New York City Yellow Taxi Data

Objective

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

Problem Statement

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

Tasks

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

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

↳ 3 cells hidden

Data Understanding

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

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

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

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

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

Data Description

You can find the data description here: [Data Dictionary]

Trip Records

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

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

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

Taxi Zones

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

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

This is covered in more detail in later sections.

---

1 Data Preparation

[5 marks]

Import Libraries

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

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

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

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

```
print("matplotlib version:", plt.matplotlib.__version__)
print("seaborn version:", sns.__version__)
```

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

1.1 Load the dataset

[5 marks]

You will see twelve files, one for each month.

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

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

```
# Try loading one file
df = pd.read_parquet('2023-1.parquet')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3041714 entries, 0 to 3066765
Data columns (total 19 columns):
 #   Column                 Dtype
---  ------                 -----
 0   VendorID               int64
 1   tpep_pickup_datetime   datetime64[us]
 2   tpep_dropoff_datetime  datetime64[us]
 3   passenger_count        float64
 4   trip_distance          float64
 5   RatecodeID             float64
 6   store_and_fwd_flag     object
 7   PULocationID           int64
 8   DOLocationID           int64
 9   payment_type           int64
 10  fare_amount            float64
 11  extra                  float64
 12  mta_tax                float64
 13  tip_amount             float64
 14  tolls_amount           float64
 15  improvement_surcharge  float64
 16  total_amount           float64
 17  congestion_surcharge   float64
 18  airport_fee            float64
dtypes: datetime64[us](2), float64(12), int64(4), object(1)
memory usage: 464.1+ MB
```

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

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

Sampling the Data

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

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

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

sample = hour_data.sample(frac = 0.05, random_state = 42)
# sample 0.05 of the hour_data
```

```
# random_state is just a seed for sampling, you can define it yourself

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

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

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

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

1.1.1 [5 marks]
Figure out how to sample and combine the files.

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

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

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

```
Mounted at /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 h
# Create a single dataframe for the year combining all the monthly data

# Select the folder having data files
import os

# Select the folder having data files
os.chdir('/content/drive/My Drive/EDA/trip_records')

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

# initialise an empty dataframe
df_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
        current_path = os.path.join(os.getcwd(), file_name)
        print(f"Processing: {file_name}")

        # Reading the current file
        taxi_data = pd.read_parquet(current_path)

        # Ensure datetime conversion
        taxi_data["tpep_pickup_datetime"] = pd.to_datetime(
            taxi_data["tpep_pickup_datetime"]
        )
        taxi_data["pickup_date"] = taxi_data["tpep_pickup_datetime"].dt.date
        taxi_data["pickup_hour"] = taxi_data["tpep_pickup_datetime"].dt.hour

        # We will store the sampled data for the current date in this df by appending the sampled data from e
        # After completing iteration through each date, we will append this data to the final dataframe.
        sampled_data = (
            taxi_data
            .groupby(["pickup_date", "pickup_hour"], group_keys=False)
            .sample(frac=0.05, random_state=42)
```

```
            )

            # Concatenate the sampled data of all the dates to a single dataframe
            df_list.append(sampled_data)

            print(f"Finished processing file: {file_name} | Sampled rows: {len(sampled_data)}")

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

    # Create a single dataframe for the year combining all the monthly data
    df = pd.concat(df_list, ignore_index=True)
```

```
Processing: 2023-3.parquet
Finished processing file: 2023-3.parquet | Sampled rows: 163786
Processing: 2023-4.parquet
Finished processing file: 2023-4.parquet | Sampled rows: 139641
Processing: 2023-5.parquet
Finished processing file: 2023-5.parquet | Sampled rows: 144458
Processing: 2023-6.parquet
Finished processing file: 2023-6.parquet | Sampled rows: 162910
Processing: 2023-8.parquet
Finished processing file: 2023-8.parquet | Sampled rows: 143782
Processing: 2023-7.parquet
Finished processing file: 2023-7.parquet | Sampled rows: 174068
Processing: 2023-9.parquet
Finished processing file: 2023-9.parquet | Sampled rows: 140875
Processing: 2023-2.parquet
Finished processing file: 2023-2.parquet | Sampled rows: 168696
Processing: 2023-1.parquet
Finished processing file: 2023-1.parquet | Sampled rows: 152087
Processing: 2023-10.parquet
Finished processing file: 2023-10.parquet | Sampled rows: 174255
Processing: 2023-12.parquet
Finished processing file: 2023-12.parquet | Sampled rows: 166709
Processing: 2023-11.parquet
Finished processing file: 2023-11.parquet | Sampled rows: 165133
```

