2)
(1.26.4)
Requirement already satisfied: pandas>=1.2 in
/opt/anaconda3/lib/python3.11/site-packages (from seaborn==0.13.2)
(2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/opt/anaconda3/lib/python3.11/site-packages (from seaborn==0.13.2)
(3.10.0)
Requirement already satisfied: contourpy>=1.0.1 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn==0.13.2) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn==0.13.2) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn==0.13.2) (4.25.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn==0.13.2) (1.4.4)
Requirement already satisfied: packaging>=20.0 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn==0.13.2) (23.1)
Requirement already satisfied: pillow>=8 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn==0.13.2) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn==0.13.2) (3.0.9)
Requirement already satisfied: python-dateutil>=2.7 in
/opt/anaconda3/lib/python3.11/site-packages (from matplotlib!
=3.6.1,>=3.4->seaborn==0.13.2) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
```

```
/opt/anaconda3/lib/python3.11/site-packages (from pandas>=1.2-
>seaborn==0.13.2) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.7 in
/opt/anaconda3/lib/python3.11/site-packages (from pandas>=1.2-
>seaborn==0.13.2) (2023.3)
Requirement already satisfied: six>=1.5 in
/opt/anaconda3/lib/python3.11/site-packages (from python-
dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

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

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

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

```
numpy version: 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')`

```python
# Try loading one file

df = pd.read_parquet('/Users/vaidehimallela/Downloads/Datasets and
Dictionary/trip_records/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
```

```
df.head(20)
```

```
    VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
0          2  2023-01-01 00:32:10   2023-01-01 00:40:36
1.0
1          2  2023-01-01 00:55:08   2023-01-01 01:01:27
1.0
2          2  2023-01-01 00:25:04   2023-01-01 00:37:49
1.0
3          1  2023-01-01 00:03:48   2023-01-01 00:13:25
0.0
4          2  2023-01-01 00:10:29   2023-01-01 00:21:19
1.0
5          2  2023-01-01 00:50:34   2023-01-01 01:02:52
1.0
6          2  2023-01-01 00:09:22   2023-01-01 00:19:49
1.0
7          2  2023-01-01 00:27:12   2023-01-01 00:49:56
1.0
8          2  2023-01-01 00:21:44   2023-01-01 00:36:40
1.0
9          2  2023-01-01 00:39:42   2023-01-01 00:50:36
1.0
10         2  2023-01-01 00:53:01   2023-01-01 01:01:45
```

```
1.0
11         1  2023-01-01 00:43:37   2023-01-01 01:17:18
4.0
12         2  2023-01-01 00:34:44   2023-01-01 01:04:25
1.0
13         2  2023-01-01 00:09:29   2023-01-01 00:29:23
2.0
14         2  2023-01-01 00:33:53   2023-01-01 00:49:15
1.0
15         2  2023-01-01 00:13:04   2023-01-01 00:22:10
1.0
16         2  2023-01-01 00:45:11   2023-01-01 01:07:39
1.0
17         1  2023-01-01 00:04:33   2023-01-01 00:19:22
1.0
18         1  2023-01-01 00:03:36   2023-01-01 00:09:36
3.0
19         1  2023-01-01 00:15:23   2023-01-01 00:29:41
2.0

    trip_distance  RatecodeID store_and_fwd_flag  PULocationID
DOLocationID  \
0            0.97         1.0                  N           161
141
1            1.10         1.0                  N            43
237
2            2.51         1.0                  N            48
238
3            1.90         1.0                  N           138
7
4            1.43         1.0                  N           107
79
5            1.84         1.0                  N           161
137
6            1.66         1.0                  N           239
143
7           11.70         1.0                  N           142
200
8            2.95         1.0                  N           164
236
9            3.01         1.0                  N           141
107
10           1.80         1.0                  N           234
68
11           7.30         1.0                  N            79
264
12           3.23         1.0                  N           164
143
13          11.43         1.0                  N           138
```

```
33
14           2.95         1.0                  N            33
61
15           1.52         1.0                  N            79
186
16           2.23         1.0                  N            90
48
17           4.50         1.0                  N           113
255
18           1.20         1.0                  N           237
239
19           2.50         1.0                  N           143
229

    payment_type  fare_amount  extra  mta_tax  tip_amount
tolls_amount  \
0              2          9.3   1.00      0.5        0.00
0.0
1              1          7.9   1.00      0.5        4.00
0.0
2              1         14.9   1.00      0.5       15.00
0.0
3              1         12.1   7.25      0.5        0.00
0.0
4              1         11.4   1.00      0.5        3.28
0.0
5              1         12.8   1.00      0.5       10.00
0.0
6              1         12.1   1.00      0.5        3.42
0.0
7              1         45.7   1.00      0.5       10.74
3.0
8              1         17.7   1.00      0.5        5.68
0.0
9              2         14.9   1.00      0.5        0.00
0.0
10             1         11.4   1.00      0.5        3.28
0.0
11             1         33.8   3.50      0.5        7.75
0.0
12             1         26.1   1.00      0.5        6.22
0.0
13             1         44.3   6.00      0.5       13.26
0.0
14             1         17.7   1.00      0.5        4.04
0.0
15             1         10.0   1.00      0.5        1.25
0.0
16             1         19.8   1.00      0.5        4.96
```

```
0.0
17              1          20.5   3.50       0.5         4.00
0.0
18              2           8.6   3.50       0.5         0.00
0.0
19              2          15.6   3.50       0.5         0.00
0.0

    improvement_surcharge  total_amount  congestion_surcharge
airport_fee
0                     1.0         14.30                   2.5
0.00
1                     1.0         16.90                   2.5
0.00
2                     1.0         34.90                   2.5
0.00
3                     1.0         20.85                   0.0
1.25
4                     1.0         19.68                   2.5
0.00
5                     1.0         27.80                   2.5
0.00
6                     1.0         20.52                   2.5
0.00
7                     1.0         64.44                   2.5
0.00
8                     1.0         28.38                   2.5
0.00
9                     1.0         19.90                   2.5
0.00
10                    1.0         19.68                   2.5
0.00
11                    1.0         46.55                   2.5
0.00
12                    1.0         37.32                   2.5
0.00
13                    1.0         66.31                   0.0
1.25
14                    1.0         24.24                   0.0
0.00
15                    1.0         16.25                   2.5
0.00
16                    1.0         29.76                   2.5
0.00
17                    1.0         29.50                   2.5
0.00
18                    1.0         13.60                   2.5
0.00
```

```
19                     1.0         20.60                     2.5
0.00
```

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

# Select the folder having data files
import os

# Select the folder having data files
directory_path='/Users/vaidehimallela/Downloads/Datasets and
Dictionary/trip_records'
os.chdir(directory_path)

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

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

sampled_data_list=[]

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

data['tpep_pickup_datetime']=pd.to_datetime(data['tpep_pickup_datetime
'])
        data['date'] = data['tpep_pickup_datetime'].dt.date  # Extract
date part
        data['hour'] = data['tpep_pickup_datetime'].dt.hour  # Extract
hour
        # We will store the sampled data for the current date in this
df by appending the sampled data from each hour to this
        # After completing iteration through each date, we will append
this data to the final dataframe.
        #sampled_data = pd.DataFrame()
        file_sampled_data = []
        # Loop through dates and then loop through every hour of each
date
        for date in data['date'].unique():
            daily_data=data[data['date']==date]
            # Iterate through each hour of the selected date
            for hour in daily_data['hour'].unique():
```

```
                hourly_data=daily_data[daily_data['hour']==hour]
                # Sample 5% of the hourly data randomly
                sampled_hourly_data = hourly_data.sample(frac=0.05,
random_state=42)
                # add data of this hour to the dataframe
                file_sampled_data.append(sampled_hourly_data)
        # Concatenate the sampled data of all the dates to a single
dataframe
        file_sampled_data = pd.concat(file_sampled_data,
ignore_index=True)
        sampled_data_list.append(file_sampled_data)
    except Exception as e:
        print(f"Error reading file {file_name}: {e}")
final_sampled_data1 = pd.concat(sampled_data_list, ignore_index=True)
#final_sampled_data1.to_parquet('sampled_taxi_data.parquet',
index=False)
```

```
Error reading file .DS_Store: Could not open Parquet input source
'<Buffer>': Parquet magic bytes not found in footer. Either the file
is corrupted or this is not a parquet file.
```

```
final_sampled_data1.head()
```

```
   VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
0         2  2023-12-01 00:27:51   2023-12-01 00:50:12
1.0
1         2  2023-12-01 00:38:48   2023-12-01 01:01:55
NaN
2         2  2023-12-01 00:06:19   2023-12-01 00:16:57
1.0
3         2  2023-12-01 00:00:50   2023-12-01 00:14:37
NaN
4         2  2023-12-01 00:16:07   2023-12-01 00:19:17
1.0

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID
DOLocationID  \
0           3.99         1.0                  N           148
50
1           4.79         NaN               None           231
61
2           1.05         1.0                  N           161
161
3           2.08         NaN               None           137
144
4           0.40         1.0                  N            68
68

   payment_type  ...  mta_tax  tip_amount  tolls_amount  \
```

```
0              1  ...      0.5        5.66           0.0
1              0  ...      0.5        3.00           0.0
2              1  ...      0.5        3.14           0.0
3              0  ...      0.5        0.00           0.0
4              1  ...      0.5        0.00           0.0

   improvement_surcharge  total_amount  congestion_surcharge
Airport_fee  \
0                    1.0         33.96                   2.5
0.0
1                    1.0         29.43                   NaN
NaN
2                    1.0         18.84                   2.5
0.0
3                    1.0         21.22                   NaN
NaN
4                    1.0         10.10                   2.5
0.0

         date  hour airport_fee
0  2023-12-01     0         NaN
1  2023-12-01     0         NaN
2  2023-12-01     0         NaN
3  2023-12-01     0         NaN
4  2023-12-01     0         NaN

[5 rows x 22 columns]
```

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
final_sampled_data1.to_parquet('/Users/vaidehimallela/Downloads/Datasets and Dictionary/trip_records/Sampled_taxi_data_2023', index=False)
```

2 Data Cleaning

[30 marks]

Now we can load the new data directly.

```
# Load the new data file
df = pd.read_parquet('/Users/vaidehimallela/Downloads/Datasets and Dictionary/trip_records/Sampled_taxi_data_2023')

df.head()
```

```
   VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
0         2  2023-12-01 00:27:51   2023-12-01 00:50:12
1.0
1         2  2023-12-01 00:38:48   2023-12-01 01:01:55
NaN
2         2  2023-12-01 00:06:19   2023-12-01 00:16:57
1.0
3         2  2023-12-01 00:00:50   2023-12-01 00:14:37
NaN
4         2  2023-12-01 00:16:07   2023-12-01 00:19:17
1.0

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID
DOLocationID  \
0           3.99         1.0                  N           148
50
1           4.79         NaN               None           231
61
2           1.05         1.0                  N           161
161
3           2.08         NaN               None           137
144
4           0.40         1.0                  N            68
68

   payment_type  ...  mta_tax  tip_amount  tolls_amount  \
0             1  ...      0.5        5.66           0.0
1             0  ...      0.5        3.00           0.0
2             1  ...      0.5        3.14           0.0
3             0  ...      0.5        0.00           0.0
4             1  ...      0.5        0.00           0.0

   improvement_surcharge  total_amount  congestion_surcharge
Airport_fee  \
0                    1.0         33.96                   2.5
0.0
1                    1.0         29.43                   NaN
NaN
2                    1.0         18.84                   2.5
0.0
3                    1.0         21.22                   NaN
NaN
4                    1.0         10.10                   2.5
0.0

         date  hour airport_fee
0  2023-12-01     0         NaN
1  2023-12-01     0         NaN
2  2023-12-01     0         NaN
```

```
3  2023-12-01     0         NaN
4  2023-12-01     0         NaN

[5 rows x 22 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1996062 entries, 0 to 1996061
Data columns (total 22 columns):
 #   Column                 Dtype
---  ------                 -----
 0   VendorID               int64
 1   tpep_pickup_datetime   datetime64[us]
 2   tpep_dropoff_datetime  datetime64[us]
 3   passenger_count        float64
 4   trip_distance          float64
 5   RatecodeID             float64
 6   store_and_fwd_flag     object
 7   PULocationID           int64
 8   DOLocationID           int64
 9   payment_type           int64
 10  fare_amount            float64
 11  extra                  float64
 12  mta_tax                float64
 13  tip_amount             float64
 14  tolls_amount           float64
 15  improvement_surcharge  float64
 16  total_amount           float64
 17  congestion_surcharge   float64
 18  Airport_fee            float64
 19  date                   object
 20  hour                   int32
 21  airport_fee            float64
dtypes: datetime64[us](2), float64(13), int32(1), int64(4), object(2)
memory usage: 327.4+ 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

df.reset_index(drop=True,inplace=True)
```

```
df=df.drop(columns=['extra'])
```

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
df['Airport__Fee'] =
df['airport_fee'].combine_first(df['Airport_fee'])

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1996062 entries, 0 to 1996061
Data columns (total 22 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  mta_tax                float64
 12  tip_amount             float64
 13  tolls_amount           float64
 14  improvement_surcharge  float64
 15  total_amount           float64
 16  congestion_surcharge   float64
 17  Airport_fee            float64
 18  date                   object
 19  hour                   int32
 20  airport_fee            float64
 21  Airport__Fee           float64
dtypes: datetime64[us](2), float64(13), int32(1), int64(4), object(2)
memory usage: 327.4+ MB

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

2.1.3 [5 marks] Fix columns with negative (monetary) values

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

```
#df[df['fare_amount'] < 0]
(df['fare_amount'] < 0).sum()
```

```
0
```

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

```
# Analyse RatecodeID for the negative fare amounts
df[df['RatecodeID']<0].shape[0]
```

```
0
```

```
# there are no -ve values for RatecodeID
# lets check max and min value in RatecodeID
print("maximum value for RatecodeID :"+str(df['RatecodeID'].max())+"\n
Minimum value for RatecodeID :"+str(df['RatecodeID'].min()))
```

```
maximum value for RatecodeID :99.0
 Minimum value for RatecodeID :1.0
```

```
# lets check null value in Ratecode id
print("no of rows for column RatecodeID is having null value :
"+str(df[df['RatecodeID'].isnull()].shape[0]))
```

```
no of rows for column RatecodeID is having null value : 68203
```

```
# as 64874 rows is just 3% of the data set df_1 we can remove this
rows from the df_1
df=df[~(df['RatecodeID'].isnull())]

# Find which columns have negative values
columns_with_numeric=df.select_dtypes(exclude=['object','datetime64'])
.columns
columns_with_neg_values=columns_with_numeric[(df[columns_with_numeric]
< 0).any()].tolist()
print("Columns with negative values : \n", columns_with_neg_values)
```

```
Columns with negative values :
 ['mta_tax', 'improvement_surcharge', 'total_amount',
'congestion_surcharge', 'Airport__Fee']
```

```
# now I can make RatecodeID as integer
df['RatecodeID']=df['RatecodeID'].astype(int)

# fix these negative values
# 'mta_tax', 'improvement_surcharge', 'total_amount',
'congestion_surcharge', 'Airport__Fee' are the columns were -ve values
present
#lets count no of -ve values these columns have
for c in columns_with_neg_values :
```

```
  print("no of -ve values in column "+c+" :
"+str(df[df[c]<0].shape[0]))
```

```
no of -ve values in column mta_tax : 76
no of -ve values in column improvement_surcharge : 81
no of -ve values in column total_amount : 81
no of -ve values in column congestion_surcharge : 59
no of -ve values in column Airport__Fee : 15
```

```
for colm in columns_with_neg_values :
  df=df[~(df[colm]<0)]
```

2.2 Handling Missing Values

[10 marks]

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

```
# Find the proportion of missing values in each column
print('Number of null values in  each column : \n\n'
+str((df.isnull().mean()*100)))
```

```
Number of null values in  each column :

VendorID                 0.0
tpep_pickup_datetime     0.0
tpep_dropoff_datetime    0.0
passenger_count          0.0
trip_distance            0.0
RatecodeID               0.0
store_and_fwd_flag       0.0
PULocationID             0.0
DOLocationID             0.0
payment_type             0.0
fare_amount              0.0
mta_tax                  0.0
tip_amount               0.0
tolls_amount             0.0
improvement_surcharge    0.0
total_amount             0.0
congestion_surcharge     0.0
date                     0.0
hour                     0.0
Airport__Fee             0.0
dtype: float64
```

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

```
# Display the rows with null values
# Impute NaN values in 'passenger_count'
```

```python
print('number of rows with null value for column passenger_count :
'+str(df[df['passenger_count'].isnull()].shape[0]))
```

```
number of rows with null value for column passenger_count :   0
```

Did you find zeroes in passenger_count? Handle these.

```python
# but lets check how rows are having passenger value 0
print('number of rows with  value 0 for column passenger_count :
'+str(df[df['passenger_count']==0].shape[0]))
```

```
number of rows with  value 0 for column passenger_count :   31256
```

```python
df[df['passenger_count']==0][['passenger_count','total_amount']]
```

	passenger_count	total_amount
152	0.0	22.20
173	0.0	96.75
349	0.0	15.45
382	0.0	15.23
530	0.0	10.90
...	...	...
1995783	0.0	18.85
1995862	0.0	25.55
1996048	0.0	37.95
1996051	0.0	21.00
1996060	0.0	24.00

```
[31256 rows x 2 columns]
```

```python
print("number of rows in dataset : "+str(df.shape[0]))
```

```
number of rows in dataset : 1927778
```

```python
# so rows with passenger count 0 is 1.6 percentage hence we can
remove this data from analysis rather than subtituting values with
mean or median
```

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

```python
# Fix missing values in 'RatecodeID'
print('number of rows with RatecodeID as null:
'+str(df_1[df_1['RatecodeID'].isnull()].shape[0]))
```

```
number of rows with RatecodeID as null: 0
```

2.2.4 [3 marks] Impute NaN in `congestion_surcharge`

```python
# handle null values in congestion_surcharge
print('number of rows with RatecodeID as null:
```

```
'+str(df_1[df_1['congestion_surcharge'].isnull()].shape[0]))

number of rows with RatecodeID as null: 0
```

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
```

2.3 Handling Outliers

[10 marks]

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

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)

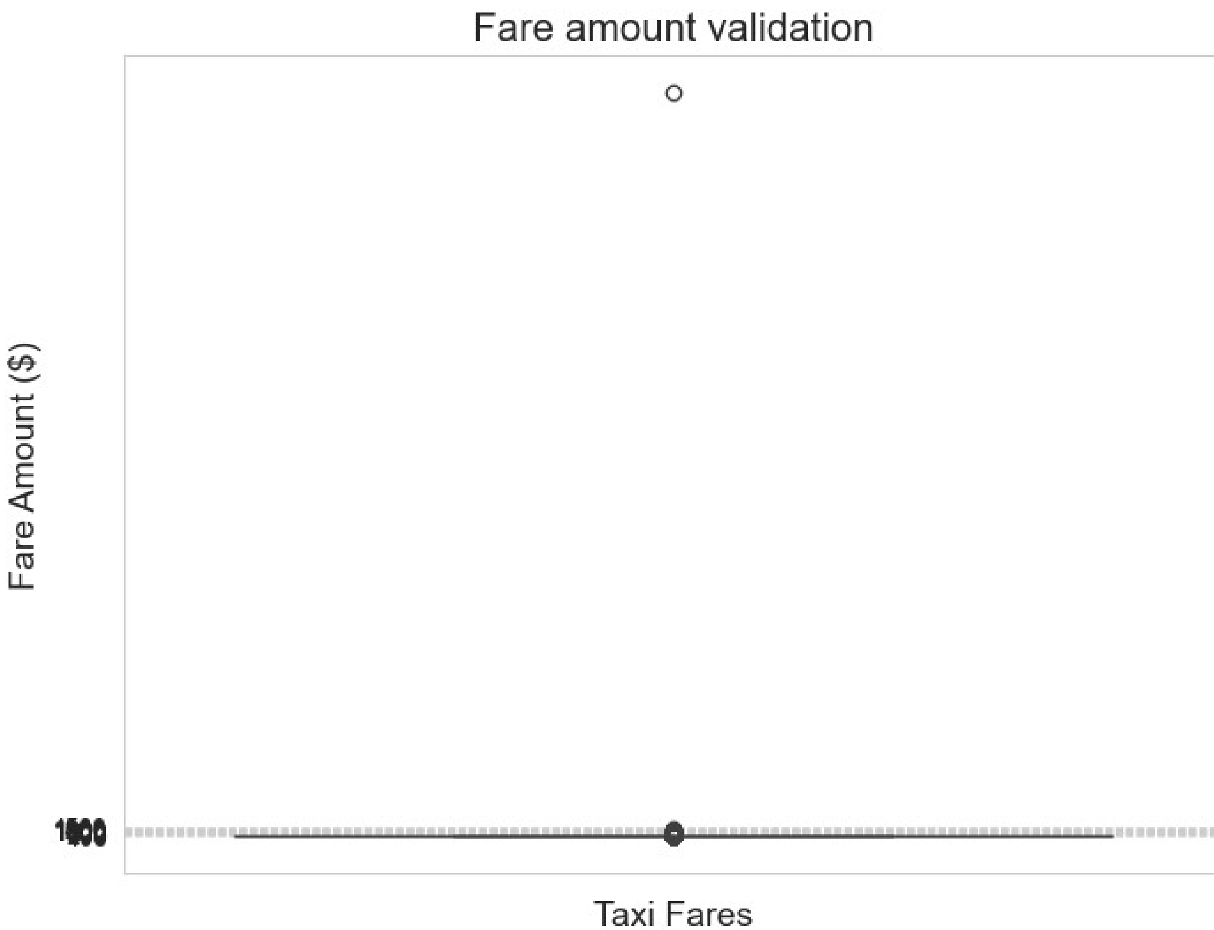
These are just some suggestions. You can handle outliers in any way you wish, using the insights from above outlier analysis.

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

print('rows with passenger_count 
>6 :'+str(df[df['passenger_count']>6].shape[0]))
df=df[~(df['passenger_count']>6)]

rows with passenger_count >6 :21

# Continue with outlier handling
# Continue with outlier handling
# lets check the  fare_amount
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['fare_amount'])
plt.title('Fare amount validation',fontsize=16)
plt.ylabel('Fare Amount ($)', fontsize=14)
```

```
plt.xlabel('Taxi Fares', fontsize=14)
plt.yticks(range(0, 1600, 100))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



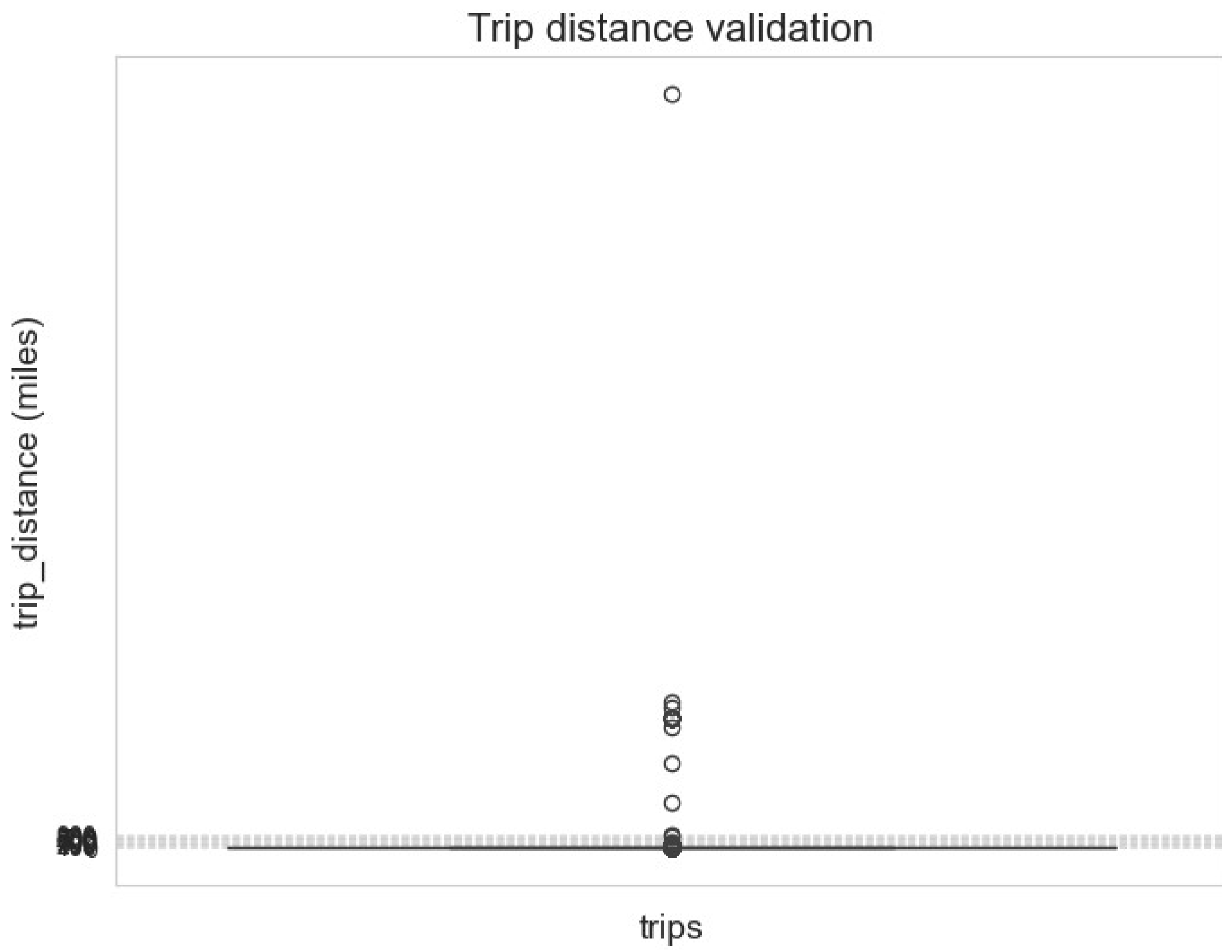

```
# most of the fares are below 300  (it is not clear as it is too much
skewed )
#lets check how many rows have fare amount more than 50 100 , 200  and
300
print('no of rows with fare_amount 0 :
'+str(df[df['fare_amount']==0].shape[0]))
for fare in (0,50,100,200,300,500,600,700,800,900,1000):
    print('no of rows with fare_amount >'+str(fare)+' :
'+str(df[df['fare_amount']>fare].shape[0]))

no of rows with fare_amount 0 : 579
no of rows with fare_amount >0 : 1927178
no of rows with fare_amount >50 : 145258
no of rows with fare_amount >100 : 6216
no of rows with fare_amount >200 : 674
```

```
no of rows with fare_amount >300 : 173
no of rows with fare_amount >500 : 23
no of rows with fare_amount >600 : 14
no of rows with fare_amount >700 : 8
no of rows with fare_amount >800 : 4
no of rows with fare_amount >900 : 4
no of rows with fare_amount >1000 : 2
```

```
#lets find out  max and min values for trip_distance
print('maximum distance taxi travelled in
data :'+str(df['trip_distance'].max()))
print('minimum distance taxi travelled in
data :'+str(df['trip_distance'].min()))
```

```
maximum distance taxi travelled in data :56823.8
minimum distance taxi travelled in data :0.0
```

```
# seems interesting about the fareamount grater than 300  are those
real values
# lets check the distance travelled as well
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['trip_distance'])
plt.title('Trip distance validation',fontsize=16)
plt.ylabel('trip_distance (miles)', fontsize=14)
plt.xlabel('trips', fontsize=14)
plt.yticks(range(0, 1000, 100))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```




```
# as we have seen above box plot and values displayed , lets consider our base fare amount as 300$ and lets trim the data above 300$ fare amount
#let df4 be that dataframe
df=df[df['fare_amount']<=300]

df.reset_index(drop=True,inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1927584 entries, 0 to 1927583
Data columns (total 20 columns):
 #   Column                 Dtype
---  ------                 -----
 0   VendorID               int64
 1   tpep_pickup_datetime   datetime64[us]
 2   tpep_dropoff_datetime  datetime64[us]
 3   passenger_count        float64
 4   trip_distance          float64
 5   RatecodeID             int64
 6   store_and_fwd_flag     object
```

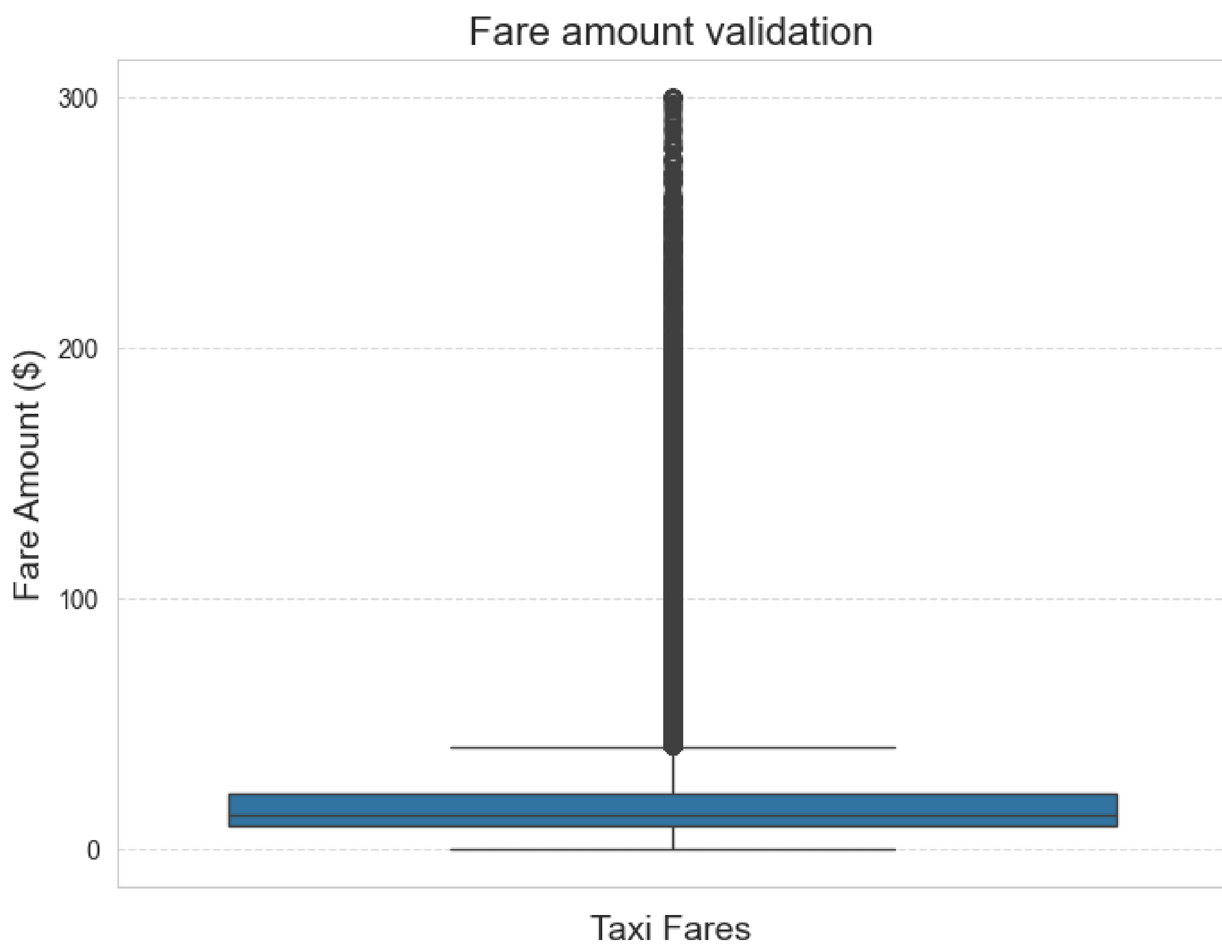
```
 7   PULocationID           int64
 8   DOLocationID           int64
 9   payment_type           int64
 10  fare_amount            float64
 11  mta_tax                float64
 12  tip_amount             float64
 13  tolls_amount           float64
 14  improvement_surcharge  float64
 15  total_amount           float64
 16  congestion_surcharge   float64
 17  date                   object
 18  hour                   int32
 19  Airport__Fee           float64
dtypes: datetime64[us](2), float64(10), int32(1), int64(5), object(2)
memory usage: 286.8+ MB
```

```
# checking my data set is having duplicate rows
duplicates = df.duplicated()
print(df[duplicates])
```

```
         VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
784181          1  2023-12-01 00:45:28   2023-12-01 00:52:38
6.0
784182          1  2023-12-01 00:21:40   2023-12-01 00:46:40
0.0
784183          2  2023-12-01 00:16:19   2023-12-01 00:22:29
1.0
784184          1  2023-12-01 00:52:04   2023-12-01 00:54:48
1.0
784185          2  2023-12-01 00:20:34   2023-12-01 00:26:07
1.0
...           ...                  ...                   ...
...
1927529         2  2023-06-30 22:55:29   2023-06-30 22:58:56
1.0
1927533         1  2023-06-30 22:08:31   2023-06-30 22:36:10
1.0
1927549         2  2023-06-30 22:05:10   2023-06-30 22:19:36
1.0
1927555         2  2023-06-30 22:09:59   2023-06-30 22:22:04
1.0
1927557         2  2023-06-30 22:52:32   2023-06-30 23:04:12
1.0

         trip_distance  RatecodeID store_and_fwd_flag  PULocationID  \
784181            1.40           1                  N           230
784182           17.60           2                  N           132
784183            1.25           1                  N           229
784184            0.80           1                  N           234
```

```
784185         0.97          1                  N          90
...             ...        ...                ...         ...
1927529        0.73          1                  N         151
1927533       10.50          1                  N         138
1927549        1.46          1                  N         186
1927555        1.04          1                  N         100
1927557        2.01          1                  N         237

         DOLocationID  payment_type  fare_amount  mta_tax  tip_amount  
\
784181             90             1          9.3      0.5        3.55  

784182            162             1         70.0      0.5       14.06  

784183            137             1          8.6      0.5        2.72  

784184            100             1          5.8      0.5        2.15  

784185            164             1          7.9      0.5        2.58  

...               ...           ...          ...      ...         ...  

1927529           239             1          6.5      0.5        2.30  

1927533           170             4         42.9      0.5        0.00  

1927549           233             1         13.5      0.5        3.70  

1927555           233             2         11.4      0.5        0.00  

1927557           234             1         12.8      0.5        3.56  


         tolls_amount  improvement_surcharge  total_amount  \
784181           0.00                    1.0         17.85   
784182           6.94                    1.0         96.75   
784183           0.00                    1.0         16.32   
784184           0.00                    1.0         12.95   
784185           0.00                    1.0         15.48   
...               ...                    ...           ...   
1927529          0.00                    1.0         13.80   
1927533          6.55                    1.0         61.20   
1927549          0.00                    1.0         22.20   
1927555          0.00                    1.0         16.40   
1927557          0.00                    1.0         21.36   

         congestion_surcharge        date  hour  Airport_fee  
784181                    2.5  2023-12-01     0         0.00  
784182                    2.5  2023-12-01     0         1.75  
784183                    2.5  2023-12-01     0         0.00  
784184                    2.5  2023-12-01     0         0.00  
```

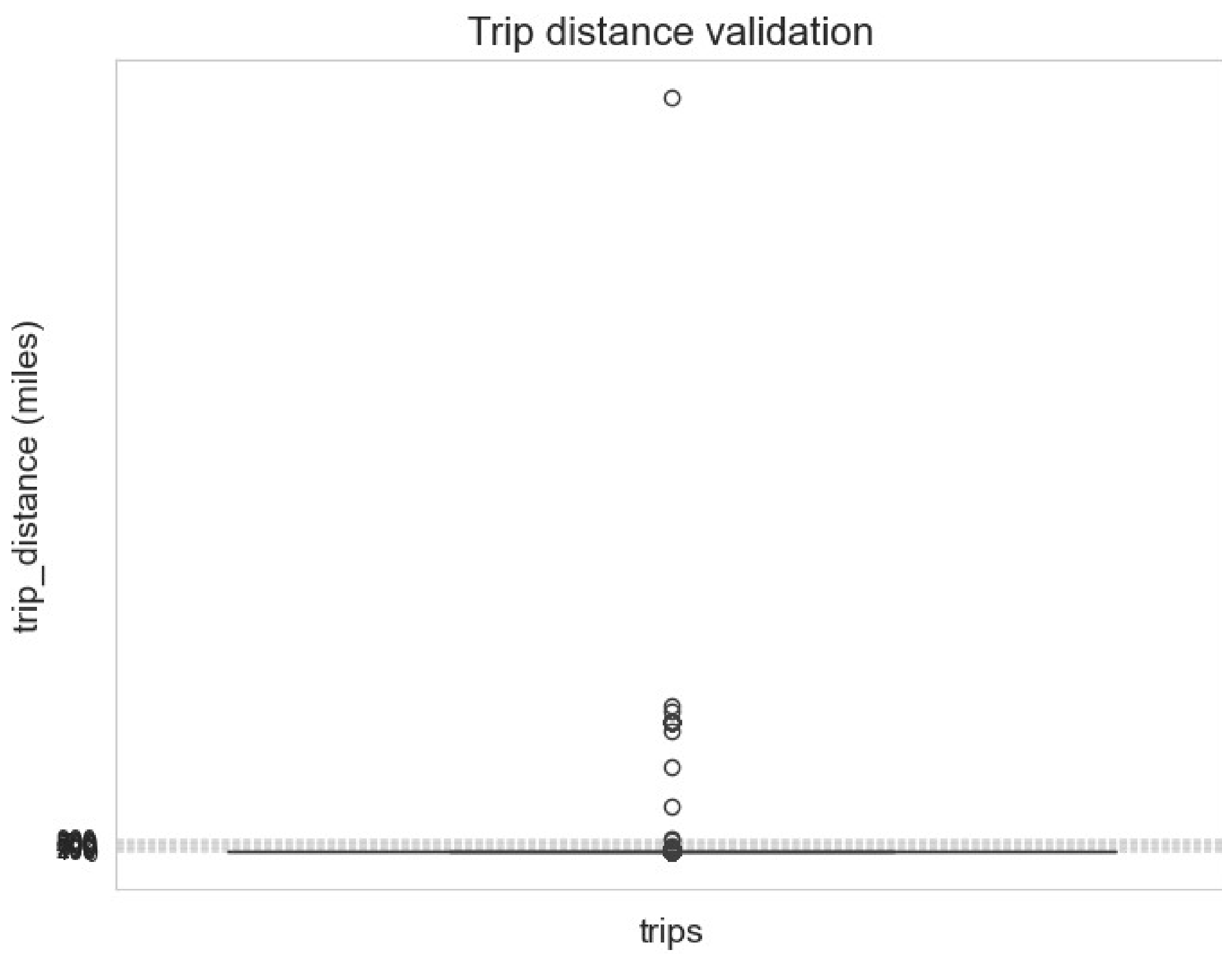
```
784185                          2.5  2023-12-01     0           0.00
...                             ...         ...   ...            ...
1927529                         2.5  2023-06-30    22           0.00
1927533                         2.5  2023-06-30    22           1.75
1927549                         2.5  2023-06-30    22           0.00
1927555                         2.5  2023-06-30    22           0.00
1927557                         2.5  2023-06-30    22           0.00

[96320 rows x 20 columns]

# Continue with outlier handling
# lets check the  fare_amount
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['fare_amount'])
plt.title('Fare amount validation',fontsize=16)
plt.ylabel('Fare Amount ($)', fontsize=14)
plt.xlabel('Taxi Fares', fontsize=14)
plt.yticks(range(0, 400, 100))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



```
print('maximum distance taxi travelled in
data :'+str(df['trip_distance'].max()))
print('minimum distance taxi travelled in
data :'+str(df['trip_distance'].min()))
print('minimum distance taxi travelled in
data :'+str(df['trip_distance'].mean()))
```

```
maximum distance taxi travelled in data :56823.8
minimum distance taxi travelled in data :0.0
minimum distance taxi travelled in data :3.5476380173315407
```

```
# lets check the distance travelled as well
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['trip_distance'])
plt.title('Trip distance validation',fontsize=16)
plt.ylabel('trip_distance (miles)', fontsize=14)
plt.xlabel('trips', fontsize=14)
plt.yticks(range(0, 1000, 100))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



```
#as seen above  some large values are there and some distance values
are 0 we need to remove this as well from the data set
#first find out columns
trip_distance,tpep_pickup_datetime,tpep_dropoff_datetime for
trip_distance is 0
print(df[df['trip_distance'] == 0][['trip_distance',
'tpep_pickup_datetime', 'tpep_dropoff_datetime']])
```

```
         trip_distance tpep_pickup_datetime tpep_dropoff_datetime
5                  0.0  2023-12-01 00:36:28   2023-12-01 00:36:34
212                0.0  2023-12-01 01:08:22   2023-12-01 01:08:56
220                0.0  2023-12-01 01:03:11   2023-12-01 01:03:15
239                0.0  2023-12-01 01:08:05   2023-12-01 01:14:02
267                0.0  2023-12-01 01:33:00   2023-12-01 01:33:35
...                ...                  ...                   ...
1926517            0.0  2023-06-30 15:34:46   2023-06-30 15:35:07
1926531            0.0  2023-06-30 15:45:52   2023-06-30 15:46:35
1927248            0.0  2023-06-30 18:50:38   2023-06-30 18:50:48
1927366            0.0  2023-06-30 22:36:31   2023-06-30 22:36:34
1927582            0.0  2023-06-30 22:29:55   2023-06-30 22:30:20

[24154 rows x 3 columns]
```

```
# from above there are 3338 rows with trip distance 0.0 lets remove
those
df=df[~(df['trip_distance']==0)]
print('maximum distance taxi travelled in
data :'+str(df['trip_distance'].max()))
print('minimum distance taxi travelled in
data :'+str(df['trip_distance'].min()))
print('minimum distance taxi travelled in
data :'+str(df['trip_distance'].mean()))
```

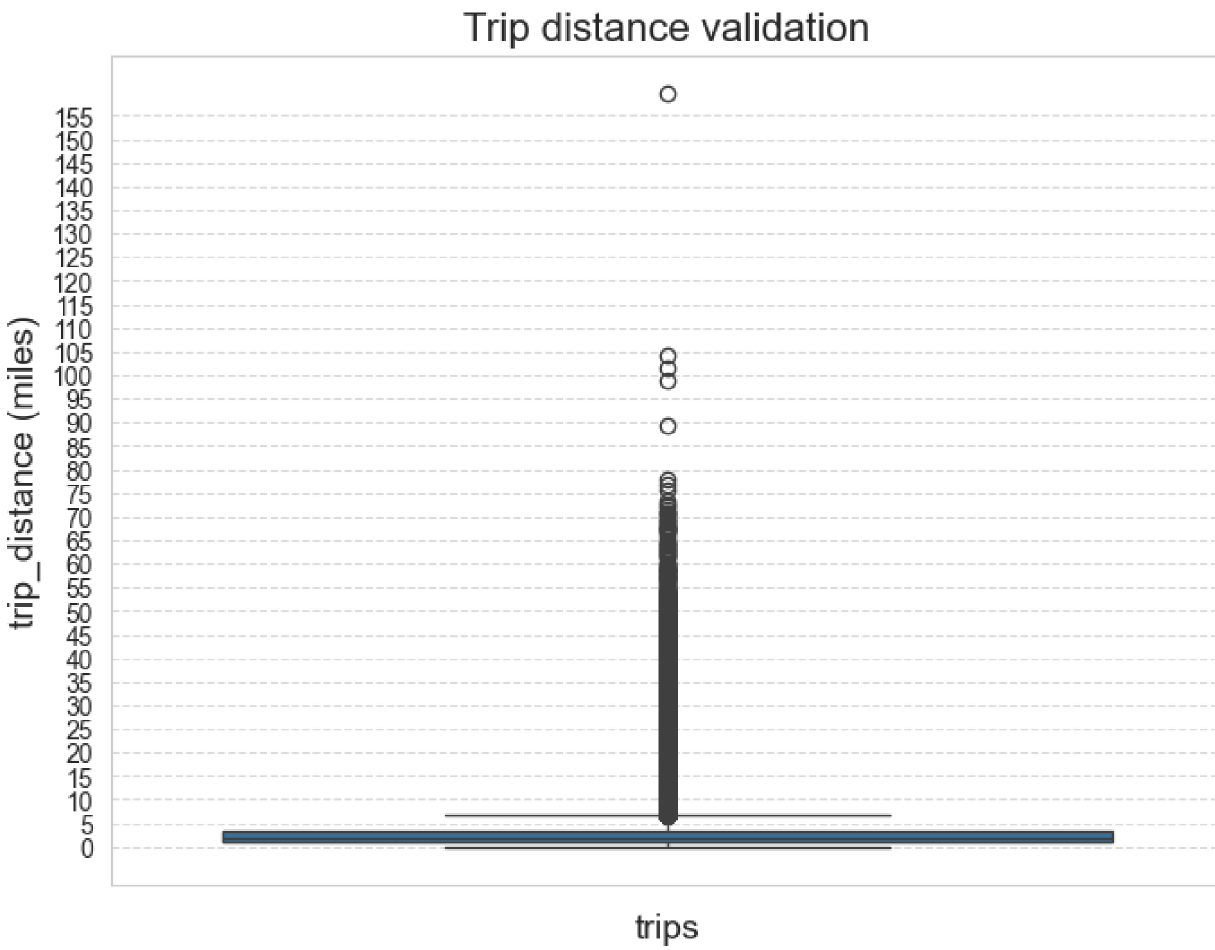
```
maximum distance taxi travelled in data :56823.8
minimum distance taxi travelled in data :0.01
minimum distance taxi travelled in data :3.5926565621010487
```