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

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

```
# Store the df in csv/parquet
# df.to_parquet('')
max_entries = 300000
print("First sampled data shape =",df.shape)

if len(df) > max_entries:
    df = df.sample(n=max_entries, random_state=42)
df.to_csv('sampled_nyc_taxi_data.csv', index=False)

print("Final shape =", df.shape)
```

```
First sampled data shape = (1896400, 22)
Final shape = (300000, 22)
```

2 Data Cleaning

[30 marks]

Now we can load the new data directly.

```
# Load the new data file
df = pd.read_csv("/content/drive/My Drive/EDA/trip_records/sampled_nyc_taxi_data.csv")
```

```
df.head()
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_anc
0	2	2023-10-02 18:54:59	2023-10-02 19:04:20	1.0	2.10	1.0	
1	2	2023-06-08 12:46:14	2023-06-08 12:48:58	1.0	0.36	1.0	
2	2	2023-03-15 17:18:43	2023-03-15 17:44:54	1.0	4.38	1.0	
3	1	2023-05-10 07:19:08	2023-05-10 07:30:21	1.0	4.20	1.0	
4	1	2023-11-08 15:48:24	2023-11-08 16:01:36	1.0	0.60	1.0	

5 rows × 22 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 300000 entries, 0 to 299999
Data columns (total 22 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   VendorID               300000 non-null  int64  
 1   tpep_pickup_datetime   300000 non-null  object 
 2   tpep_dropoff_datetime  300000 non-null  object 
 3   passenger_count        289673 non-null  float64
 4   trip_distance          300000 non-null  float64
 5   RatecodeID             289673 non-null  float64
 6   store_and_fwd_flag     289673 non-null  object 
 7   PULocationID           300000 non-null  int64  
 8   DOLocationID           300000 non-null  int64  
 9   payment_type           300000 non-null  int64  
 10  fare_amount            300000 non-null  float64
 11  extra                  300000 non-null  float64
 12  mta_tax                300000 non-null  float64
 13  tip_amount             300000 non-null  float64
 14  tolls_amount           300000 non-null  float64
 15  improvement_surcharge  300000 non-null  float64
 16  total_amount           300000 non-null  float64
 17  congestion_surcharge   289673 non-null  float64
 18  Airport_fee            265995 non-null  float64
 19  pickup_date            300000 non-null  object 
 20  pickup_hour            300000 non-null  int64  
 21  airport_fee            23678 non-null   float64
dtypes: float64(13), int64(5), object(4)
memory usage: 50.4+ 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 = df.reset_index(drop=True)
print(df.head())
```

```
   VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
0         2  2023-10-02 18:54:59   2023-10-02 19:04:20              1.0   
1         2  2023-06-08 12:46:14   2023-06-08 12:48:58              1.0   
2         2  2023-03-15 17:18:43   2023-03-15 17:44:54              1.0   
3         1  2023-05-10 07:19:08   2023-05-10 07:30:21              1.0   
4         1  2023-11-08 15:48:24   2023-11-08 16:01:36              1.0   

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID  DOLocationID  \
0           2.10         1.0                  N            90           163   
1           0.36         1.0                  N           143           143   
2           4.38         1.0                  N           140           158   
3           4.20         1.0                  N           107            88   
4           0.60         1.0                  N           140           237   

   payment_type  ...  mta_tax  tip_amount  tolls_amount  \
0             1  ...      0.5        3.72           0.0   
1             2  ...      0.5        0.00           0.0   
```

```
2             1  ...      0.5          6.38            0.0
3             1  ...      0.5          3.00            0.0
4             1  ...      0.5          3.85            0.0

   improvement_surcharge  total_amount  congestion_surcharge  Airport_fee  \
0                    1.0         22.32                   2.5          0.0
1                    1.0          9.10                   2.5          0.0
2                    1.0         38.28                   2.5          0.0
3                    1.0         26.10                   2.5          0.0
4                    1.0         19.25                   2.5          0.0

   pickup_date  pickup_hour airport_fee
0   2023-10-02           18         NaN
1   2023-06-08           12         NaN
2   2023-03-15           17         NaN
3   2023-05-10            7         NaN
4   2023-11-08           15         NaN

[5 rows x 22 columns]
```