```
# now sort the column trip distance and find the 3rd largest value as
1st and 2nd highest value seems to be outlier
sorted_value=df['trip_distance'].sort_values(ascending=False)
print("Highest:" + str(sorted_value.iloc[0]))
print("Second Highest:" + str(sorted_value.iloc[1]))
print("Third Highest:"+str(sorted_value.iloc[2]))
```

```
Highest:56823.8
Second Highest:10961.43
Third Highest:10452.6
```

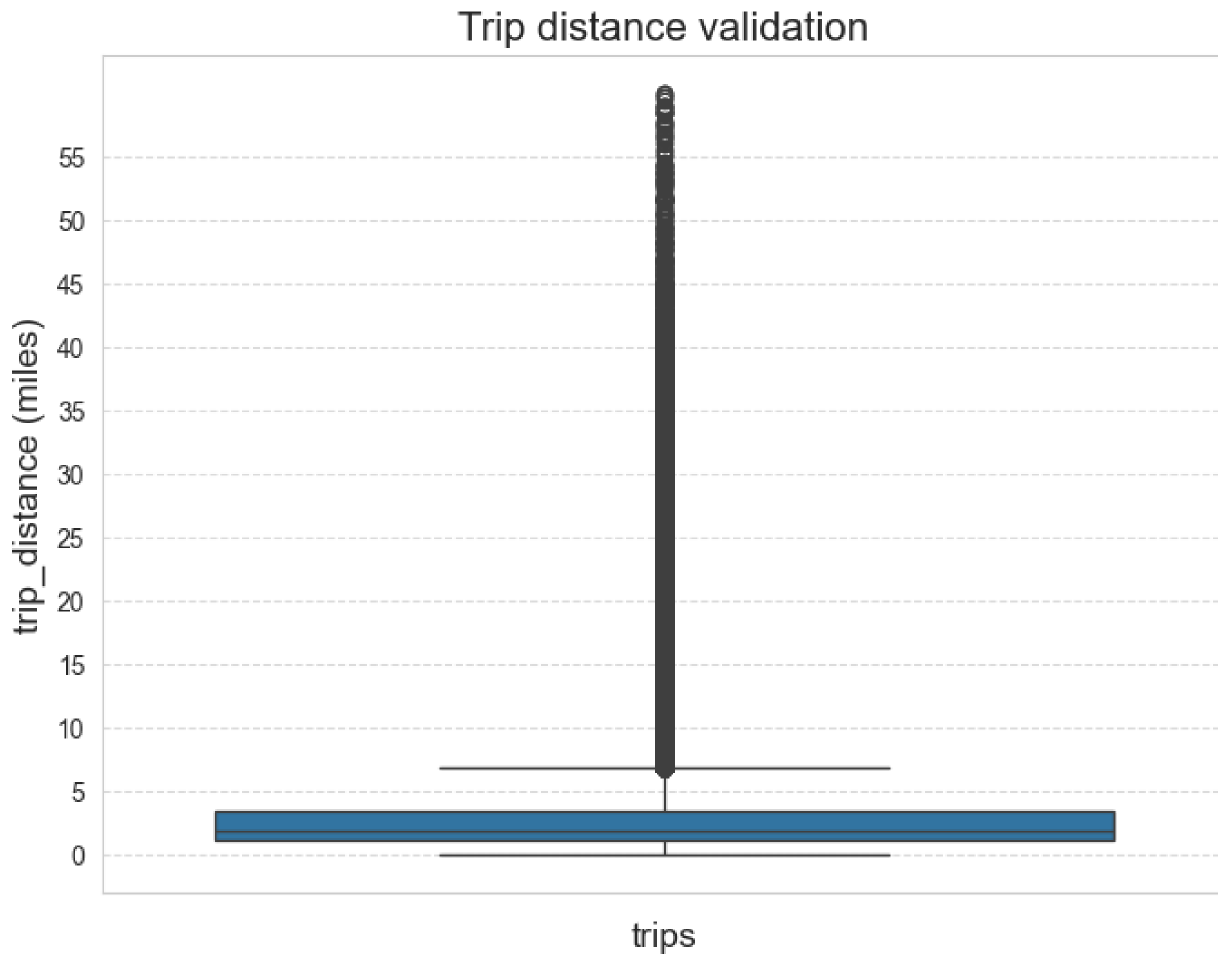
```
# lets remove trip distance more than 160 miles
df=df[~(df['trip_distance']>160)]
# lets check the distance travelled as well
```

```
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['trip_distance'])
plt.title('Trip distance validation',fontsize=16)
plt.ylabel('trip_distance (miles)', fontsize=14)
plt.xlabel('trips', fontsize=14)
plt.yticks(range(0, 160, 5))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```




```
# again if you notice only few trips above 60 we can remove that as
well proper analysis
df=df[~(df['trip_distance']>60)]
# lets check the distance travelled as well
plt.figure(figsize=(8, 6))
sns.boxplot(y=df['trip_distance'])
plt.title('Trip distance validation',fontsize=16)
plt.ylabel('trip_distance (miles)', fontsize=14)
plt.xlabel('trips', fontsize=14)
plt.yticks(range(0, 60, 5))
```

```
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```




```
# Do any columns need standardising?
df.reset_index(drop=True,inplace=True)
df.info()
df.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1903362 entries, 0 to 1903361
Data columns (total 20 columns):
 #   Column                 Dtype
---  ------                 -----
 0   VendorID               int64
 1   tpep_pickup_datetime   datetime64[us]
 2   tpep_dropoff_datetime  datetime64[us]
 3   passenger_count        float64
 4   trip_distance          float64
 5   RatecodeID             int64
 6   store_and_fwd_flag     object
```

```
 7   PULocationID           int64
 8   DOLocationID           int64
 9   payment_type           int64
 10  fare_amount            float64
 11  mta_tax                float64
 12  tip_amount             float64
 13  tolls_amount           float64
 14  improvement_surcharge  float64
 15  total_amount           float64
 16  congestion_surcharge   float64
 17  date                   object
 18  hour                   int32
 19  Airport__Fee           float64
dtypes: datetime64[us](2), float64(10), int32(1), int64(5), object(2)
memory usage: 283.2+ MB

   VendorID tpep_pickup_datetime tpep_dropoff_datetime
passenger_count  \
0         2  2023-12-01 00:27:51   2023-12-01 00:50:12
1.0
1         2  2023-12-01 00:06:19   2023-12-01 00:16:57
1.0
2         2  2023-12-01 00:16:07   2023-12-01 00:19:17
1.0
3         2  2023-12-01 00:57:08   2023-12-01 01:05:49
1.0
4         2  2023-12-01 00:46:28   2023-12-01 00:59:29
2.0

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID
DOLocationID  \
0           3.99           1                  N           148
50
1           1.05           1                  N           161
161
2           0.40           1                  N            68
68
3           1.66           1                  N           114
186
4           2.45           1                  N           164
232

   payment_type  fare_amount  mta_tax  tip_amount  tolls_amount  \
0             1         23.3      0.5        5.66           0.0
1             1         10.7      0.5        3.14           0.0
2             1          5.1      0.5        0.00           0.0
3             1         10.7      0.5        3.14           0.0
4             1         14.9      0.5        1.00           0.0

   improvement_surcharge  total_amount  congestion_surcharge
```

```
date  \
0                    1.0          33.96                        2.5  2023-12-
01
1                    1.0          18.84                        2.5  2023-12-
01
2                    1.0          10.10                        2.5  2023-12-
01
3                    1.0          18.84                        2.5  2023-12-
01
4                    1.0          20.90                        2.5  2023-12-
01

   hour  Airport__Fee
0     0           0.0
1     0           0.0
2     0           0.0
3     0           0.0
4     0           0.0
```

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',
 'mta_tax',
 'tip_amount',
 'tolls_amount',
 'improvement_surcharge',
 'total_amount',
 'congestion_surcharge',
 'date',
 'hour',
 '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

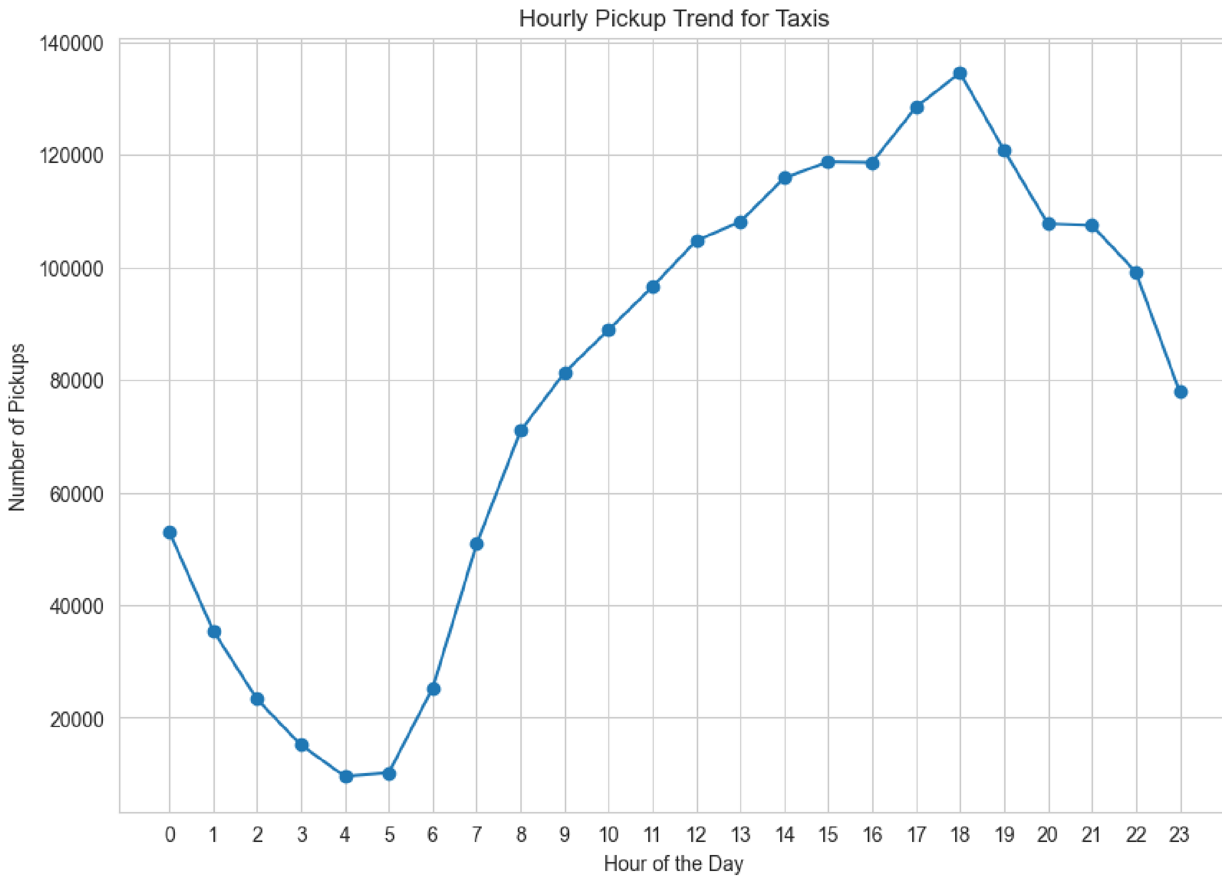
3.1.2 [5 marks] Analyse the distribution of taxi pickups by hours, days of the week, and months.

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

# Find and show the hourly trends in taxi pickups
hourly_pickups =
df.groupby('hour').size().reset_index(name='pickup_count')
print(hourly_pickups)
```

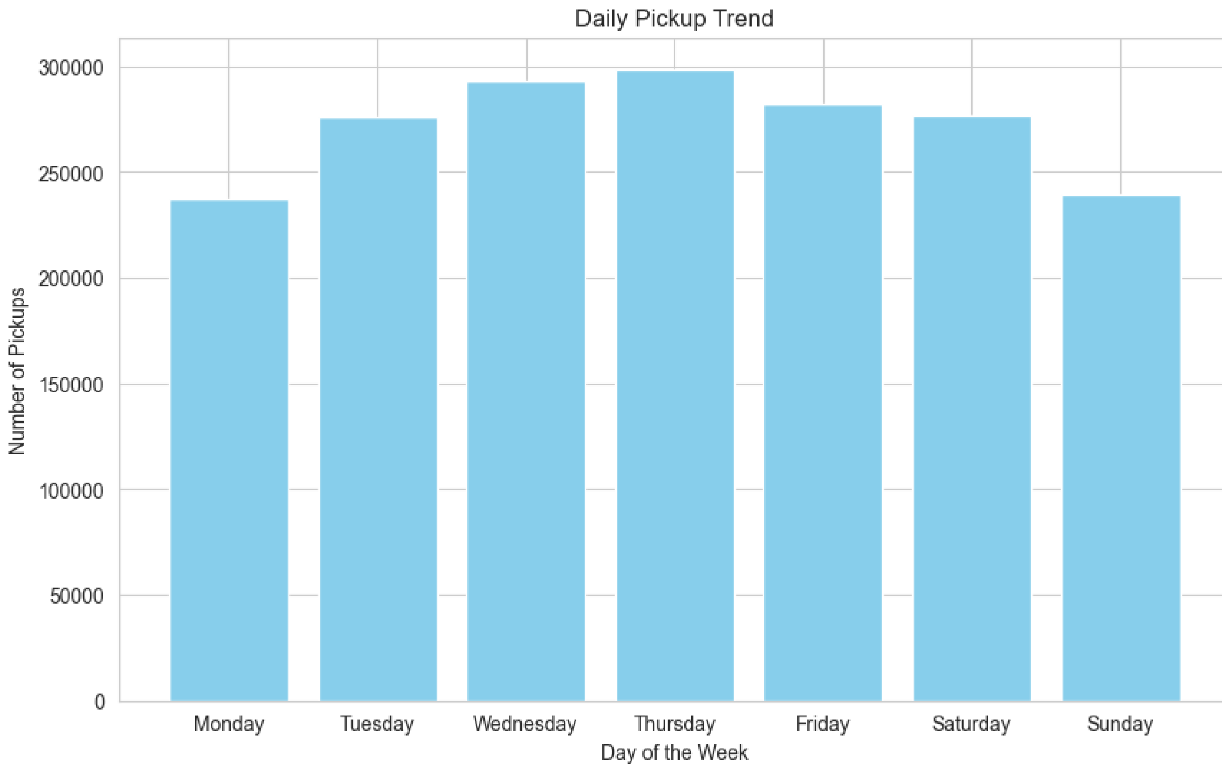
```
    hour  pickup_count
0      0         53102
1      1         35534
2      2         23302
3      3         15162
4      4          9596
5      5         10241
6      6         25300
7      7         50876
8      8         70994
9      9         81320
```

```
10    10         88895
11    11         96564
12    12        104761
13    13        108115
14    14        115891
15    15        118769
16    16        118661
17    17        128494
18    18        134567
19    19        120871
20    20        107773
21    21        107492
22    22         99193
23    23         77889
```

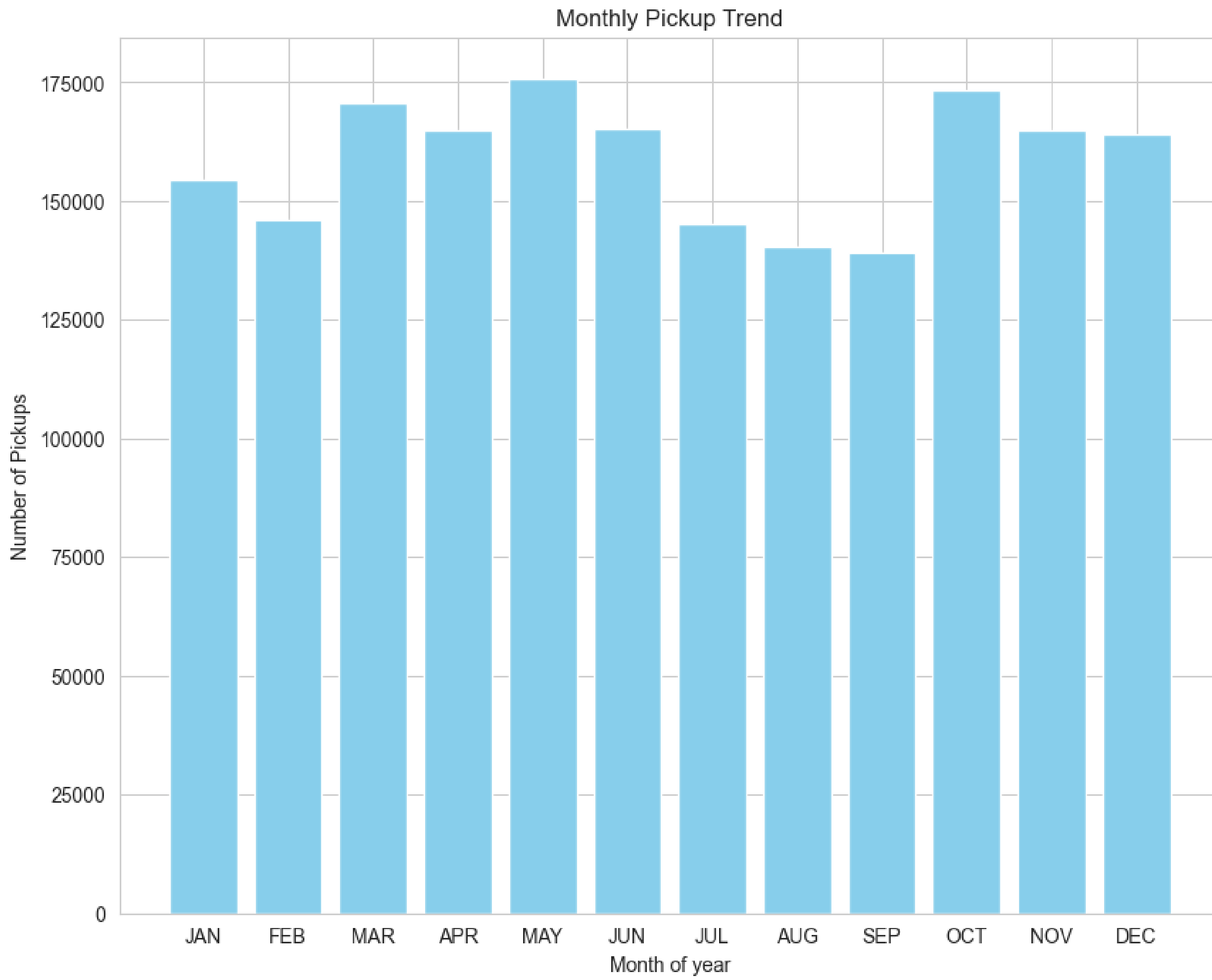
```
# Find and show the daily trends in taxi pickups (days of the week)
plt.figure(figsize=(10, 7))
plt.plot(hourly_pickups['hour'], hourly_pickups['pickup_count'],
marker='o', linestyle='-')
plt.title('Hourly Pickup Trend for Taxis')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Pickups')
plt.xticks(range(24))
plt.grid(True)
plt.show()
```




```
# lets add one column for week_Day
df['week_Day']=df['tpep_pickup_datetime'].dt.dayofweek
#lets consider as a set
day_names = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday',
4: 'Friday', 5: 'Saturday', 6: 'Sunday'}
df['day_name'] = df['week_Day'].map(day_names)
daily_pickups =
df.groupby('day_name').size().reindex(day_names.values()).reset_index(
name='pickup_count')
plt.figure(figsize=(10, 6))
plt.bar(daily_pickups['day_name'], daily_pickups['pickup_count'],
color='skyblue')
plt.title('Daily Pickup Trend')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Pickups')
plt.grid(True)
plt.show()
```




```
# Show the monthly trends in pickups

df['Month']=df['tpep_pickup_datetime'].dt.month
months_ = {1: 'JAN', 2: 'FEB', 3: 'MAR', 4: 'APR', 5: 'MAY', 6: 'JUN',
7: 'JUL', 8: 'AUG', 9: 'SEP', 10: 'OCT', 11: 'NOV', 12: 'DEC' }
df['Month_name']=df['Month'].map(months_)
monthly_trend=df.groupby('Month_name').size().reindex(months_.values()
).reset_index(name='pickup_count')
plt.figure(figsize=(10, 8))
plt.bar(monthly_trend['Month_name'], monthly_trend['pickup_count'],
color='skyblue')
plt.title('Monthly Pickup Trend')
plt.xlabel('Month of year')
plt.ylabel('Number of Pickups')
plt.grid(True)
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
print('now of rows with
fare_amount=0'+str(df[df['fare_amount']<=0].shape[0]))
print('now of rows with
total_amount=0'+str(df[df['total_amount']<=0].shape[0]))
print('now of rows with
tip_amount=0'+str(df[df['tip_amount']<=0].shape[0]))
print('now of rows with
trip_distance=0'+str(df[df['trip_distance']<=0].shape[0]))
```

```
now of rows with fare_amount=0268
now of rows with total_amount=081
now of rows with tip_amount=0416209
now of rows with trip_distance=00
```

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

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1903362 entries, 0 to 1903361
Data columns (total 24 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             int64
 6   store_and_fwd_flag     object
 7   PULocationID           int64
 8   DOLocationID           int64
 9   payment_type           int64
 10  fare_amount            float64
 11  mta_tax                float64
 12  tip_amount             float64
 13  tolls_amount           float64
 14  improvement_surcharge  float64
 15  total_amount           float64
 16  congestion_surcharge   float64
 17  date                   object
 18  hour                   int32
 19  Airport__Fee           float64
 20  week_Day               int32
 21  day_name               object
 22  Month                  int32
 23  Month_name             object
dtypes: datetime64[us](2), float64(10), int32(3), int64(5), object(4)
memory usage: 326.7+ MB
```

3.1.3 [2 marks] Filter out the zero values from the above columns.

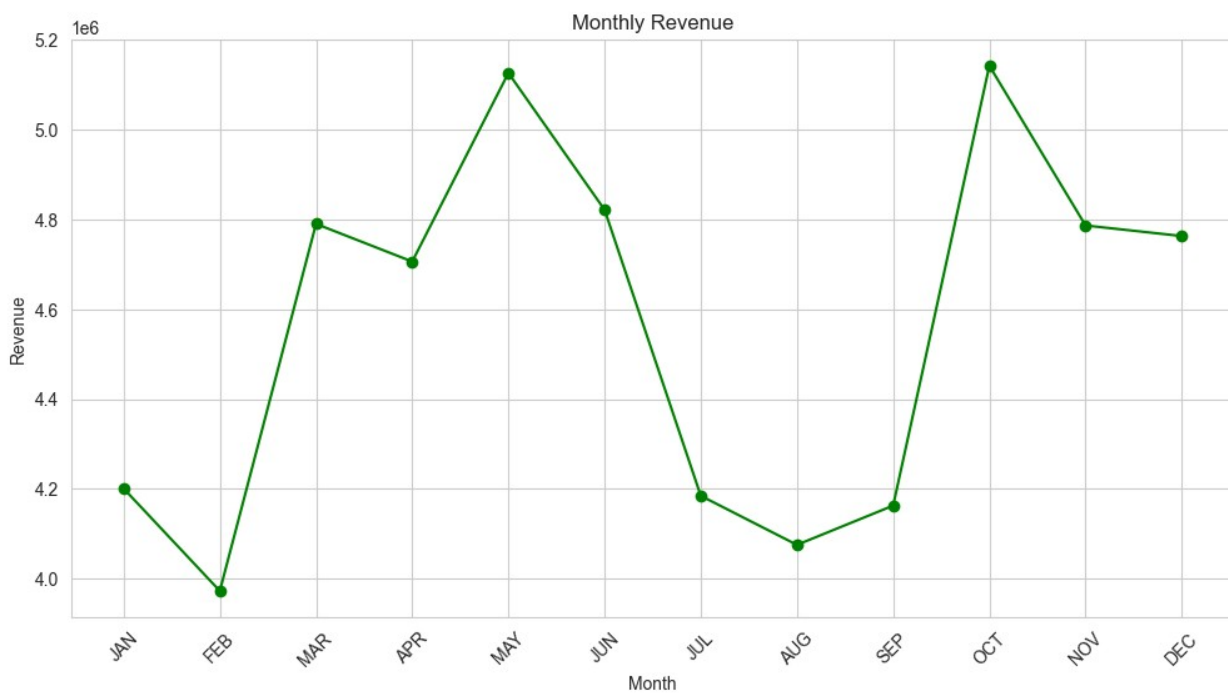
Note: The distance might be 0 in cases where pickup and drop is in the same zone. Do you think it is suitable to drop such cases of zero distance?

```
# Create a df with non zero entries for the selected parameters.
print('now of rows with
fare_amount=0'+str(df[df['fare_amount']<=0].shape[0]))
print('now of rows with
total_amount=0'+str(df[df['total_amount']<=0].shape[0]))
print('now of rows with
tip_amount=0'+str(df[df['tip_amount']<=0].shape[0]))
print('now of rows with
trip_distance=0'+str(df[df['trip_distance']<=0].shape[0]))
```

```
now of rows with fare_amount=0268
now of rows with total_amount=081
now of rows with tip_amount=0416209
now of rows with trip_distance=00
```

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

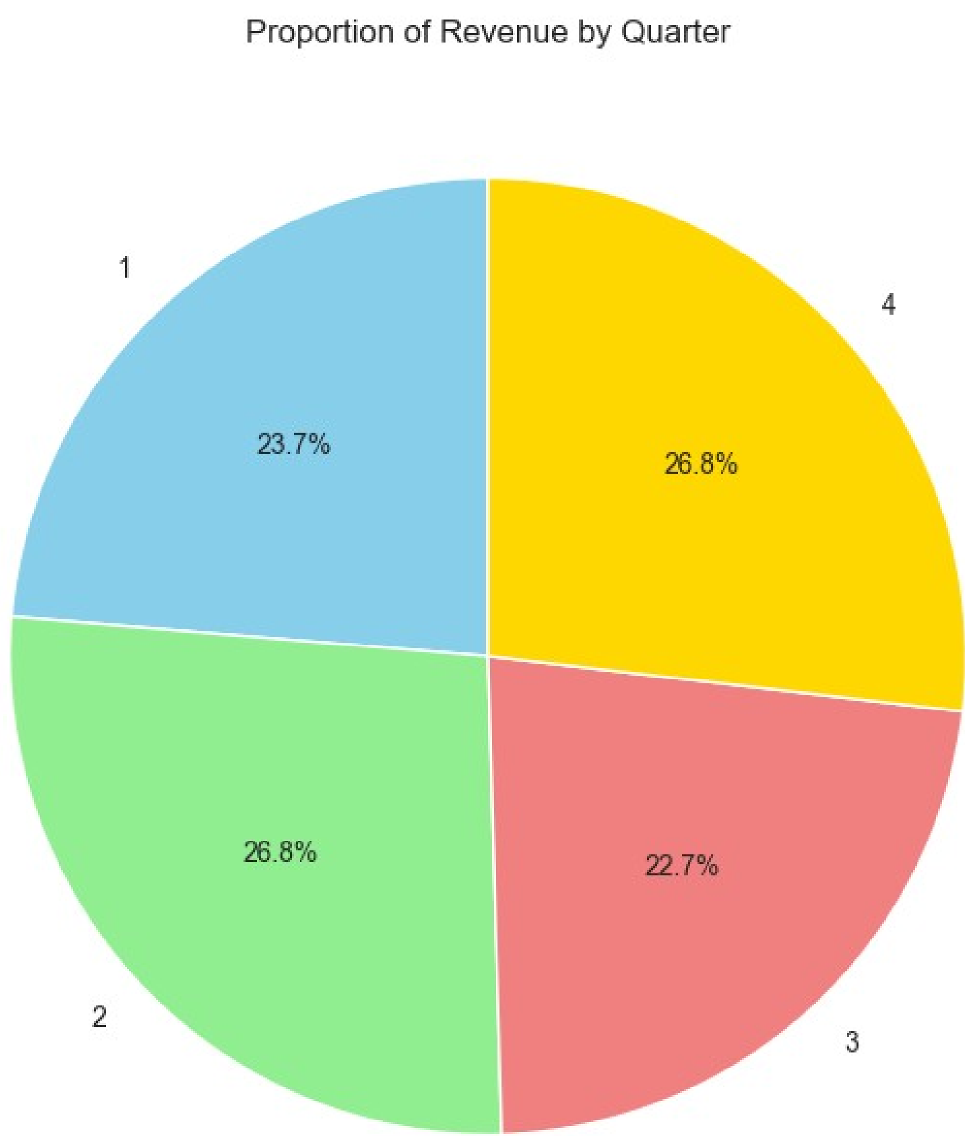
```
# Group data by month and analyse monthly revenue
# Group data by month and analyse monthly revenue
monthly_revenue = df.groupby('Month_name')
['total_amount'].sum().reindex(months_.values()).reset_index()
plt.figure(figsize=(12, 6))
plt.plot(monthly_revenue['Month_name'],
monthly_revenue['total_amount'], marker='o', linestyle='-',
color='green')
plt.title('Monthly Revenue')
plt.xlabel('Month')
plt.ylabel('Revenue')
plt.grid(True)
plt.xticks(rotation=45)
plt.show()
```



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

```
# Calculate proportion of each quarter
# lets use a pie chart for displaying quarter wise revenue
df['quarter_']=df['tpep_pickup_datetime'].dt.quarter
```

```
quarter_revenue = df.groupby('quarter_')
['total_amount'].sum().reset_index()
total_revenue = quarter_revenue['total_amount'].sum()
quarter_revenue['proportion'] = (quarter_revenue['total_amount'] /
total_revenue) * 100
#print(quarter_revenue)
plt.figure(figsize=(8, 8))
plt.pie(quarter_revenue['proportion'],
labels=quarter_revenue['quarter_'], autopct='%1.1f%%', startangle=90,
colors=['skyblue', 'lightgreen', 'lightcoral', 'gold'])
plt.title('Proportion of Revenue by Quarter')
plt.show()
```

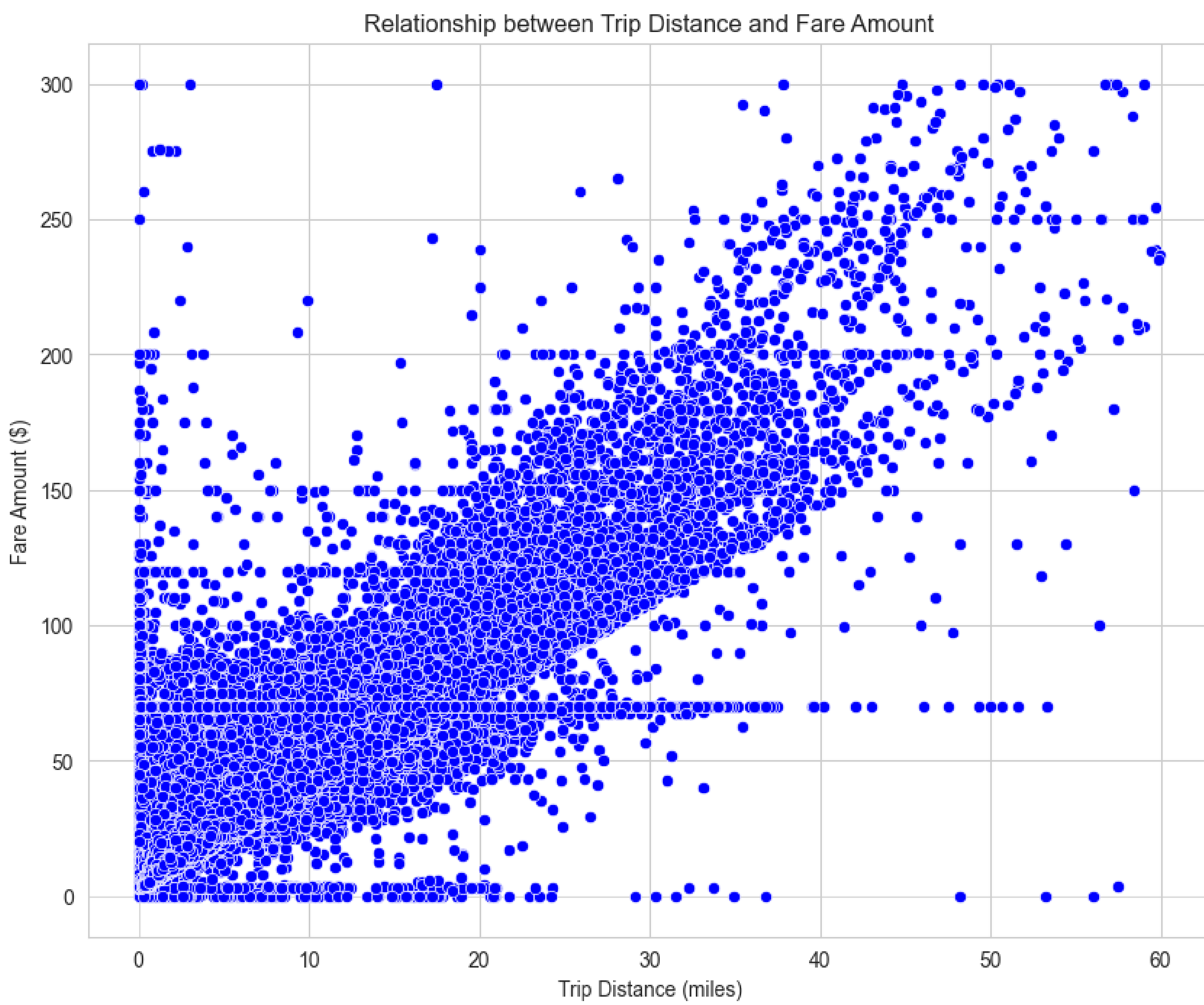


3.1.6 [3 marks] Visualise the relationship between `trip_distance` and `fare_amount`. Also find the correlation value for these two.

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

```
# Show how trip fare is affected by distance
# I have already cleaned up the data with trip_distance having 0

# we can use a scatterplot for the same
plt.figure(figsize=(10, 8))
sns.scatterplot(x='trip_distance', y='fare_amount', data=df,
color='blue')
```

```
plt.title('Relationship between Trip Distance and Fare Amount')
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Fare Amount ($)')
plt.grid(True)
plt.show()
#Calculate the correlation value
correlation = df['trip_distance'].corr(df['fare_amount'])
print(f"Correlation between Trip Distance and Fare Amount:
{correlation:.2f}")
```

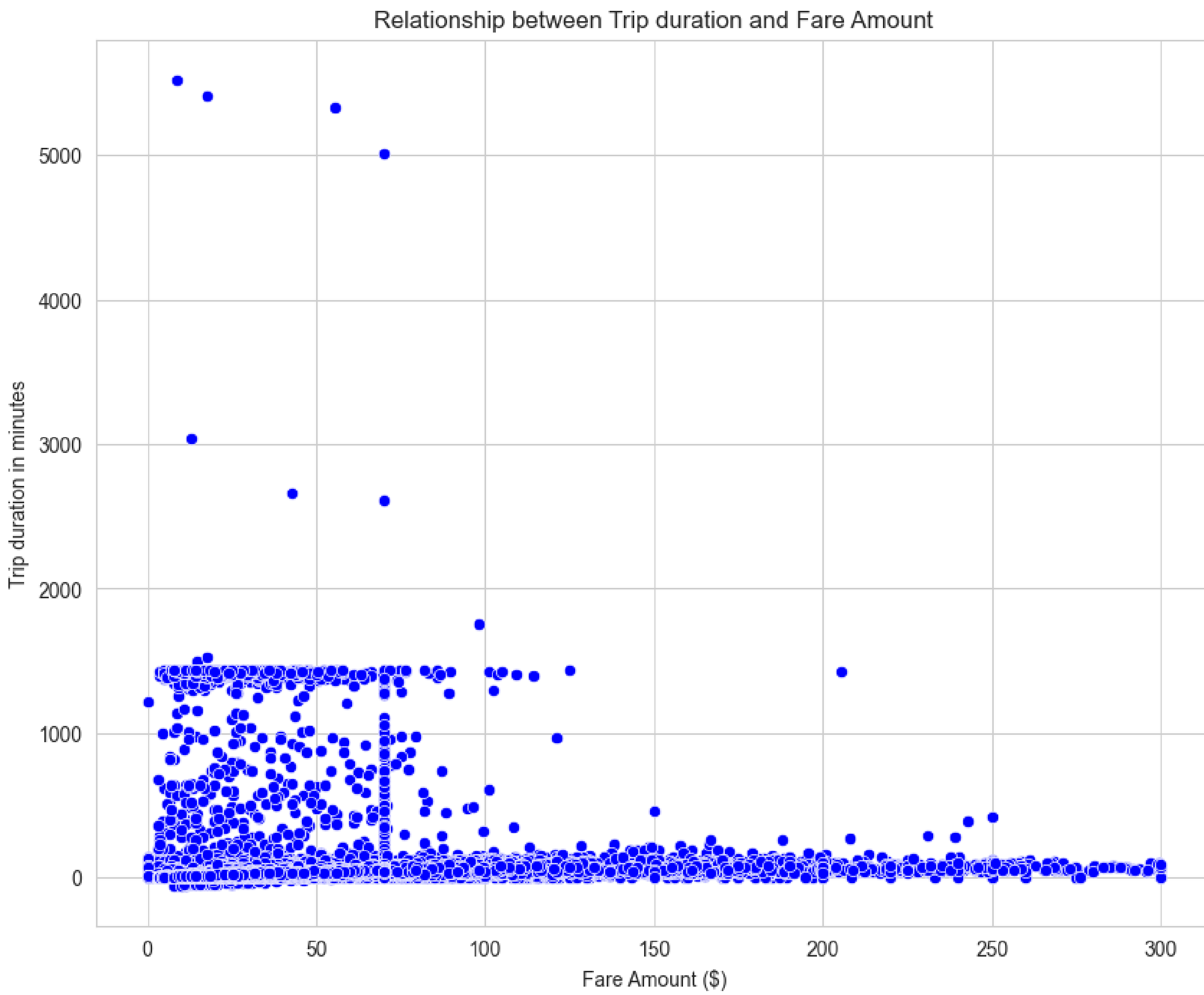



```
Correlation between Trip Distance and Fare Amount: 0.95
```

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

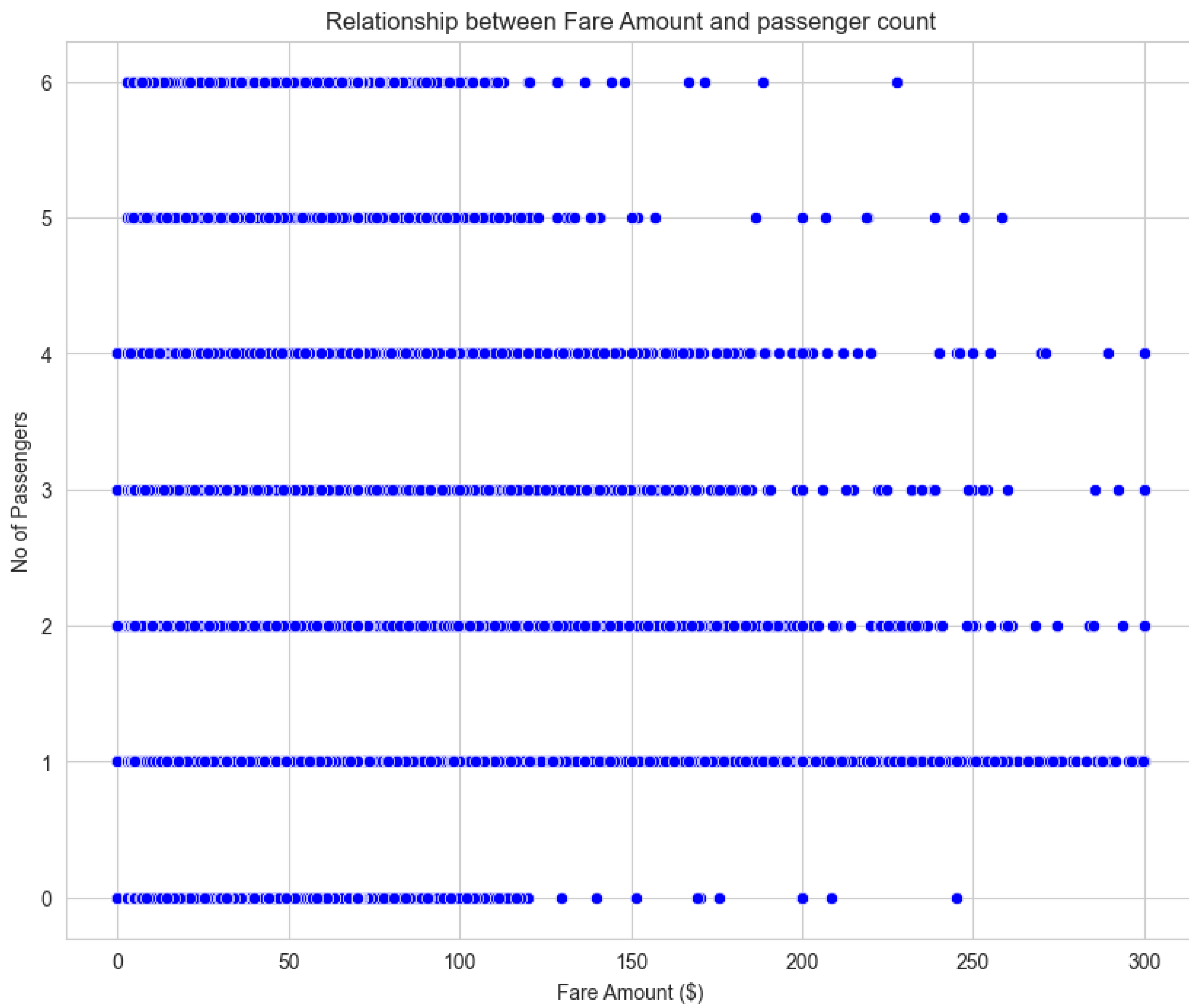
1. `fare_amount` and trip duration (pickup time to dropoff time)
2. `fare_amount` and `passenger_count`
3. `tip_amount` and `trip_distance`

```
# Show relationship between fare and trip duration
df['trip_duration'] = (df['tpep_dropoff_datetime'] -
df['tpep_pickup_datetime']).dt.total_seconds() / 60
plt.figure(figsize=(10, 8))
sns.scatterplot(x='fare_amount', y='trip_duration', data=df,
color='blue')
plt.title('Relationship between Trip duration and Fare Amount')
plt.xlabel('Fare Amount ($)')
plt.ylabel('Trip duration in minutes')
plt.grid(True)
plt.show()
#Calculate the correlation value
correlation = df['fare_amount'].corr(df['trip_duration'])
print(f"Correlation between fare Amount and trip_duration:
{correlation:.2f}")
```



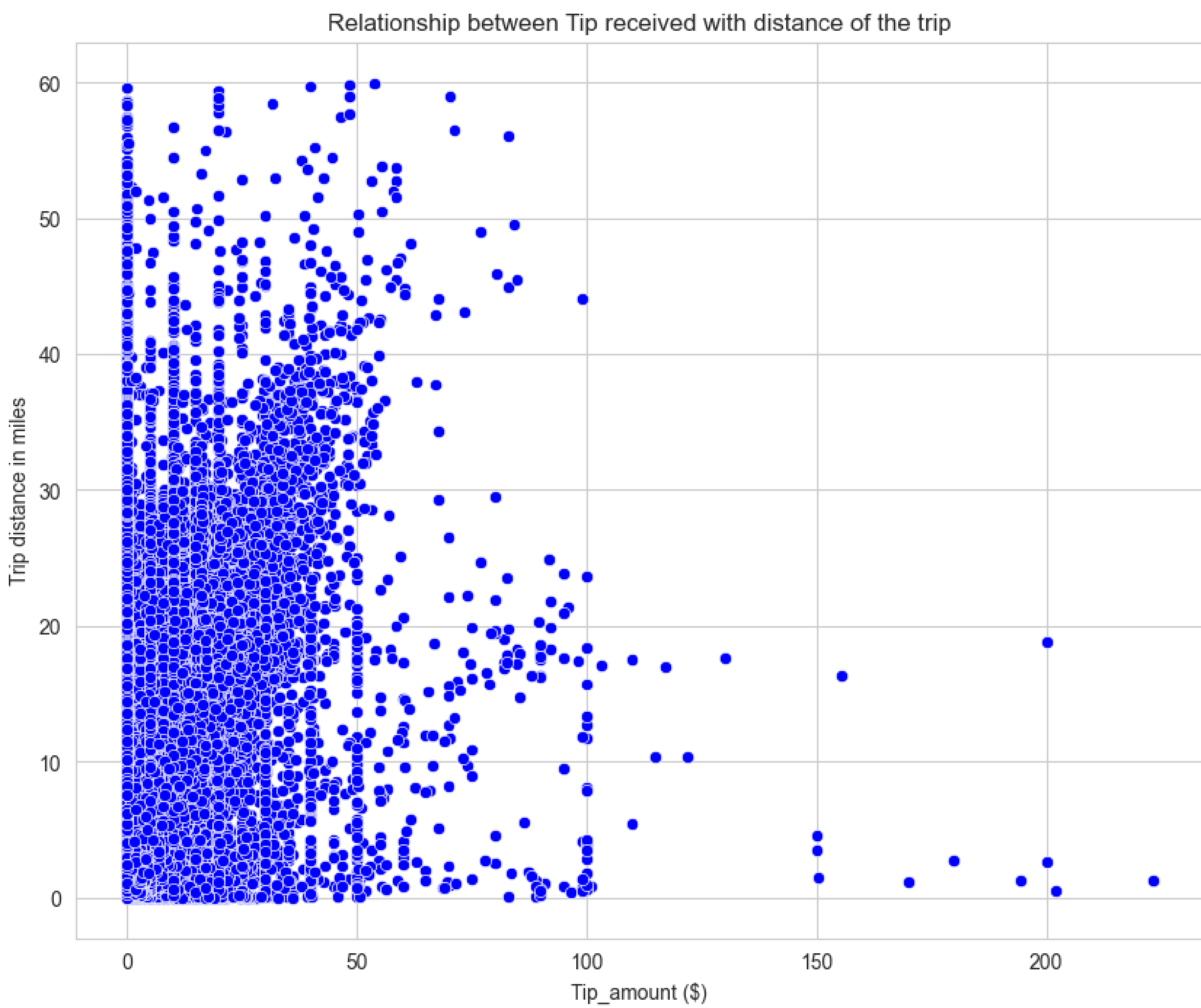