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'].fillna(df['Airport_fee'])

# Drop the column 'Airport_fee'
df = df.drop(columns=['Airport_fee'])
print(df.head())
```

```
   VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
0         2  2023-10-02 18:54:59   2023-10-02 19:04:20              1.0
1         2  2023-06-08 12:46:14   2023-06-08 12:48:58              1.0
2         2  2023-03-15 17:18:43   2023-03-15 17:44:54              1.0
3         1  2023-05-10 07:19:08   2023-05-10 07:30:21              1.0
4         1  2023-11-08 15:48:24   2023-11-08 16:01:36              1.0

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID  DOLocationID  \
0           2.10         1.0                  N            90           163
1           0.36         1.0                  N           143           143
2           4.38         1.0                  N           140           158
3           4.20         1.0                  N           107            88
4           0.60         1.0                  N           140           237

   payment_type  ...  extra  mta_tax  tip_amount  tolls_amount  \
0             1  ...    2.5      0.5        3.72           0.0
1             2  ...    0.0      0.5        0.00           0.0
2             1  ...    2.5      0.5        6.38           0.0
3             1  ...    2.5      0.5        3.00           0.0
4             1  ...    2.5      0.5        3.85           0.0

   improvement_surcharge  total_amount  congestion_surcharge  pickup_date  \
0                    1.0         22.32                   2.5   2023-10-02
1                    1.0          9.10                   2.5   2023-06-08
2                    1.0         38.28                   2.5   2023-03-15
3                    1.0         26.10                   2.5   2023-05-10
4                    1.0         19.25                   2.5   2023-11-08

  pickup_hour  airport_fee
0          18          0.0
1          12          0.0
2          17          0.0
3           7          0.0
4          15          0.0

[5 rows x 21 columns]
```

2.1.3 [5 marks]

Fix columns with negative (monetary) values

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

negFareTrip = df[df['fare_amount'] < 0]
print(negFareTrip)
```

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

[0 rows x 21 columns]
```

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

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

```
RatecodeID
1.0     273411
2.0      11337
5.0       1714
99.0      1671
3.0        947
4.0        593
Name: count, dtype: int64
```

```
# Find which columns have negative values
numeric_columns = df.select_dtypes(include=['number']).columns
columns_with_negatives = df[numeric_columns].lt(0).any()
negative_columns = columns_with_negatives[columns_with_negatives].index

print("List of columns containing negative values:", list(negative_columns))
```

```
List of columns containing negative values: ['extra', 'mta_tax', 'improvement_surcharge', 'total_amount', 'cong
```

```
# fix these negative values

# Count of negative values
print("Number of negative values before fixing :")
print((df[negative_columns] < 0).sum())
```

```
Number of negative values before fixing :
extra                     1
mta_tax                  13
improvement_surcharge    14
total_amount             14
congestion_surcharge      9
airport_fee               3
dtype: int64
```

```
# Count of Negative values are less so remove it

cleaned_df = df[~(df[negative_columns] < 0).any(axis=1)]
print("Preview of dataset after removing rows with negative values:")
print(cleaned_df.head())
```

```
Preview of dataset after removing rows with negative values:
   VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
0         2  2023-10-02 18:54:59   2023-10-02 19:04:20              1.0
1         2  2023-06-08 12:46:14   2023-06-08 12:48:58              1.0
2         2  2023-03-15 17:18:43   2023-03-15 17:44:54              1.0
3         1  2023-05-10 07:19:08   2023-05-10 07:30:21              1.0
4         1  2023-11-08 15:48:24   2023-11-08 16:01:36              1.0