```
Correlation between fare Amount and trip_duration: 0.28
```

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

plt.figure(figsize=(10, 8))
sns.scatterplot(x='fare_amount', y='passenger_count', data=df,
color='blue')
plt.title('Relationship between Fare Amount and passenger count')
plt.xlabel('Fare Amount ($)')
plt.ylabel('No of Passengers')
plt.grid(True)
plt.show()
#Calculate the correlation value
correlation = df['fare_amount'].corr(df['passenger_count'])
print(f"Correlation between fare Amount and No of passengers :
{correlation:.2f}")
```




```
Correlation between fare Amount and No of passengers : 0.05
```

```
# Show relationship between tip and trip distance

# Show relationship between tip and trip distance
plt.figure(figsize=(10, 8))
sns.scatterplot(x='tip_amount', y='trip_distance', data=df,
color='blue')
plt.title('Relationship between Tip received with distance of the
trip')
plt.xlabel('Tip_amount ($)')
plt.ylabel('Trip distance in miles')
plt.grid(True)
plt.show()
#Calculate the correlation value
correlation = df['tip amount'].corr(df['trip_distance'])
print(f"Correlation between Tip Amount and trip_distance :
{correlation:.2f}")
```

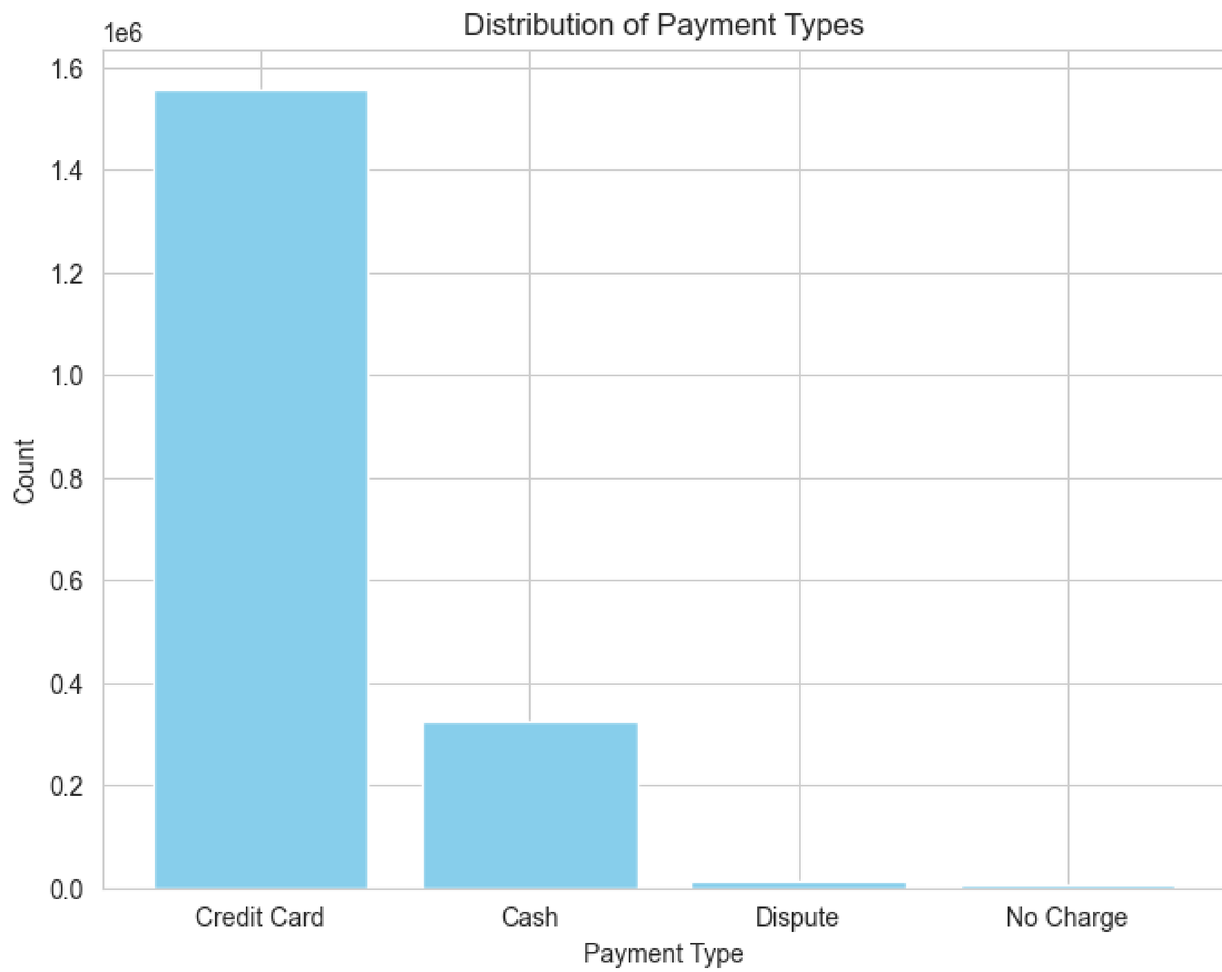


```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File /opt/anaconda3/lib/python3.11/site-packages/pandas/core/indexes/base.py:3805, in Index.get_loc(self, key)
   3804 try:
-> 3805     return self._engine.get_loc(casted_key)
   3806 except KeyError as err:

File index.pyx:167, in pandas._libs.index.IndexEngine.get_loc()

File index.pyx:196, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:7081, in pandas._libs.hashtable.PyObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:7089, in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'tip amount'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In[233], line 12
     10 plt.show()
     11 #Calculate the correlation value
---> 12 correlation = df['tip amount'].corr(df['trip_distance'])
     13 print(f"Correlation between Tip Amount and trip_distance : {correlation:.2f}")

File /opt/anaconda3/lib/python3.11/site-packages/pandas/core/frame.py:4102, in DataFrame.__getitem__(self, key)
   4100 if self.columns.nlevels > 1:
   4101     return self._getitem_multilevel(key)
-> 4102 indexer = self.columns.get_loc(key)
   4103 if is_integer(indexer):
   4104     indexer = [indexer]

File /opt/anaconda3/lib/python3.11/site-packages/pandas/core/indexes/base.py:3812, in Index.get_loc(self, key)
   3807     if isinstance(casted_key, slice) or (
   3808         isinstance(casted_key, abc.Iterable)
   3809         and any(isinstance(x, slice) for x in casted_key)
   3810     ):
   3811         raise InvalidIndexError(key)
```

```
-> 3812     raise KeyError(key) from err
   3813 except TypeError:
   3814     # If we have a listlike key, _check_indexing_error will
raise
   3815     #  InvalidIndexError. Otherwise we fall through and re-
raise
   3816     #  the TypeError.
   3817     self._check_indexing_error(key)

KeyError: 'tip amount'
```

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

```
# Analyse the distribution of different payment types (payment_type).
# Payment_type_name can be new column

Payment_name = {1: 'Credit Card',2: 'Cash',3: 'No Charge',4:
'Dispute'}
df['Payment_type_name'] = df['payment_type'].map(Payment_name)
payment_distribution =
df['Payment_type_name'].value_counts().reset_index()
payment_distribution.columns = ['Payment Type', 'Count']
plt.figure(figsize=(8, 6))
plt.bar(payment_distribution['Payment Type'],
payment_distribution['Count'], color='skyblue')
plt.title('Distribution of Payment Types')
plt.xlabel('Payment Type')
plt.ylabel('Count')
plt.grid(True)
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
/opt/anaconda3/lib/python3.11/site-packages (1.0.1)
Requirement already satisfied: numpy>=1.22 in
/opt/anaconda3/lib/python3.11/site-packages (from geopandas) (1.26.4)
Requirement already satisfied: pyogrio>=0.7.2 in
/opt/anaconda3/lib/python3.11/site-packages (from geopandas) (0.11.0)
Requirement already satisfied: packaging in
/opt/anaconda3/lib/python3.11/site-packages (from geopandas) (23.1)
Requirement already satisfied: pandas>=1.4.0 in
/opt/anaconda3/lib/python3.11/site-packages (from geopandas) (2.2.2)
Requirement already satisfied: pyproj>=3.3.0 in
/opt/anaconda3/lib/python3.11/site-packages (from geopandas) (3.7.1)
Requirement already satisfied: shapely>=2.0.0 in
/opt/anaconda3/lib/python3.11/site-packages (from geopandas) (2.1.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/opt/anaconda3/lib/python3.11/site-packages (from pandas>=1.4.0-
>geopandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/opt/anaconda3/lib/python3.11/site-packages (from pandas>=1.4.0-
>geopandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.7 in
/opt/anaconda3/lib/python3.11/site-packages (from pandas>=1.4.0-
>geopandas) (2023.3)
Requirement already satisfied: certifi in
/opt/anaconda3/lib/python3.11/site-packages (from pyogrio>=0.7.2-
>geopandas) (2024.2.2)
Requirement already satisfied: six>=1.5 in
/opt/anaconda3/lib/python3.11/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


# Read the shapefile using geopandas
zones = gpd.read_file('/Users/vaidehimallela/Downloads/Datasets and
Dictionary/taxi_zones/taxi_zones.shp')
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.

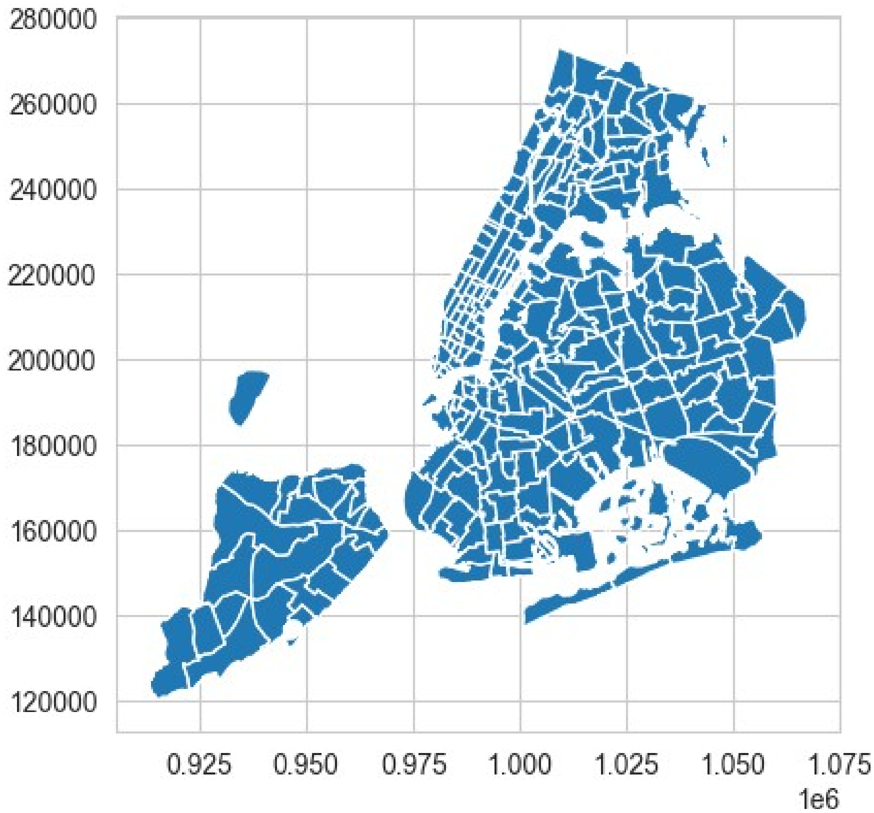
The geometric parameters like shape length, shape area and geometry are used to plot the zones on a map.

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.

```
# Merge zones and trip records using locationID and PULocationID
df_zones = pd.merge(
    zones,
    df,
    left_on='LocationID',
    right_on='PULocationID',
    how='left'
)
```

```
df_zones.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 1886357 entries, 0 to 1886356
Data columns (total 34 columns):
 #   Column                 Dtype
---  ------                 -----
 0   OBJECTID               int32
 1   Shape_Leng             float64
 2   Shape_Area             float64
 3   zone                   object
 4   LocationID             int32
 5   borough                object
 6   geometry               geometry
 7   VendorID               float64
 8   tpep_pickup_datetime   datetime64[us]
 9   tpep_dropoff_datetime  datetime64[us]
 10  passenger_count        float64
 11  trip_distance          float64
 12  RatecodeID             float64
 13  store_and_fwd_flag     object
 14  PULocationID           float64
 15  DOLocationID           float64
 16  payment_type           float64
 17  fare_amount            float64
 18  mta_tax                float64
 19  tip_amount             float64
 20  tolls_amount           float64
 21  improvement_surcharge  float64
 22  total_amount           float64
 23  congestion_surcharge   float64
 24  date                   object
 25  hour                   float64
 26  Airport__Fee           float64
 27  week_Day               float64
 28  day_name               object
 29  Month                  float64
 30  Month_name             object
 31  quarter_               float64
 32  trip_duration          float64
 33  Payment_type_name      object
dtypes: datetime64[us](2), float64(22), geometry(1), int32(2),
object(7)
memory usage: 474.9+ MB
```

3.1.11 [3 marks] Group data by location IDs to find the total number of trips per location ID

```python
# Group data by location and calculate the number of trips
pickup_trip_counts =
df_zones.groupby('PULocationID').size().reset_index(name='pickup_trip_count')
```

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.

```python
# Merge trip counts back to the zones GeoDataFrame
zones_merge=zones.merge(pickup_trip_counts, left_on='LocationID',
right_on='PULocationID', how='left')
zones_merge['pickup_trip_count'] =
zones_merge['pickup_trip_count'].fillna(0)
zones_merge = zones_merge.sort_values('pickup_trip_count',
ascending=False)
```

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:

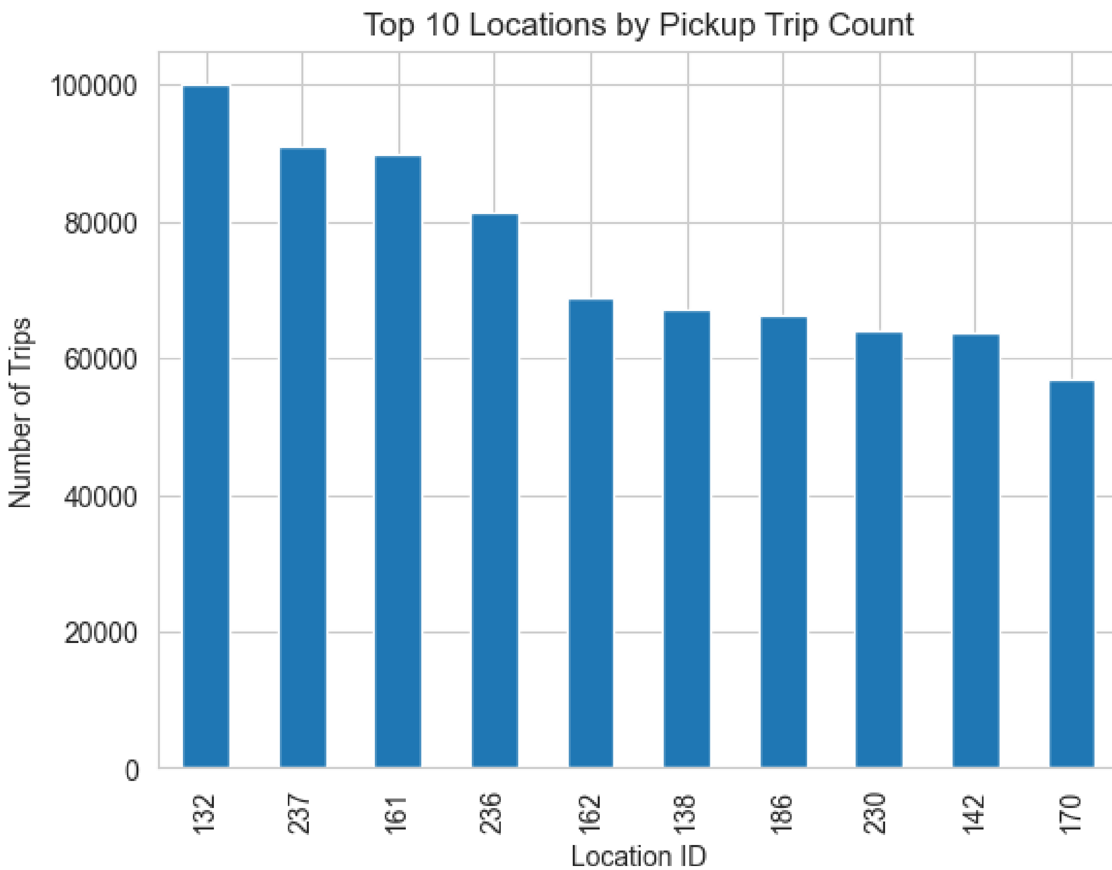
```
column = '',
ax = ax,
legend = True,
legend_kwds = {'label': "label", 'orientation':
"<horizontal/vertical>"}
```

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


# Plot the map and display it
```

```python
zones_merge.head(10).plot(kind='bar', x='LocationID',
y='pickup_trip_count', legend=False)
plt.title('Top 10 Locations by Pickup Trip Count')
plt.xlabel('Location ID')
plt.ylabel('Number of Trips')
plt.show()
```




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

zones_merge.head(20)['pickup_trip_count']
```

```
131    99922.0
236    90955.0
160    89945.0
235    81247.0
161    68724.0
137    67047.0
185    66289.0
229    64002.0
141    63721.0
169    56893.0
162    56250.0
```

```
238    52660.0
233    51720.0
47     50892.0
67     49793.0
140    45458.0
78     44950.0
163    44618.0
248    42322.0
106    40060.0
Name: pickup_trip_count, dtype: float64
```

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
slow_routes_per_hour=df_zones.groupby(['hour','PULocationID','DOLocationID'])['trip_duration'].mean().reset_index()
slow_routes_per_hour=slow_routes_per_hour.sort_values(by=['hour','trip_duration'],ascending=[True,False])
slowest_by_hour=slow_routes_per_hour.groupby('hour').head(1)
slowest_by_hour
```

```
        hour  PULocationID  DOLocationID  trip_duration
856      0.0          74.0         116.0     686.891667
9199     1.0         230.0         112.0    1431.400000
12200    2.0         144.0         100.0    1416.833333
15249    3.0         114.0         226.0    1430.633333
19686    4.0         211.0         230.0    1433.200000
21631    5.0         137.0         188.0    1420.850000
23723    6.0          70.0         138.0    1042.566667
30796    7.0         246.0          41.0    1437.383333
35345    8.0         232.0          68.0     928.500000
41689    9.0         262.0         256.0    1403.933333
46773   10.0         237.0          57.0    1432.016667
52741   11.0         238.0          51.0    1435.416667
57998   12.0         229.0          45.0    1430.933333
64177   13.0         232.0          65.0    5522.433333
68692   14.0         161.0          39.0    1432.266667
71411   15.0          22.0          22.0    1522.100000
80964   16.0         163.0         255.0    1438.216667
84585   17.0          88.0           1.0    1435.200000
93695   18.0         226.0         145.0    1810.761111
96167   19.0          70.0         203.0    1383.066667
101940  20.0          68.0          80.0    1420.083333
107488  21.0          40.0          65.0    1434.433333
118119  22.0         230.0          69.0    1425.200000
122844  23.0         148.0         258.0    1438.816667
```

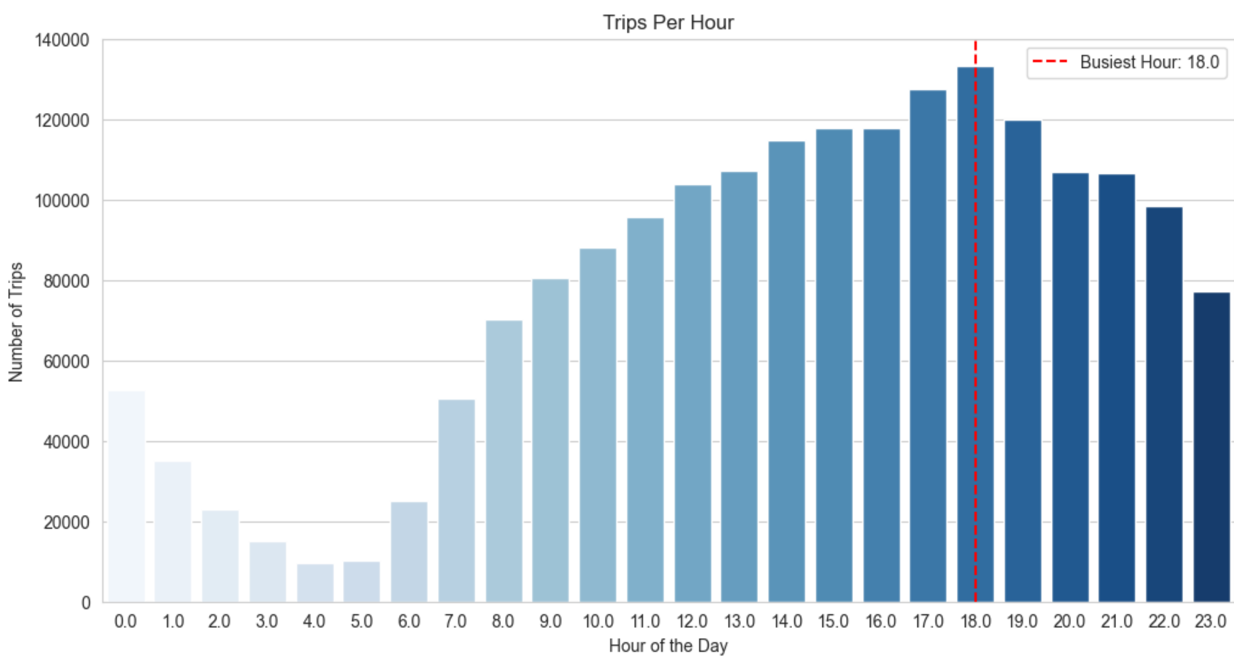
How does identifying high-traffic, high-demand routes help us?