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID  DOLocationID  \
0           2.10         1.0                  N            90           163
1           0.36         1.0                  N           143           143
2           4.38         1.0                  N           140           158
3           4.20         1.0                  N           107            88
4           0.60         1.0                  N           140           237

   payment_type  ...  extra  mta_tax  tip_amount  tolls_amount  \
0             1  ...    2.5      0.5        3.72           0.0
1             2  ...    0.0      0.5        0.00           0.0
2             1  ...    2.5      0.5        6.38           0.0
3             1  ...    2.5      0.5        3.00           0.0
4             1  ...    2.5      0.5        3.85           0.0
```

```
   improvement_surcharge  total_amount  congestion_surcharge  pickup_date  \
0                    1.0         22.32                   2.5   2023-10-02
1                    1.0          9.10                   2.5   2023-06-08
2                    1.0         38.28                   2.5   2023-03-15
3                    1.0         26.10                   2.5   2023-05-10
4                    1.0         19.25                   2.5   2023-11-08

  pickup_hour  airport_fee
0          18          0.0
1          12          0.0
2          17          0.0
3           7          0.0
4          15          0.0

[5 rows x 21 columns]
```

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
missing_value_ratio = cleaned_df.isna().mean() * 100
print("Proportion of missing values per column (sorted in descending order):")
print(missing_value_ratio.sort_values(ascending=False))
```

```
Proportion of missing values per column (sorted in descending order):
passenger_count          3.442494
airport_fee              3.442494
congestion_surcharge     3.442494
store_and_fwd_flag       3.442494
RatecodeID               3.442494
trip_distance            0.000000
tpep_dropoff_datetime    0.000000
tpep_pickup_datetime     0.000000
VendorID                 0.000000
payment_type             0.000000
fare_amount              0.000000
PULocationID             0.000000
DOLocationID             0.000000
mta_tax                  0.000000
extra                    0.000000
tip_amount               0.000000
tolls_amount             0.000000
total_amount             0.000000
improvement_surcharge    0.000000
pickup_date              0.000000
pickup_hour              0.000000
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'

rows_with_nulls = cleaned_df[cleaned_df['passenger_count'].isna()]
print("Rows where 'passenger_count' is missing:")
print(rows_with_nulls)
print("Number of missing values in 'passenger_count':",
      cleaned_df['passenger_count'].isna().sum())

passenger_count_mode = cleaned_df['passenger_count'].mode()[0]
cleaned_df['passenger_count'] = cleaned_df['passenger_count'].fillna(passenger_count_mode)

print("Number of missing values in 'passenger_count' after imputation:",
      cleaned_df['passenger_count'].isna().sum())
```

```
299928        1  2023-05-31 13:18:09   2023-05-31 13:45:30             NaN
299931        1  2023-12-12 13:03:50   2023-12-12 13:09:47             NaN

        trip_distance  RatecodeID store_and_fwd_flag  PULocationID  \
25               2.38         NaN                NaN           249
42               0.00         NaN                NaN           161
88               5.38         NaN                NaN           100
92               3.41         NaN                NaN           246
129              5.76         NaN                NaN           223
...               ...         ...                ...           ...
299741           3.15         NaN                NaN            79
299878           3.29         NaN                NaN           141
299887          16.13         NaN                NaN           107
299928           3.10         NaN                NaN           162
299931           0.80         NaN                NaN            50

        DOLocationID  payment_type  ...  extra  mta_tax  tip_amount  \
25                87             0  ...    0.0      0.5        0.00
42               233             0  ...    0.0      0.5        0.00
88                87             0  ...    0.0      0.5        5.92
92               148             0  ...    0.0      0.5        0.00
129               75             0  ...    0.0      0.5        5.03
...              ...           ...  ...    ...      ...         ...
299741           246             0  ...    0.0      0.5        0.00
299878            90             0  ...    0.0      0.5        3.01
299887             1             0  ...    0.0      0.0       14.71
299928           125             0  ...    0.0      0.5        2.87
299931            48             0  ...    0.0      0.5        0.00

        tolls_amount  improvement_surcharge  total_amount  \
25              0.00                    1.0         17.00
42              0.00                    1.0         10.83
88              0.00                    1.0         25.51
92              0.00                    1.0         39.55
129             6.55                    1.0         38.58
...              ...                    ...           ...
299741          0.00                    1.0         20.23
299878          0.00                    1.0         33.11
299887         12.75                    1.0         88.27
299928          0.00                    1.0         31.57
299931          0.00                    1.0         11.20

        congestion_surcharge  pickup_date pickup_hour  airport_fee
25                       NaN   2023-06-03           2          NaN
42                       NaN   2023-02-25          23          NaN
88                       NaN   2023-11-22          14          NaN
92                       NaN   2023-12-15          18          NaN
129                      NaN   2023-03-02           9          NaN
...                      ...          ...         ...          ...
299741                   NaN   2023-04-02           4          NaN
299878                   NaN   2023-01-12          18          NaN
299887                   NaN   2023-07-04           3          NaN
299928                   NaN   2023-05-31          13          NaN
299931                   NaN   2023-12-12          13          NaN

[10327 rows x 21 columns]
Number of missing values in 'passenger_count': 10327
Number of missing values in 'passenger_count' after imputation: 0
```

Did you find zeroes in passenger_count? Handle these.

```
zero_count_before = (cleaned_df['passenger_count'] == 0).sum()
print("Initial count of zero values in 'passenger_count':", zero_count_before)

# Replace zero values in 'passenger_count' with the most frequent value (mode)
passenger_count_mode = cleaned_df['passenger_count'].mode()[0]
cleaned_df.loc[:, 'passenger_count'] = cleaned_df['passenger_count'].replace(0, passenger_count_mode)

# Recheck the number of zero values in the 'passenger_count' column after replacement
zero_count_after = (cleaned_df['passenger_count'] == 0).sum()
print("Count of zero values in 'passenger_count' after replacement:", zero_count_after)
```

```
Initial count of zero values in 'passenger_count': 4654
Count of zero values in 'passenger_count' after replacement: 0
```

2.2.3 [2 marks]

Handle missing values in `RatecodeID`

```
# Fix missing values in 'RatecodeID'
rows_with_missing_ratecode = cleaned_df[cleaned_df['RatecodeID'].isna()]
print("Rows where 'RatecodeID' is missing:")
print(rows_with_missing_ratecode)

# Replace missing values with the most frequent value (mode)
ratecode_mode = cleaned_df['RatecodeID'].mode()[0]
print("Most frequent value (mode) of 'RatecodeID':", ratecode_mode)

cleaned_df['RatecodeID'].fillna(ratecode_mode, inplace=True)

print("Number of missing values in 'RatecodeID' after imputation:",
      cleaned_df['RatecodeID'].isna().sum())
```

```
299928        1  2023-05-31 13:18:09   2023-05-31 13:45:30              1.0
299931        1  2023-12-12 13:03:50   2023-12-12 13:09:47              1.0

        trip_distance  RatecodeID store_and_fwd_flag  PULocationID  \
25               2.38         NaN                NaN           249
42               0.00         NaN                NaN           161
88               5.38         NaN                NaN           100
92               3.41         NaN                NaN           246
129              5.76         NaN                NaN           223
...               ...         ...                ...           ...
299741           3.15         NaN                NaN            79
299878           3.29         NaN                NaN           141
299887          16.13         NaN                NaN           107
299928           3.10         NaN                NaN           162
299931           0.80         NaN                NaN            50

        DOLocationID  payment_type  ...  extra  mta_tax  tip_amount  \
25                87             0  ...    0.0      0.5        0.00
42               233             0  ...    0.0      0.5        0.00
88                87             0  ...    0.0      0.5        5.92
92               148             0  ...    0.0      0.5        0.00
129               75             0  ...    0.0      0.5        5.03
...              ...           ...  ...    ...      ...         ...
299741           246             0  ...    0.0      0.5        0.00
299878            90             0  ...    0.0      0.5        3.01
299887             1             0  ...    0.0      0.0       14.71
299928           125             0  ...    0.0      0.5        2.87
299931            48             0  ...    0.0      0.5        0.00

        tolls_amount  improvement_surcharge  total_amount  \
25              0.00                    1.0         17.00
42              0.00                    1.0         10.83
88              0.00                    1.0         25.51
92              0.00                    1.0         39.55
129             6.55                    1.0         38.58
...              ...                    ...           ...
299741          0.00                    1.0         20.23
299878          0.00                    1.0         33.11
299887         12.75                    1.0         88.27
299928          0.00                    1.0         31.57
299931          0.00                    1.0         11.20

        congestion_surcharge  pickup_date pickup_hour  airport_fee
25                       NaN   2023-06-03           2          NaN
42                       NaN   2023-02-25          23          NaN
88                       NaN   2023-11-22          14          NaN
92                       NaN   2023-12-15          18          NaN
129                      NaN   2023-03-02           9          NaN
...                      ...          ...         ...          ...
299741                   NaN   2023-04-02           4          NaN
299878                   NaN   2023-01-12          18          NaN
299887                   NaN   2023-07-04           3          NaN
299928                   NaN   2023-05-31          13          NaN
299931                   NaN   2023-12-12          13          NaN

[10327 rows x 21 columns]
Most frequent value (mode) of 'RatecodeID': 1.0
Number of missing values in 'RatecodeID' after imputation: 0
```