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

trips_per_hour =
df_zones.groupby('hour').size().reset_index(name='trip_count')
busiest_hour =
trips_per_hour.loc[trips_per_hour['trip_count'].idxmax()]
print(f"Busiest hour: {busiest_hour['hour']} with
{busiest_hour['trip_count']} trips")
plt.figure(figsize=(12, 6))
sns.barplot(data=trips_per_hour, x='hour', y='trip_count',
palette="Blues")

# Highlight busiest hour
plt.axvline(busiest_hour['hour'], color='red', linestyle='--',
label=f"Busiest Hour: {busiest_hour['hour']}")

# Labels and title
plt.xlabel('Hour of the Day')
```

```
plt.ylabel('Number of Trips')
plt.title('Trips Per Hour')
plt.legend()
plt.xticks(range(24))  # Ensure all hours are visible

plt.show()
```

```
Busiest hour: 18.0 with 133334.0 trips

/var/folders/fc/k_p1s4pj2f70fysh__74y2l00000gn/T/
ipykernel_61790/2091570198.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=trips_per_hour, x='hour', y='trip_count',
palette="Blues")
```



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

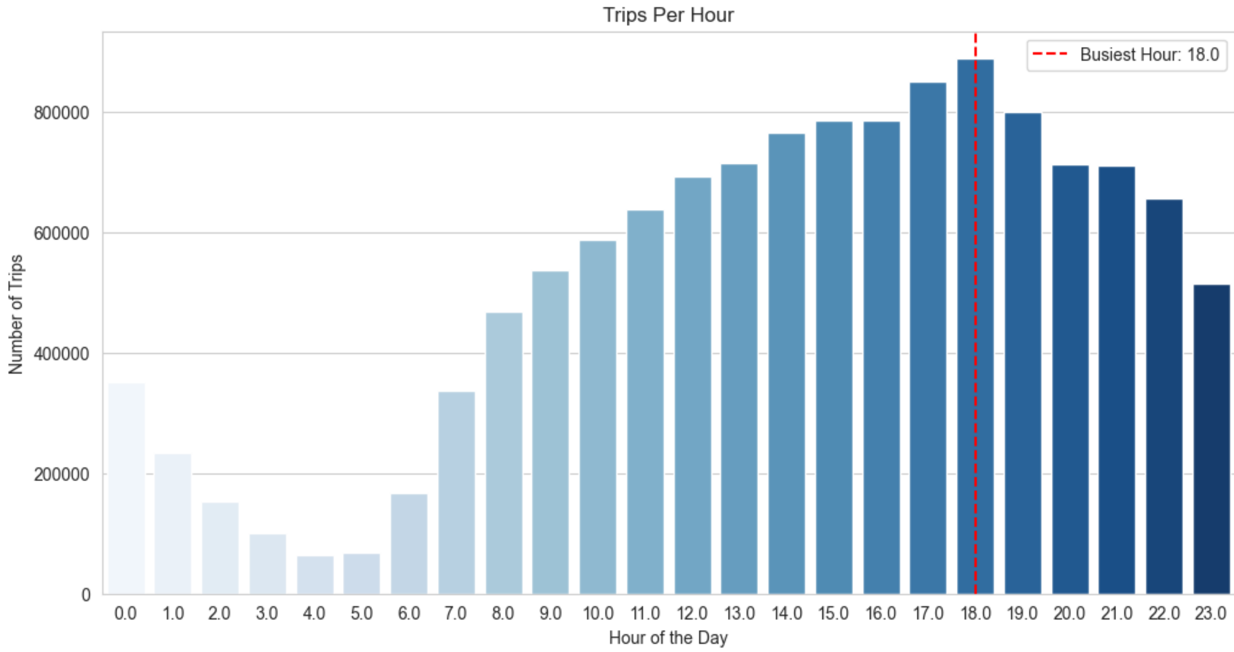
3.2.3 [2 mark] Find the actual number of trips in the five busiest hours

```
# Scale up the number of trips

# Fill in the value of your sampling fraction and use that to scale up
the numbers
sample_fraction =.15
scaling_facter =1/sample_fraction
```

```
trips_per_hour =
df_zones.groupby('hour').size().reset_index(name='trip_count')
trips_per_hour['actual_trip_count'] = trips_per_hour['trip_count'] *
scaling_facter
busiest_hour =
trips_per_hour.loc[trips_per_hour['actual_trip_count'].idxmax()]
print(f"Busiest hour: {busiest_hour['hour']} with
{busiest_hour['actual_trip_count']} trips")
plt.figure(figsize=(12, 6))
sns.barplot(data=trips_per_hour, x='hour', y='actual_trip_count',
palette="Blues")

plt.axvline(busiest_hour['hour'], color='red', linestyle='--',
label=f"Busiest Hour: {busiest_hour['hour']}")

# Labels and title
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Trips')
plt.title('Trips Per Hour')
plt.legend()
plt.xticks(range(24))

plt.show()
```

```
Busiest hour: 18.0 with 888893.3333333334 trips
```

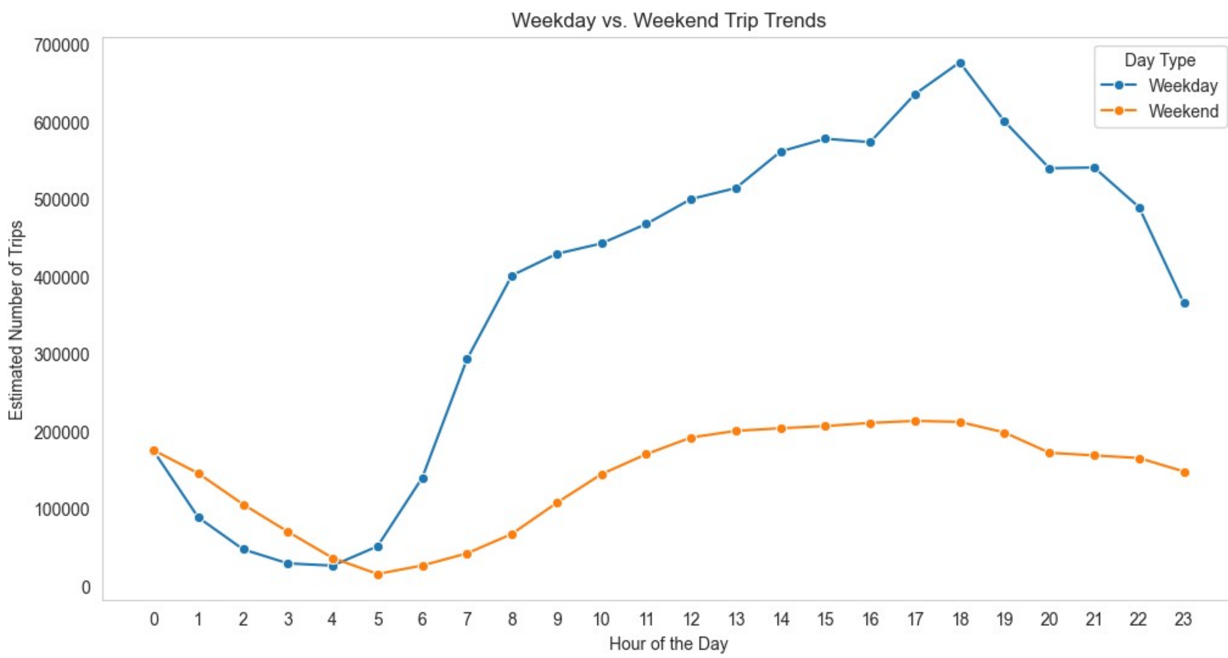
```
/var/folders/fc/k_p1s4pj2f70fysh__74y2l00000gn/T/
ipykernel_61790/2837954313.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=trips_per_hour, x='hour', y='actual_trip_count',
palette="Blues")
```



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

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

sample_fraction =.15
scaling_facter =1/sample_fraction
df_zones['day_type']=df_zones['tpep_pickup_datetime'].dt.weekday.apply
(lambda x: 'Weekend' if x >= 5 else 'Weekday')
trips_per_hour =
df_zones.groupby(['day_type','hour']).size().reset_index(name='trip_co
unt')
trips_per_hour['actual_trip_count'] = trips_per_hour['trip_count'] *
scaling_facter

busiest_weekday = trips_per_hour[trips_per_hour['day_type'] ==
'Weekday'].nlargest(1, 'actual_trip_count')
busiest_weekend = trips_per_hour[trips_per_hour['day_type'] ==
'Weekend'].nlargest(1, 'actual_trip_count')

print(f"Busiest Weekday Hour: {busiest_weekday['hour'].values[0]} with
approx. {int(busiest_weekday['actual_trip_count'].values[0])} trips")
print(f"Busiest Weekend Hour: {busiest_weekend['hour'].values[0]} with
approx. {int(busiest_weekend['actual_trip_count'].values[0])} trips")

plt.figure(figsize=(12, 6))
sns.lineplot(data=trips_per_hour, x='hour', y='actual_trip_count',
hue='day_type', marker='o')
plt.xlabel('Hour of the Day')
plt.ylabel('Estimated Number of Trips')
plt.title('Weekday vs. Weekend Trip Trends')
```

```
plt.legend(title="Day Type")
plt.xticks(range(24))
plt.grid()
plt.show()

Busiest Weekday Hour: 18.0 with approx. 676593 trips
Busiest Weekend Hour: 17.0 with approx. 213586 trips
```



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

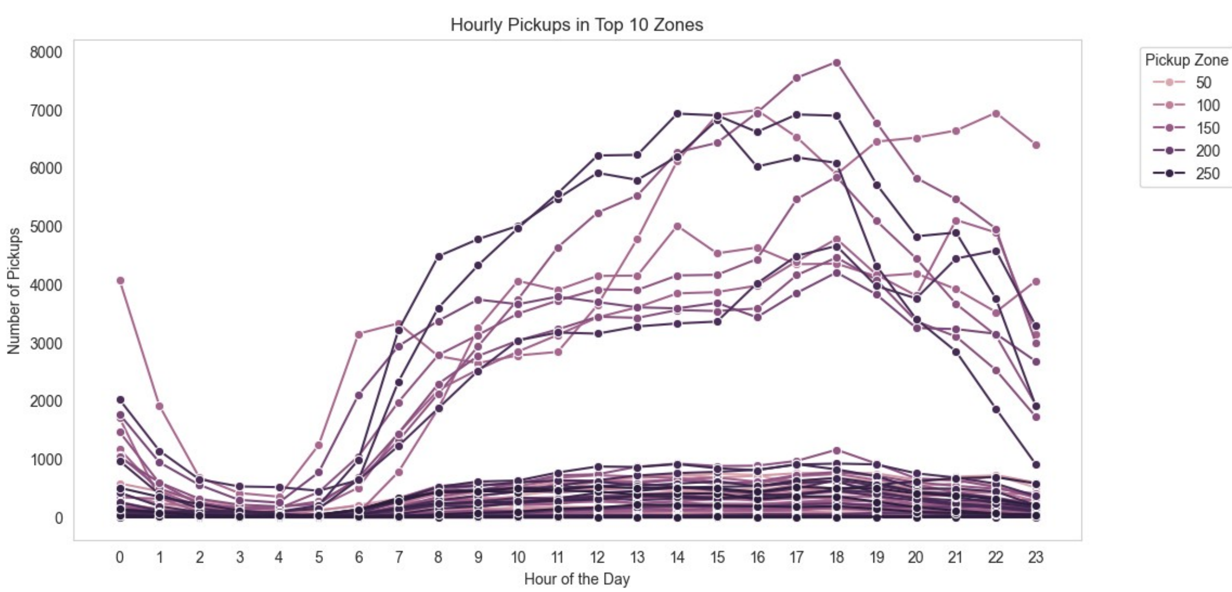
3.2.5 [3 marks] Identify top 10 zones with high hourly pickups. Do the same for hourly dropoffs. Show pickup and dropoff trends in these zones.

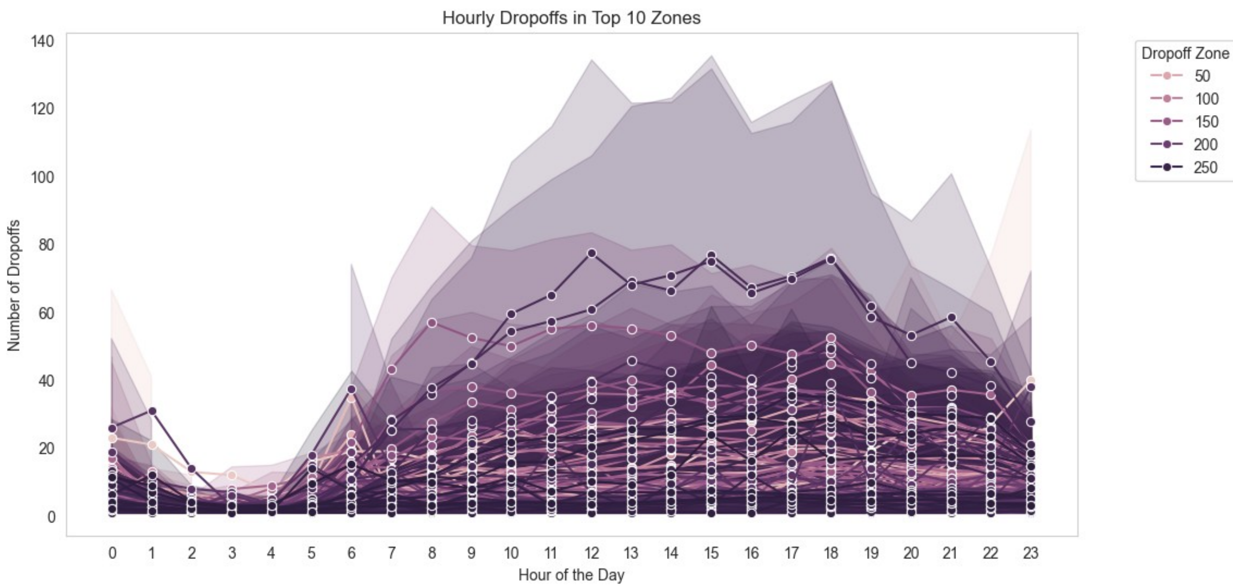
```
# Find top 10 pickup and dropoff zones

df_pickups= df.merge(zones, left_on="PULocationID",
right_on="LocationID", how="left")
pickup_counts =
df_pickups.groupby(['PULocationID','zone']).size().reset_index(name='p
ickup_count')
top_10_pickup_zones = pickup_counts.nlargest(10, 'pickup_count')
print(top_10_pickup_zones)
top_10_pickup_zones_l =pickup_counts.nlargest(10, 'pickup_count')
['PULocationID'].tolist()
#print("\n",top_10_pickup_zones_l)
df_dropoff= df.merge(zones, left_on="DOLocationID",
right_on="LocationID", how="left")
dropoff_counts =
```

```python
df_pickups.groupby(['DOLocationID','zone']).size().reset_index(name='dropoff_count')
top_10_droppoff_zones = dropoff_counts.nlargest(10, 'dropoff_count')
top_10_droppoff_zones_l= dropoff_counts.nlargest(10, 'dropoff_count')
['DOLocationID'].tolist()
print(top_10_droppoff_zones)
#print("\n",top_10_droppoff_zones_l)


filtered_ =
df_pickups[(df_pickups['PULocationID'].isin(top_10_pickup_zones_l)) |
(df_dropoff['DOLocationID'].isin(top_10_droppoff_zones_l))]
hourly_pickups = filtered_.groupby(['hour',
'PULocationID','zone']).size().reset_index(name='pickup_count')
hourly_dropoffs = filtered_.groupby(['hour',
'DOLocationID','zone']).size().reset_index(name='dropoff_count')

plt.figure(figsize=(12, 6))
sns.lineplot(data=hourly_pickups, x='hour', y='pickup_count',
hue='PULocationID', marker='o')

plt.xlabel('Hour of the Day')
plt.ylabel('Number of Pickups')
plt.title('Hourly Pickups in Top 10 Zones')
plt.legend(title="Pickup Zone", bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.xticks(range(24))
plt.grid()
plt.show()

plt.figure(figsize=(12, 6))
sns.lineplot(data=hourly_dropoffs, x='hour', y='dropoff_count',
hue='DOLocationID', marker='o')

plt.xlabel('Hour of the Day')
plt.ylabel('Number of Dropoffs')
plt.title('Hourly Dropoffs in Top 10 Zones')
plt.legend(title="Dropoff Zone", bbox_to_anchor=(1.05, 1), loc='upper
left')
plt.xticks(range(24))
plt.grid()
plt.show()
```

```
     PULocationID                           zone  pickup_count
123           132                    JFK Airport         99922
226           237          Upper East Side South         90955
152           161                 Midtown Center         89945
225           236          Upper East Side North         81247
153           162                   Midtown East         68724
129           138              LaGuardia Airport         67047
```

```
176            186  Penn Station/Madison Sq West         66289
219            230     Times Sq/Theatre District         64002
133            142           Lincoln Square East         63721
161            170                   Murray Hill         56893
       DOLocationID                     zone  dropoff_count
16369           236  Upper East Side South           13239
16479           237  Upper East Side North           11435
16480           237  Upper East Side South            8909
16368           236  Upper East Side North            8631
16444           237         Midtown Center            6355
11029           161  Upper East Side South            6036
16332           236         Midtown Center            5404
16649           239    Lincoln Square East            5198
9700            142  Upper West Side South            4849
15637           230            JFK Airport            4797

/var/folders/fc/k_p1s4pj2f70fysh__74y2l00000gn/T/
ipykernel_61790/2995816483.py:17: UserWarning: Boolean Series key will
be reindexed to match DataFrame index.
  filtered_ =
df_pickups[(df_pickups['PULocationID'].isin(top_10_pickup_zones_l)) |
(df_dropoff['DOLocationID'].isin(top_10_droppoff_zones_l))]
```





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
ickup_counts =
df_pickups.groupby(['PULocationID','zone']).size().reset_index(name='p
ickup_count')
#print(pickup_counts)
dropoff_counts =
df_pickups.groupby(['DOLocationID','zone']).size().reset_index(name='d
ropoff_count')
zone_counts = pickup_counts.merge(dropoff_counts, left_on="zone",
right_on="zone", how="outer").fillna(0)
#print(zone_counts)
#zone_counts["dropoff_count"] =
zone_counts["dropoff_count"].replace(0, 1)
zone_counts["pickup_drop_ratio"] = zone_counts["pickup_count"] /
zone_counts["dropoff_count"]

top_10_highest = zone_counts.nlargest(10, "pickup_drop_ratio")
top_10_lowest = zone_counts.nsmallest(10, "pickup_drop_ratio")

print("Top 20 Highest Pickup/Dropoff Ratios:")
print(top_10_highest[["zone", "pickup_count", "dropoff_count",
"pickup_drop_ratio"]])

print("\nTop 20 Lowest Pickup/Dropoff Ratios:")
print(top_10_lowest[["zone", "pickup_count", "dropoff_count",
"pickup_drop_ratio"]])
```

```
Top 20 Highest Pickup/Dropoff Ratios:
                        zone  pickup_count  dropoff_count
```

```
pickup_drop_ratio
7903             JFK Airport         99922              1
99922.0
16367  Upper East Side South         90955              1
90955.0
16370  Upper East Side South         90955              1
90955.0
16374  Upper East Side South         90955              1
90955.0
16376  Upper East Side South         90955              1
90955.0
16378  Upper East Side South         90955              1
90955.0
16387  Upper East Side South         90955              1
90955.0
16388  Upper East Side South         90955              1
90955.0
16401  Upper East Side South         90955              1
90955.0
16405  Upper East Side South         90955              1
90955.0

Top 20 Lowest Pickup/Dropoff Ratios:
                                       zone  pickup_count
dropoff_count  \
1187   Breezy Point/Fort Tilden/Riis Beach             1
1
3136                           Crotona Park             2
2
5149     Eltingville/Annadale/Prince's Bay             1
1
7174                    Grymes Hill/Clifton             1
1
10531                       Mariners Harbor             1
1
12206                 New Dorp/Midland Beach             1
1
12866                          Port Richmond             2
2
14105               South Beach/Dongan Hills             8
7
12207                         Newark Airport            48
39
17700                          Willets Point             4
3

       pickup_drop_ratio
1187            1.000000
3136            1.000000
```

```
5149             1.000000
7174             1.000000
10531            1.000000
12206            1.000000
12866            1.000000
14105            1.142857
12207            1.230769
17700            1.333333
```

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_pickups_dropoff= df.merge(zones, left_on="PULocationID",
right_on="LocationID", how="left").rename(columns={"zone":
"pickup_zone"})
df_pickups_dropoff= df_pickups_dropoff.merge(zones,
left_on="DOLocationID", right_on="LocationID",
how="left").rename(columns={"zone": "dropoff_zone"})
night_hours = df_pickups_dropoff[(df_pickups_dropoff['hour'] >= 23) |
(df_pickups_dropoff['hour'] <= 5)]
night_pickup_counts =
night_hours.groupby("pickup_zone").size().reset_index(name="pickup_cou
nt")
night_dropoff_counts =
night_hours.groupby("dropoff_zone").size().reset_index(name="dropoff_c
ount")

top_10_night_pickups = night_pickup_counts.nlargest(10,
"pickup_count")
top_10_night_dropoffs = night_dropoff_counts.nlargest(10,
"dropoff_count")
print("Top 10 Pickup Zones (Night Hours):")
print(top_10_night_pickups)
print("\nTop 10 Dropoff Zones (Night Hours):")
print(top_10_night_dropoffs)
```

```
Top 10 Pickup Zones (Night Hours):
                      pickup_zone  pickup_count
69                   East Village         16260
108                   JFK Airport         15121
217                  West Village         13044
41                   Clinton East         10932
127               Lower East Side         10078
96        Greenwich Village South          9212
```

```
199     Times Sq/Theatre District         8584
160  Penn Station/Madison Sq West         7248
141               Midtown South           6398
117            LaGuardia Airport          6278

Top 10 Dropoff Zones (Night Hours):
                      dropoff_zone  dropoff_count
75                    East Village           8661
45                    Clinton East           7118
162                    Murray Hill           6525
64                    East Chelsea           6061
100                       Gramercy           5989
133                Lenox Hill West           5509
254                 Yorkville West           5163
240                   West Village           5138
221      Times Sq/Theatre District           4819
220  Sutton Place/Turtle Bay North           4586
```