2.2.4 [3 marks]

Impute NaN in `congestion_surcharge`

```
# handle null values in congestion_surcharge
```

```
totalNullValuesCongestionSurcharge = cleaned_df['congestion_surcharge'].isna().sum()
print(f"Number of null values in congestion_surcharge : {totalNullValuesCongestionSurcharge}")

congestion_surcharge_mode = cleaned_df['congestion_surcharge'].mode()[0]
print("Most frequent value (mode) of 'congestion_surcharge':", congestion_surcharge_mode)

# Replace null values with the most frequent value (mode)
cleaned_df.loc[:, 'congestion_surcharge'] = cleaned_df['congestion_surcharge'].fillna(congestion_surcharge_mode

print("Remaining missing values in 'congestion_surcharge' after imputation:",
      cleaned_df['congestion_surcharge'].isna().sum())
```

```
Number of null values in congestion_surcharge : 10327
Most frequent value (mode) of 'congestion_surcharge': 2.5
Remaining missing values in 'congestion_surcharge' after imputation: 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

missing_value_ratio = cleaned_df.isna().mean() * 100
print(f"Missing values proportion before fixing :\n {missing_value_ratio}")

for col in cleaned_df.columns:
    missing_before = cleaned_df[col].isna().sum()
    dataType = cleaned_df[col].dtype

    if missing_before > 0:
      cleaned_df[col] = cleaned_df[col].fillna(cleaned_df[col].mode()[0])

    missing_after = cleaned_df[col].isna().sum()

print(f"\nMissing values proportion after fixing :\n {cleaned_df.isna().sum()}")
```

```
Missing values proportion before fixing :
 VendorID                 0.000000
tpep_pickup_datetime     0.000000
tpep_dropoff_datetime    0.000000
passenger_count          0.000000
trip_distance            0.000000
RatecodeID               0.000000
store_and_fwd_flag       3.442494
PULocationID             0.000000
DOLocationID             0.000000
payment_type             0.000000
fare_amount              0.000000
extra                    0.000000
mta_tax                  0.000000
tip_amount               0.000000
tolls_amount             0.000000
improvement_surcharge    0.000000
total_amount             0.000000
congestion_surcharge     0.000000
pickup_date              0.000000
pickup_hour              0.000000
airport_fee              3.442494
dtype: float64

Missing values proportion after fixing :
 VendorID                 0
tpep_pickup_datetime     0
tpep_dropoff_datetime    0
passenger_count          0
trip_distance            0
RatecodeID               0
store_and_fwd_flag       0
PULocationID             0
DOLocationID             0
payment_type             0
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
tolls_amount             0
improvement_surcharge    0
```

```
total_amount             0
congestion_surcharge     0
pickup_date              0
pickup_hour              0
airport_fee              0
dtype: int64
```

2.3 Handling Outliers

[10 marks]