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_trip_time = df_zones[(df_zones['hour'] >= 23) |
(df_zones['hour'] <= 5)]
day_trip_time =df_zones[(df_zones['hour'] >= 5) & (df_zones['hour'] <
23)]
nighttime_revenue = night_trip_time['fare_amount'].sum()
daytime_revenue = day_trip_time['fare_amount'].sum()
total_revenue = nighttime_revenue + daytime_revenue
night_trip_revenue_share = (nighttime_revenue / total_revenue) * 100
day_trip_revenue_share = (daytime_revenue / total_revenue) * 100

print(f"Nighttime Revenue Share: {night_trip_revenue_share:.2f}%")
print(f"Daytime Revenue Share: {day_trip_revenue_share:.2f}%")

Nighttime Revenue Share: 11.99%
Daytime Revenue Share: 88.01%
```

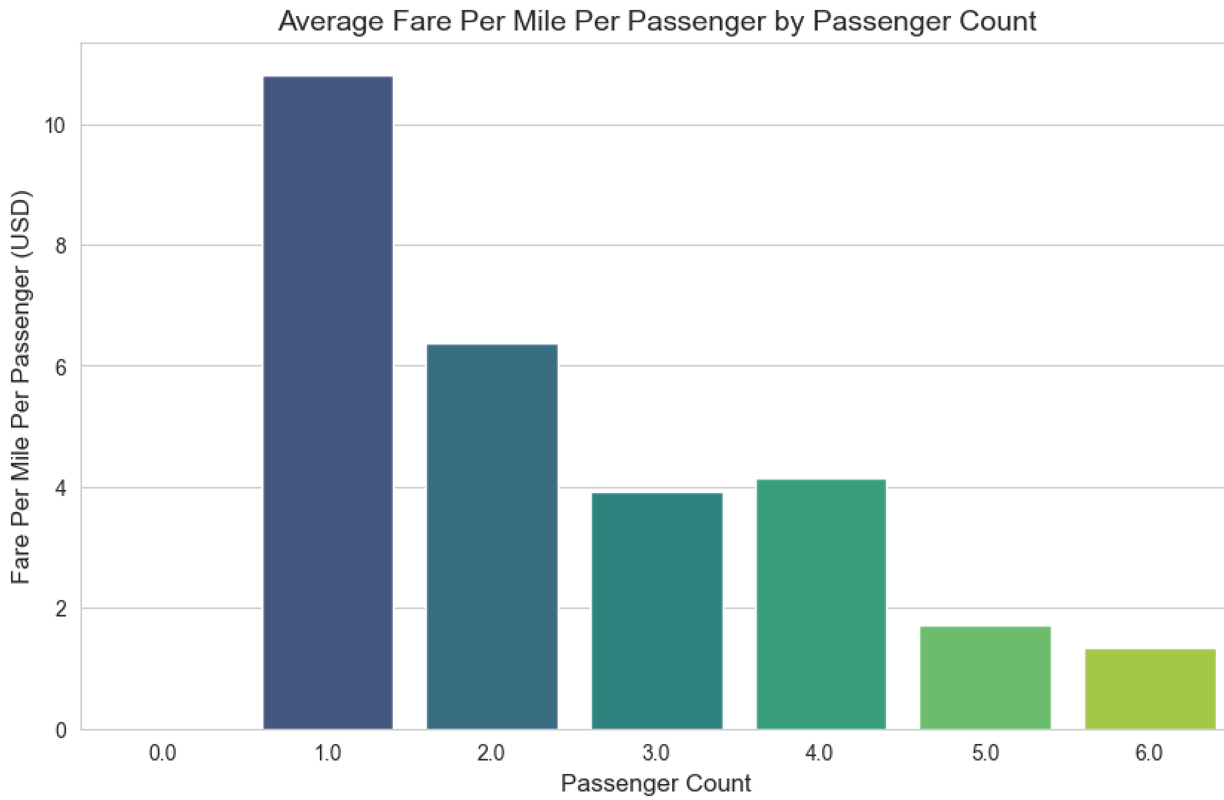
Pricing Strategy

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.

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

df_zones['fare_per_mile'] = df_zones['fare_amount'] /
df_zones['trip_distance']
avg_fare_per_mile = df_zones.groupby('passenger_count')
['fare_per_mile'].mean().reset_index()
avg_fare_per_mile['fare_per_mile_per_passenger'] =
avg_fare_per_mile['fare_per_mile'] /
avg_fare_per_mile['passenger_count']
print(avg_fare_per_mile)
sns.set_style("whitegrid")
plt.figure(figsize=(10, 6))
sns.barplot(data=avg_fare_per_mile, x="passenger_count",
y="fare_per_mile_per_passenger", palette="viridis")
plt.xlabel("Passenger Count", fontsize=12)
plt.ylabel("Fare Per Mile Per Passenger (USD)", fontsize=12)
plt.title("Average Fare Per Mile Per Passenger by Passenger Count",
fontsize=14)
plt.xticks(rotation=0)
plt.show()
```

```
   passenger_count  fare_per_mile  fare_per_mile_per_passenger
0              0.0       8.776886                          inf
1              1.0      10.808725                    10.808725
2              2.0      12.767778                     6.383889
3              3.0      11.750506                     3.916835
4              4.0      16.531068                     4.132767
5              5.0       8.533099                     1.706620
6              6.0       8.098307                     1.349718

/var/folders/fc/k_p1s4pj2f70fysh__74y2l00000gn/T/
ipykernel_61790/4144843231.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=avg_fare_per_mile, x="passenger_count",
y="fare_per_mile_per_passenger", palette="viridis")
```

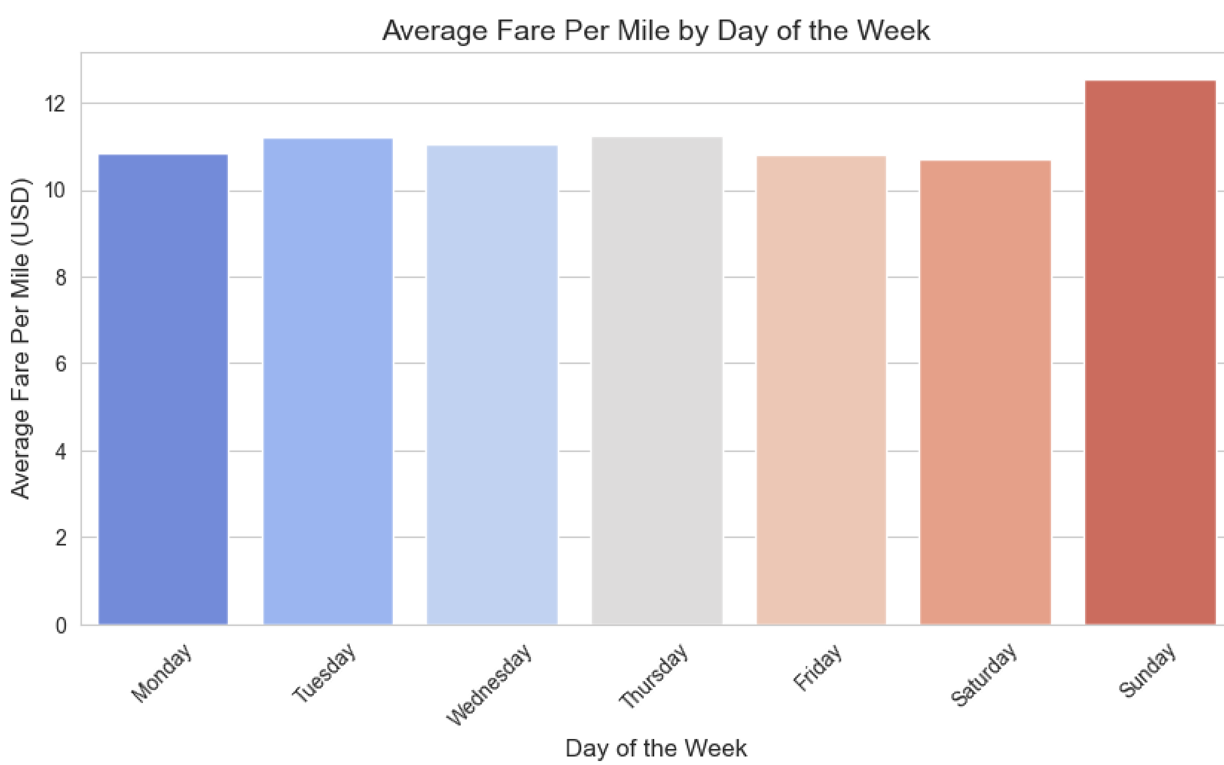


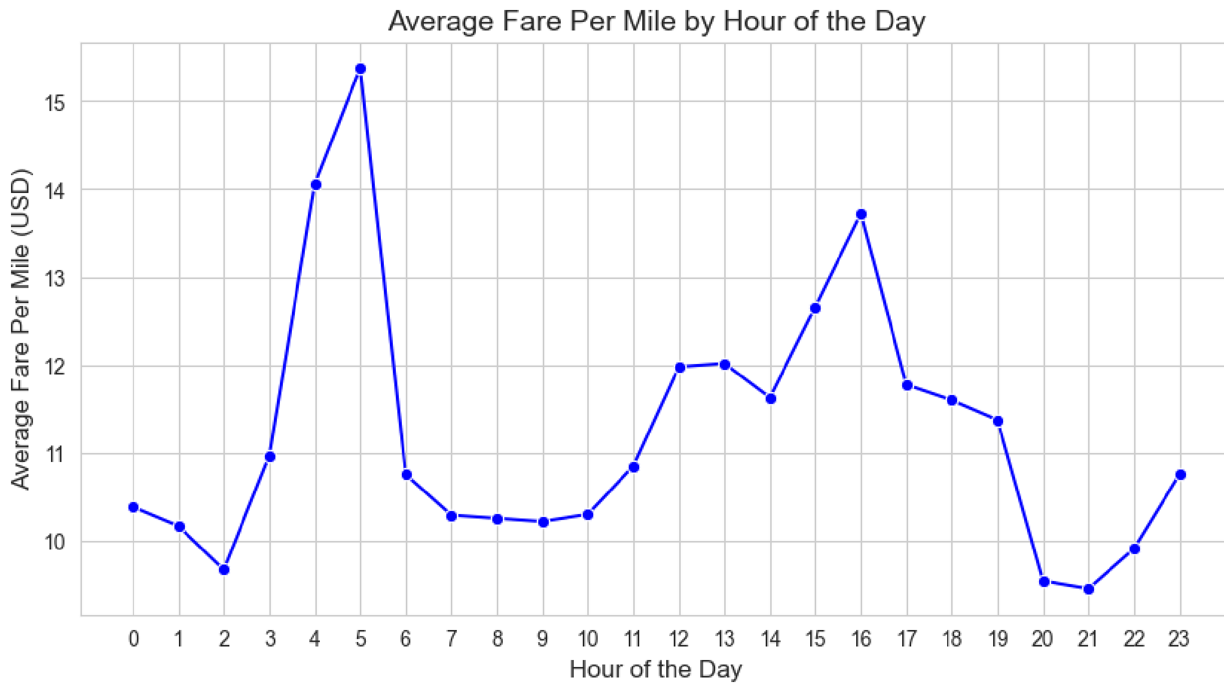
3.2.10 [3 marks] Find the average fare per mile by hours of the day and by days of the week

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

avg_fare_per_mile_by_day = df_zones.groupby("week_Day")
["fare_per_mile"].mean().reset_index()
avg_fare_per_mile_by_day["week_Day"]=avg_fare_per_mile_by_day["week_Da
y"].astype(int)
days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday", "Sunday"]
avg_fare_per_mile_by_day["week_Day"] =
avg_fare_per_mile_by_day["week_Day"].map(lambda x: days[x])
plt.figure(figsize=(10, 5))
sns.barplot(data=avg_fare_per_mile_by_day, x="week_Day",
y="fare_per_mile", palette="coolwarm")

# Labels & Title
plt.xlabel("Day of the Week", fontsize=12)
plt.ylabel("Average Fare Per Mile (USD)", fontsize=12)
plt.title("Average Fare Per Mile by Day of the Week", fontsize=14)
plt.xticks(rotation=45)


avg_fare_per_mile_by_hour =df_zones.groupby("hour")
```

```
["fare_per_mile"].mean().reset_index()
plt.figure(figsize=(10, 5))
sns.lineplot(data=avg_fare_per_mile_by_hour, x="hour",
y="fare_per_mile", marker="o", color="b")

# Labels & Title
plt.xlabel("Hour of the Day", fontsize=12)
plt.ylabel("Average Fare Per Mile (USD)", fontsize=12)
plt.title("Average Fare Per Mile by Hour of the Day", fontsize=14)
plt.xticks(range(0, 24))

plt.show()
```

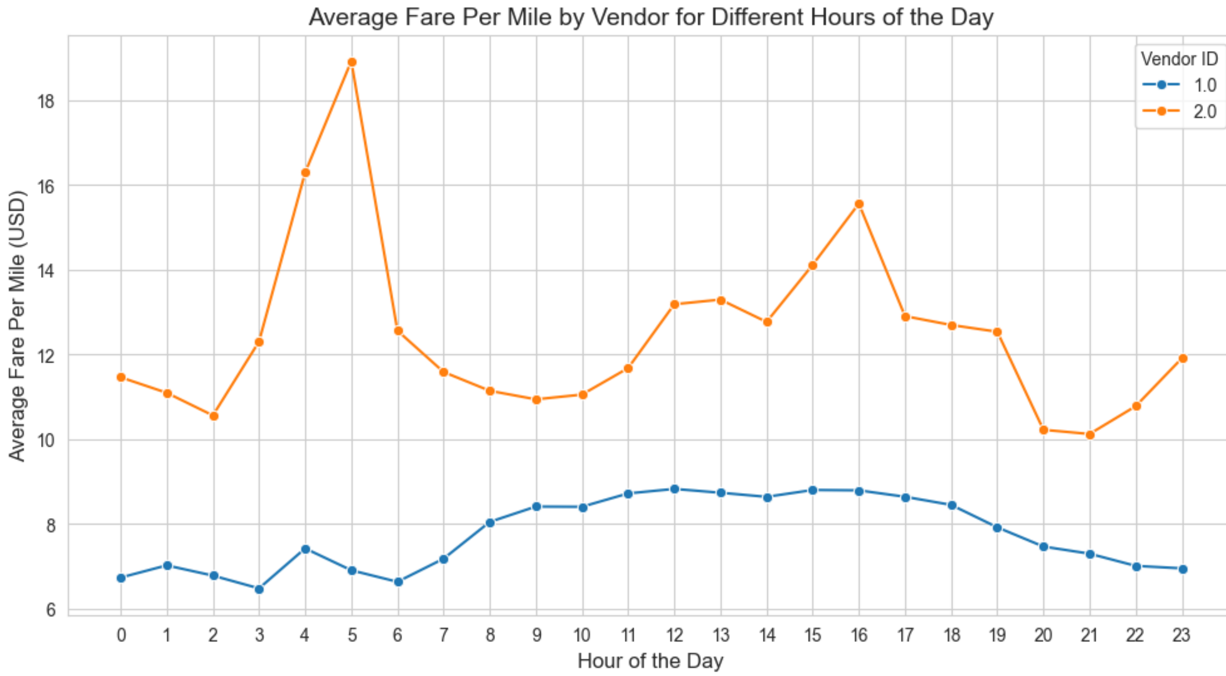
```
/var/folders/fc/k_p1s4pj2f70fysh__74y2l00000gn/T/
ipykernel_61790/574122737.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=avg_fare_per_mile_by_day, x="week_Day",
y="fare_per_mile", palette="coolwarm")
```





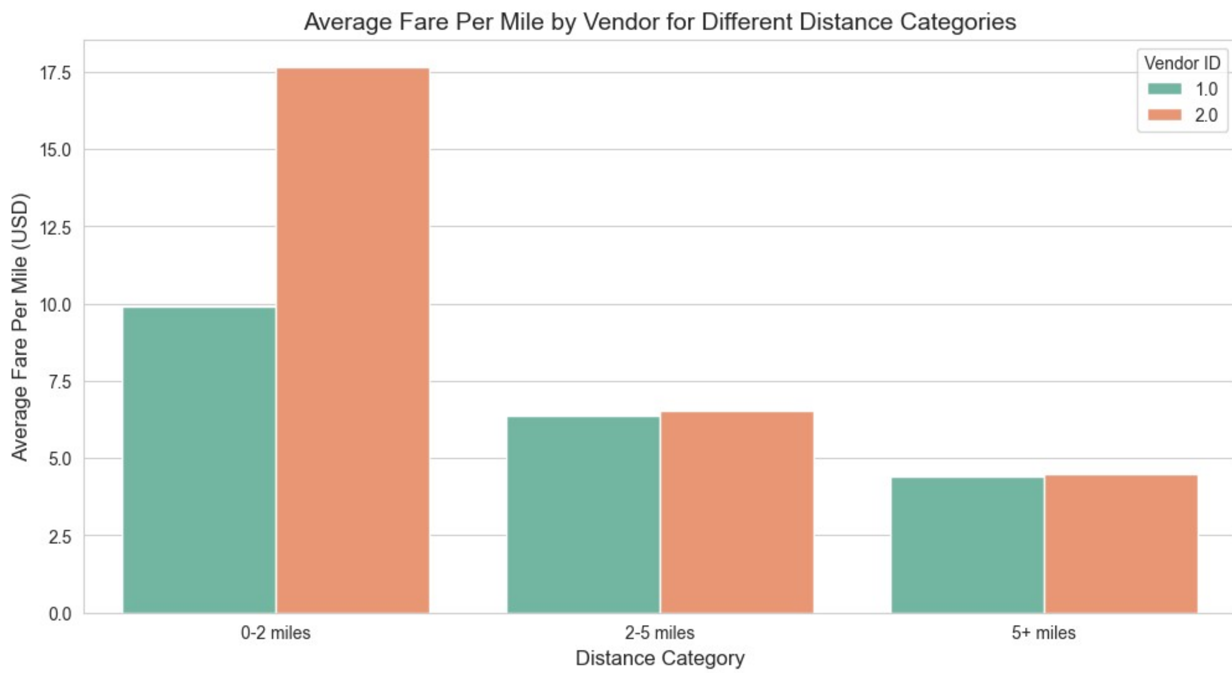
3.2.11 [3 marks] Analyse the average fare per mile for the different vendors for different hours of the day

```
# Compare fare per mile for different vendors
avg_fare_per_mile_by_vendor = df_zones.groupby(["VendorID", "hour"])
["fare_per_mile"].mean().reset_index()
sns.set_style("whitegrid")

plt.figure(figsize=(12, 6))
sns.lineplot(data=avg_fare_per_mile_by_vendor, x="hour",
y="fare_per_mile", hue="VendorID", marker="o", palette="tab10")

plt.xlabel("Hour of the Day", fontsize=12)
plt.ylabel("Average Fare Per Mile (USD)", fontsize=12)
plt.title("Average Fare Per Mile by Vendor for Different Hours of the
Day", fontsize=14)
plt.xticks(range(0, 24))

plt.legend(title="Vendor ID")

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
def distance_category(dist):
    if dist <= 2:
        return "0-2 miles"
    elif 2 < dist <= 5:
        return "2-5 miles"
    else:
        return "5+ miles"
df_zones['distance_category'] =
df_zones['trip_distance'].apply(distance_category)
avg_fare_by_vendor = df_zones.groupby(["VendorID",
"distance_category"])["fare_per_mile"].mean().reset_index()
sns.set_style("whitegrid")

plt.figure(figsize=(12, 6))
sns.barplot(data=avg_fare_by_vendor, x="distance_category",
y="fare_per_mile", hue="VendorID", palette="Set2")
plt.xlabel("Distance Category", fontsize=12)
plt.ylabel("Average Fare Per Mile (USD)", fontsize=12)
plt.title("Average Fare Per Mile by Vendor for Different Distance
Categories", fontsize=14)
plt.legend(title="Vendor ID")
plt.show()
```



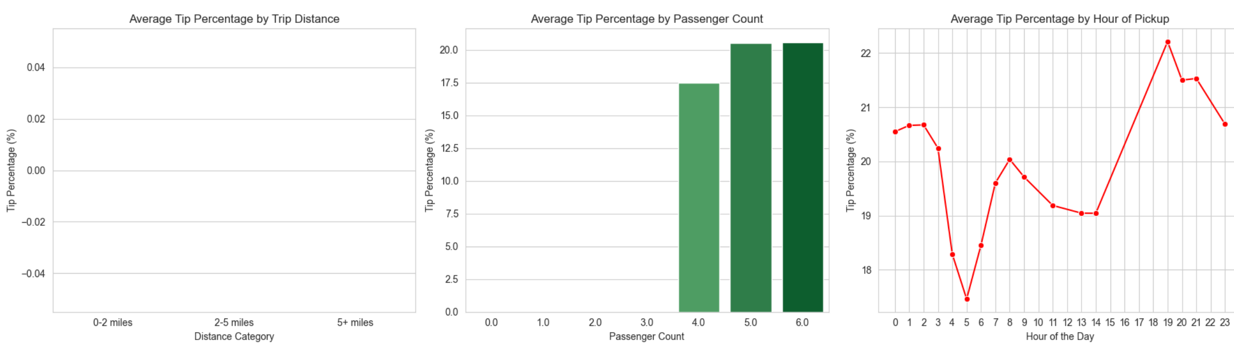
Customer Experience and Other Factors

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_zones['tip_percentage'] = (df_zones['tip_amount'] /
df_zones['fare_amount']) * 100
avg_tip_by_distance = df_zones.groupby("distance_category")
["tip_percentage"].mean().reset_index()
avg_tip_by_passenger = df_zones.groupby("passenger_count")
["tip_percentage"].mean().reset_index()
avg_tip_by_hour = df_zones.groupby("hour")
["tip_percentage"].mean().reset_index()
sns.set_style("whitegrid")

fig, axes = plt.subplots(1, 3, figsize=(18, 5))

sns.barplot(data=avg_tip_by_distance, x="distance_category",
y="tip_percentage", palette="Blues", ax=axes[0])
axes[0].set_title("Average Tip Percentage by Trip Distance")
axes[0].set_xlabel("Distance Category")
axes[0].set_ylabel("Tip Percentage (%)")

sns.barplot(data=avg_tip_by_passenger, x="passenger_count",
y="tip_percentage", palette="Greens", ax=axes[1])
axes[1].set_title("Average Tip Percentage by Passenger Count")
axes[1].set_xlabel("Passenger Count")
```

```
axes[1].set_ylabel("Tip Percentage (%)")

sns.lineplot(data=avg_tip_by_hour, x="hour", y="tip_percentage",
marker="o", color="red", ax=axes[2])
axes[2].set_title("Average Tip Percentage by Hour of Pickup")
axes[2].set_xlabel("Hour of the Day")
axes[2].set_ylabel("Tip Percentage (%)")
axes[2].set_xticks(range(0, 24))
plt.tight_layout()
plt.show()
```

```
/var/folders/fc/k_p1s4pj2f70fysh__74y2l00000gn/T/
ipykernel_61790/175312580.py:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=avg_tip_by_distance, x="distance_category",
y="tip_percentage", palette="Blues", ax=axes[0])
/var/folders/fc/k_p1s4pj2f70fysh__74y2l00000gn/T/ipykernel_61790/17531
2580.py:16: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=avg_tip_by_passenger, x="passenger_count",
y="tip_percentage", palette="Greens", ax=axes[1])
```