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

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

cleaned_df.describe()
```

	VendorID	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type
count	299986.000000	299986.000000	299986.000000	299986.000000	299986.000000	299986.000000	299986.000000
mean	1.736808	1.373164	4.443431	1.618749	165.405979	164.192599	1.163561
std	0.445410	0.867598	203.103563	7.298881	63.988885	69.861815	0.507157
min	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	0.000000
25%	1.000000	1.000000	1.040000	1.000000	132.000000	114.000000	1.000000
50%	2.000000	1.000000	1.790000	1.000000	162.000000	162.000000	1.000000
75%	2.000000	1.000000	3.400000	1.000000	234.000000	234.000000	1.000000
max	6.000000	9.000000	76886.520000	99.000000	265.000000	265.000000	4.000000

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.

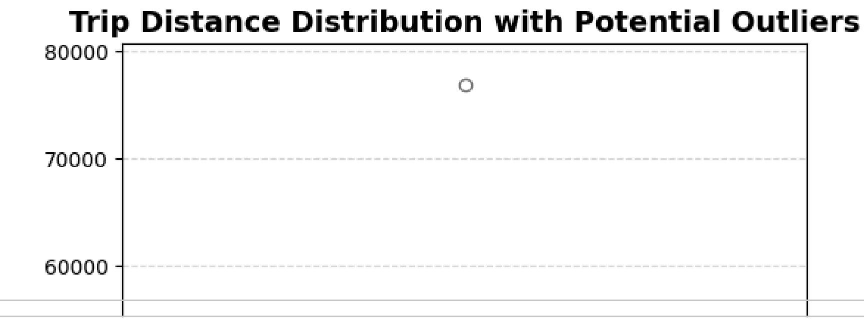
How will you fix each of these values? Which ones will you drop and which ones will you replace?

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

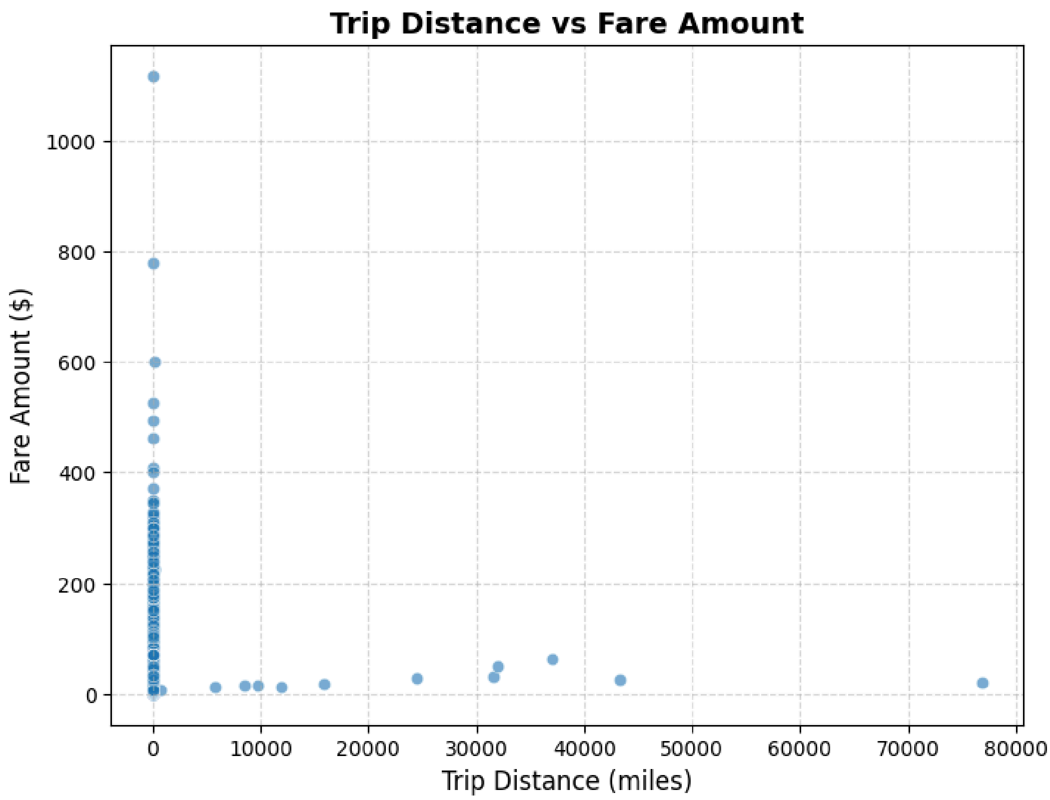
```
# remove passenger_count > 6
# Passenger count distribution before cleaning
plt.figure(figsize=(6,8))
sns.boxplot(y=cleaned_df['passenger_count'], color='lightcoral')
plt.title('Passenger Count Distribution Before Cleaning', fontsize=14, fontweight='bold')
plt.ylabel('Passenger Count', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()