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

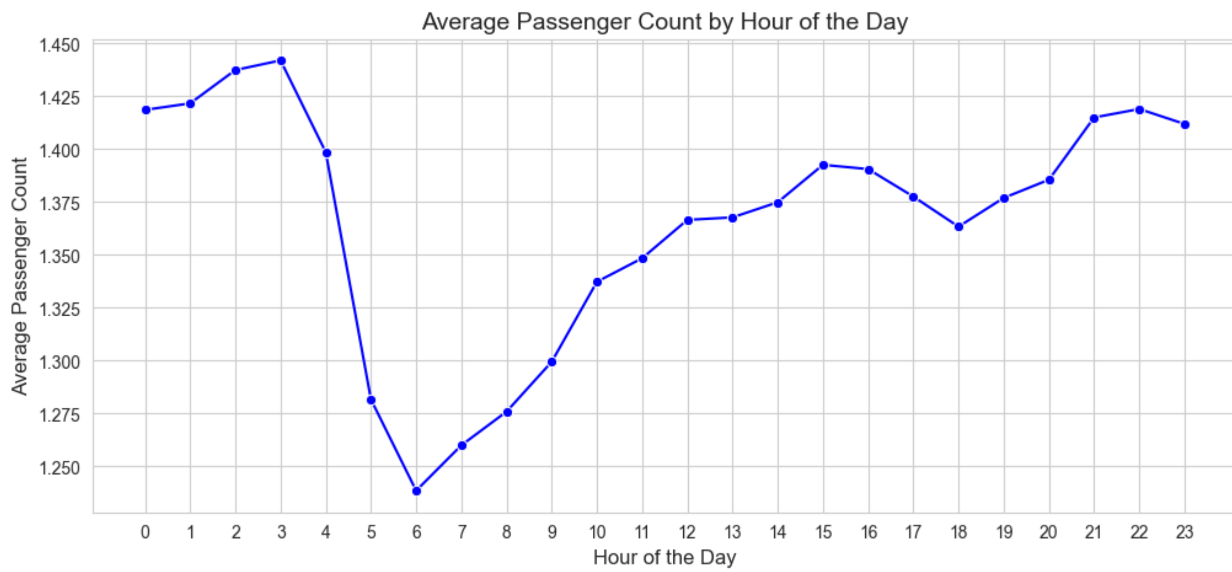
```
# Compare trips with tip percentage < 10% to trips with tip percentage
> 25%
low_tips = df_zones[df_zones['tip_percentage'] < 10]
high_tips = df_zones[df_zones['tip_percentage'] > 25]
avg_distance_low = low_tips['trip_distance'].mean()
avg_distance_high = high_tips['trip_distance'].mean()
```

```
print(f"Average Trip Distance - Low Tip (<10%): {avg_distance_low:.2f} miles")
print(f"Average Trip Distance - High Tip (>25%): {avg_distance_high:.2f} miles")

avg_fare_low = low_tips['fare_amount'].mean()
avg_fare_high = high_tips['fare_amount'].mean()

print(f"Average Fare Amount - Low Tip (<10%): ${avg_fare_low:.2f}")
print(f"Average Fare Amount - High Tip (>25%): ${avg_fare_high:.2f}")

avg_passengers_low = low_tips['passenger_count'].mean()
avg_passengers_high = high_tips['passenger_count'].mean()

print(f"Average Passenger Count - Low Tip (<10%): {round(avg_passengers_low)}")
print(f"Average Passenger Count - High Tip (>25%): {round(avg_passengers_high)}")
```

```
Average Trip Distance - Low Tip (<10%): 3.90 miles
Average Trip Distance - High Tip (>25%): 2.30 miles
Average Fare Amount - Low Tip (<10%): $21.46
Average Fare Amount - High Tip (>25%): $14.40
Average Passenger Count - Low Tip (<10%): 1
Average Passenger Count - High Tip (>25%): 1
```

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

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

avg_passengers_per_hour = df_zones.groupby("hour")["passenger_count"].mean().reset_index()
plt.figure(figsize=(12, 5))
sns.lineplot(data=avg_passengers_per_hour, x="hour", y="passenger_count", marker="o", color="blue")

plt.xlabel("Hour of the Day", fontsize=12)
plt.ylabel("Average Passenger Count", fontsize=12)
plt.title("Average Passenger Count by Hour of the Day", fontsize=14)
plt.xticks(range(0, 24))
plt.grid(True)
plt.show()
```



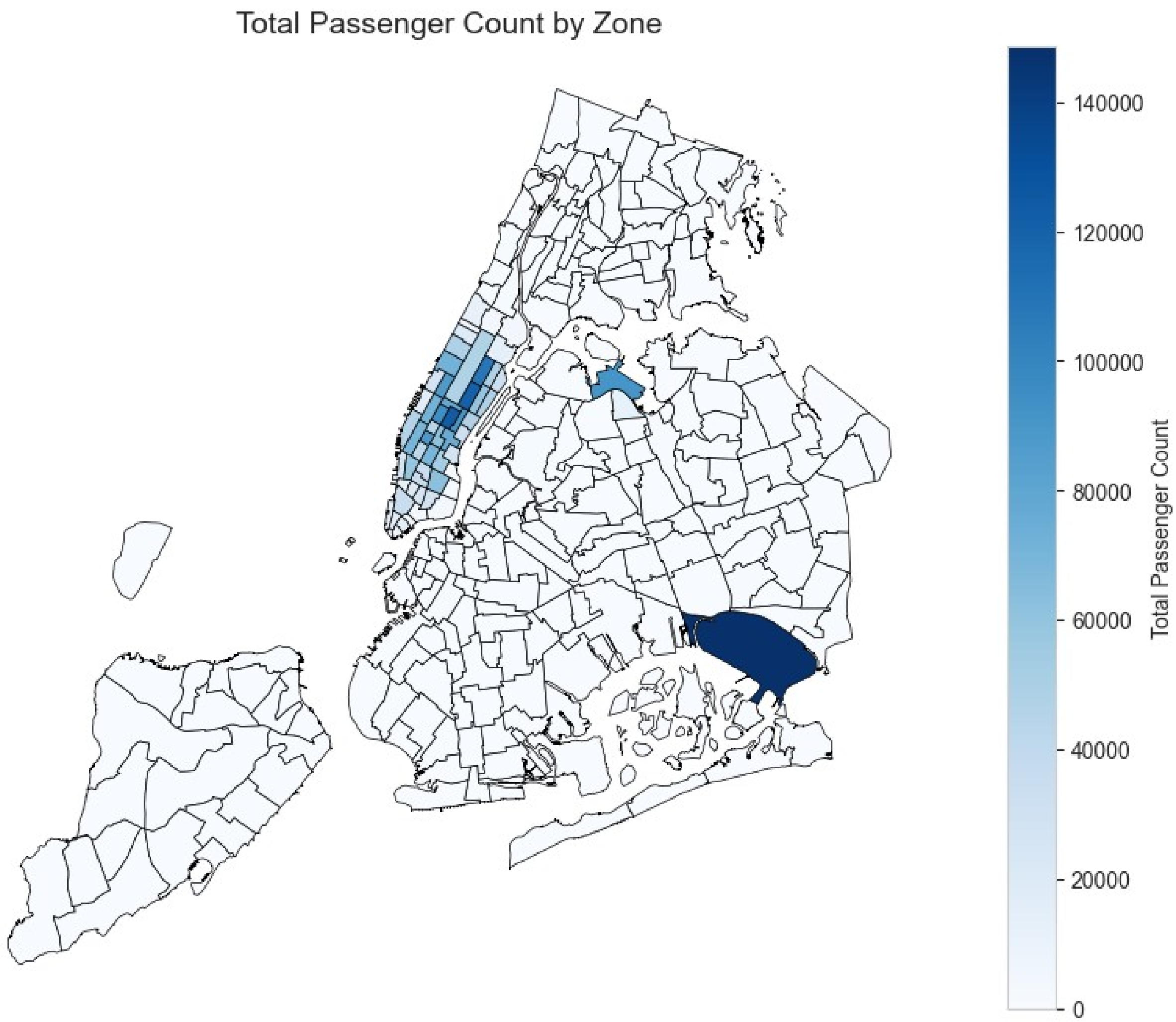
3.2.15 [2 marks] Analyse the variation of passenger counts across zones

```
# How does passenger count vary across zones
passengers_by_zone = df_zones.groupby("PULocationID")
["passenger_count"].sum().reset_index()
zones_passengers = zones.merge(passengers_by_zone,
left_on="LocationID", right_on="PULocationID", how="left")

zones_passengers["passenger_count"] =
zones_passengers["passenger_count"].fillna(0)
print(zones_passengers)
fig, ax = plt.subplots(1, 1, figsize=(12, 8))
zones_passengers.plot(column="passenger_count", cmap="Blues",
linewidth=0.5, edgecolor="black",
                      legend=True, legend_kwds={"label": "Total
Passenger Count"}, ax=ax)

ax.set_title("Total Passenger Count by Zone", fontsize=14)
ax.axis("off")
plt.show()
```

```
    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
```

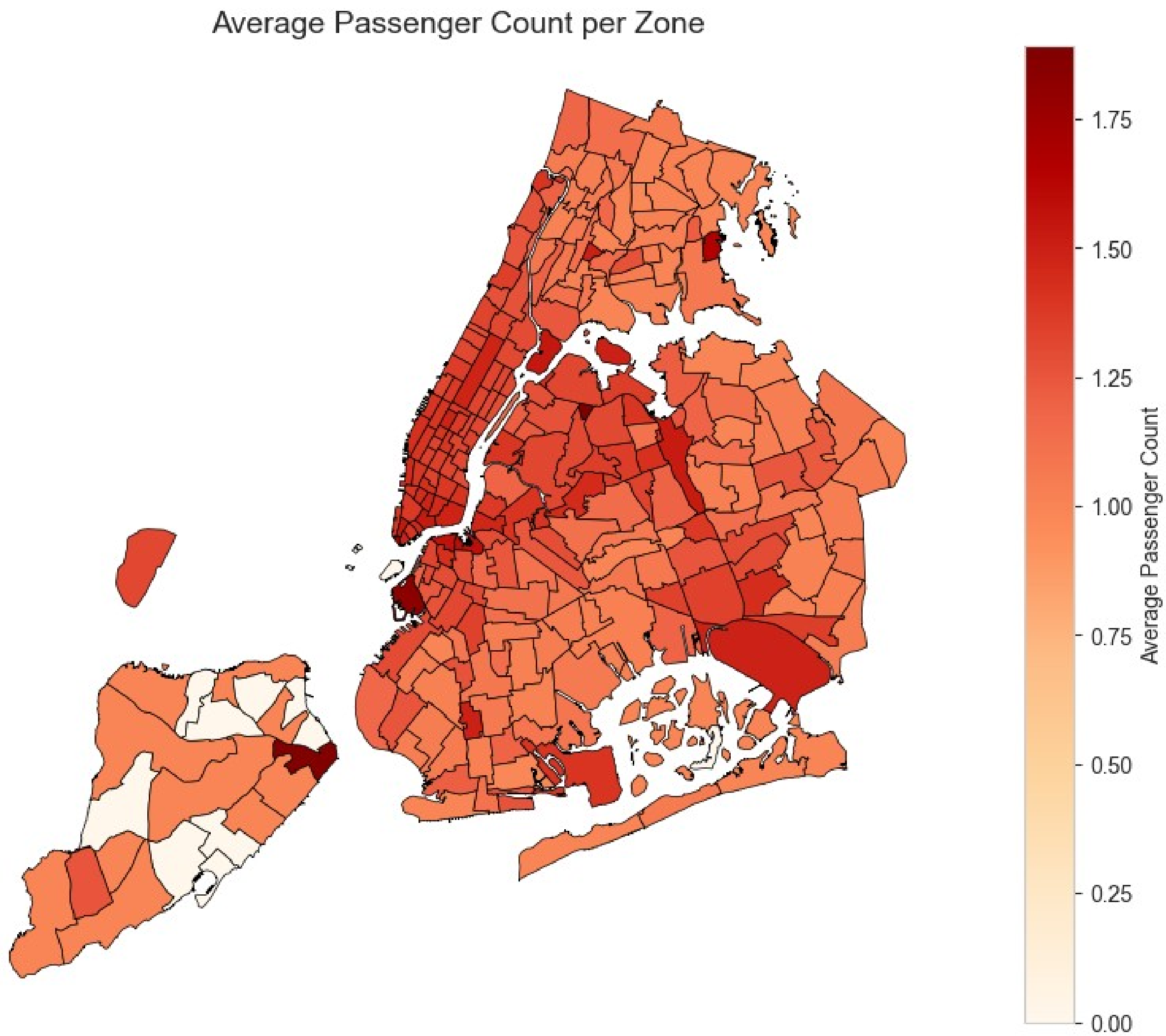
```
5
..       ...        ...        ...                    ...
...
258       259    0.126750    0.000395       Woodlawn/Wakefield
259
259       260    0.133514    0.000422                 Woodside
260
260       261    0.027120    0.000034       World Trade Center
261
261       262    0.049064    0.000122           Yorkville East
262
262       263    0.037017    0.000066           Yorkville West
263

          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...

..            ...                                               ...

258         Bronx  POLYGON ((1025414.782 270986.139, 1025138.624 ...

259        Queens  POLYGON ((1011466.966 216463.005, 1011545.889 ...

260     Manhattan  POLYGON ((980555.204 196138.486, 980570.792 19...

261     Manhattan  MULTIPOLYGON (((999804.795 224498.527, 999824....

262     Manhattan  POLYGON ((997493.323 220912.386, 997355.264 22...


     PULocationID  passenger_count
0             1.0             63.0
1             2.0              3.0
2             3.0             35.0
3             4.0           2670.0
4             5.0             10.0
..            ...              ...
258         259.0             40.0
259         260.0            416.0
260         261.0          15302.0
261         262.0          32439.0
```

```
262          263.0          48153.0

[263 rows x 9 columns]
```




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

avg_passengers_by_zone = df_zones.groupby("PULocationID")
["passenger_count"].mean().reset_index()
zones_with_trips = zones.merge(avg_passengers_by_zone,
left_on="LocationID", right_on="PULocationID", how="left")
zones_with_trips["avg_passenger_count"] =
zones_with_trips["passenger_count"].fillna(0)
zones_with_trips = zones_with_trips.drop(columns=["passenger_count"])
fig, ax = plt.subplots(1, 1, figsize=(12, 8))
zones_with_trips.plot(column="avg_passenger_count", cmap="OrRd",
linewidth=0.5, edgecolor="black",
```

```
                         legend=True, legend_kwds={"label": "Average
Passenger Count"}, ax=ax)

ax.set_title("Average Passenger Count per Zone", fontsize=14)
ax.axis("off")

plt.show()
```



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

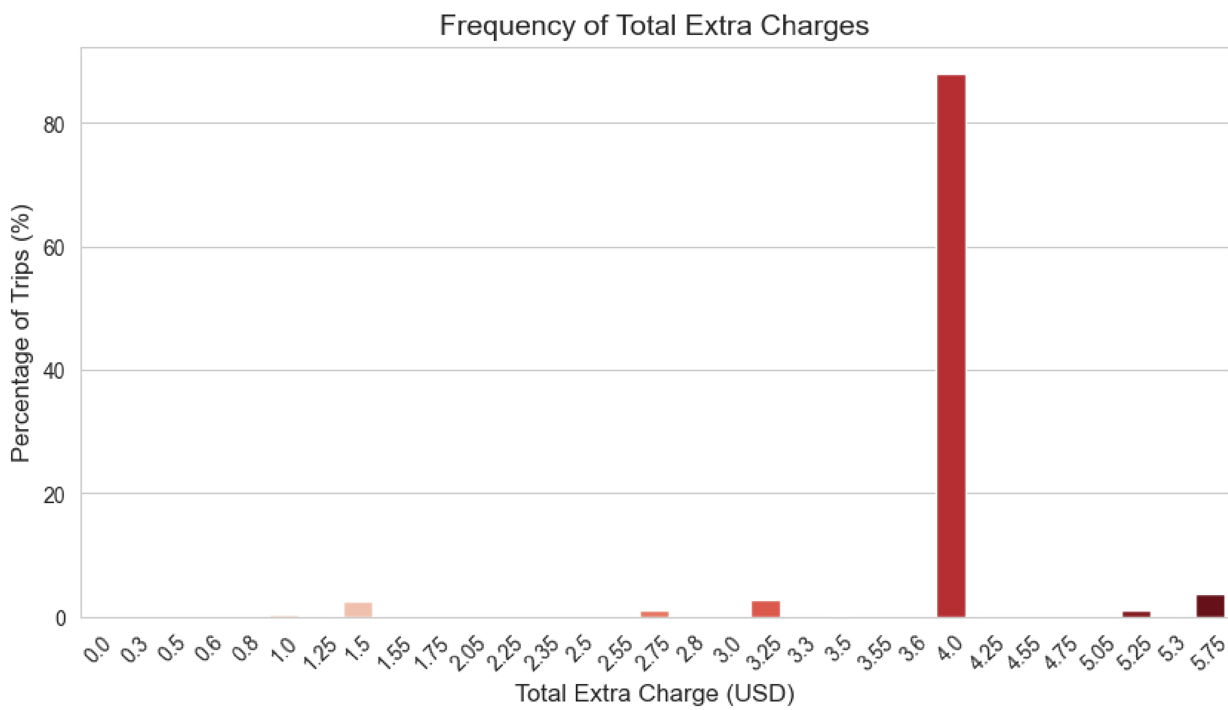
3.2.16 [5 marks] Analyse the pickup/dropoff zones or times when extra charges are applied more frequently

```
# How often is each surcharge applied?
df_zones["total_extra"] = df_zones["mta_tax"] +
df_zones["congestion_surcharge"] + df_zones["Airport__Fee"]
+df_zones["improvement_surcharge"]
```

```python
total_extra_counts =
df_zones["total_extra"].value_counts().reset_index()
total_extra_counts.columns = ["total_extra_charge", "count"]
total_extra_counts["total_extra_charge"]=total_extra_counts["total_ext
ra_charge"].round(2)
total_extra_counts["percentage"] = (total_extra_counts["count"] /
df_zones.shape[0]) * 100
print(total_extra_counts)

plt.figure(figsize=(10, 5))
sns.barplot(data=total_extra_counts, x="total_extra_charge",
y="percentage", palette="Reds")
plt.xlabel("Total Extra Charge (USD)", fontsize=12)
plt.ylabel("Percentage of Trips (%)", fontsize=12)
plt.title("Frequency of Total Extra Charges", fontsize=14)
plt.xticks(rotation=45)
plt.show()
```

```
    total_extra_charge    count  percentage
0                 4.00  1660020   88.001370
1                 5.75    71215    3.775266
2                 3.25    52762    2.797032
3                 1.50    49695    2.634443
4                 5.25    22332    1.183869
5                 2.75    19138    1.014548
6                 1.00     8354    0.442864
7                 3.50     1558    0.082593
8                 3.30      690    0.036578
9                 2.25      282    0.014949
10                0.00       89    0.004718
11                3.60       50    0.002651
12                4.75       45    0.002386
13                4.55       22    0.001166
14                2.05       19    0.001007
15                2.80       15    0.000795
16                0.80       15    0.000795
17                3.00        9    0.000477
18                5.05        8    0.000424
19                2.55        5    0.000265
20                1.75        3    0.000159
21                3.55        3    0.000159
22                0.30        3    0.000159
23                0.50        3    0.000159
24                4.25        2    0.000106
25                2.35        2    0.000106
26                2.50        2    0.000106
27                0.60        2    0.000106
28                1.25        1    0.000053
29                1.55        1    0.000053
30                5.30        1    0.000053
```

```
/var/folders/fc/k_p1s4pj2f70fysh__74y2l00000gn/T/
ipykernel_61790/1153697691.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(data=total_extra_counts, x="total_extra_charge",
y="percentage", palette="Reds")
```



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

Most drop-offs happen in East Village, Clinton East, Murray Hill East, Chelsea. o These areas have high rider availability, reducing cab downtime o Reduce cab dispatching to these areas.FK Airport, Upper East Side, Midtown Center, Midtown East are high demand points. o Cabs should be pre-positioned near these locations to reduce waiting times.

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

1# Solo passengers pay a higher fare per mile (no shared cost) 2# Trips with 1 or 5+ passengers receive higher trips 3# Peak demand: 3 PM - 7 PM (1 lakh+ requests) = opportunity for premium pricing 4# High tips during 4-6 PM → Business professionals, corporate rides, airports. 5# Implement Vendor 2’s Dynamic Pricing Model for Vendor 1 6# Optimize Route Selection to Reduce Trip Duration & Fuel Costs 7# High-traffic zones cause long trip durations, increasing operational costs 8# Choosing less congested routes reduces travel te & fuel consumption. 9#Prioritize drivers on optimized routes with minimal congestion.