print("Passenger Count before cleaning:\n")
print(cleaned_df['passenger_count'].describe())

# Only consider passenger_count <=6
cleaned_df = cleaned_df[cleaned_df['passenger_count'] <= 6]
```

```
# Passenger count distribution after cleaning
plt.figure(figsize=(6,8))
sns.boxplot(y=cleaned_df['passenger_count'], color='skyblue')
plt.title('Passenger Count Distribution After Cleaning', fontsize=14, fontweight='bold')
plt.ylabel('Passenger Count', fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()

print("Passenger Count post cleaning:\n")
print(cleaned_df['passenger_count'].describe())
```

```
# Check for potential outliers in 'trip_distance' using a boxplot

plt.figure(figsize=(6,8))
sns.boxplot(y=cleaned_df["trip_distance"], color='lightgreen')
plt.title("Trip Distance Distribution with Potential Outliers", fontsize=14, fontweight='bold')
plt.ylabel("Trip Distance (miles)", fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.show()

print("Trip distance statistics:\n")
print(cleaned_df['trip_distance'].describe())
```

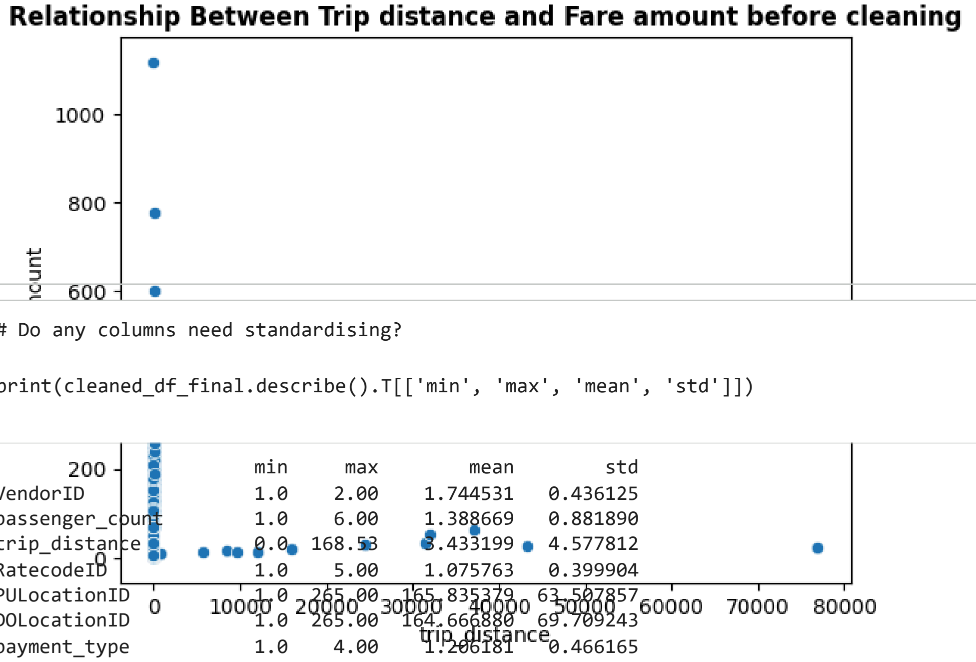



```
# Scatter plot to examine the relationship between 'trip_distance' and 'fare_amount'

plt.figure(figsize=(8,6))
sns.scatterplot(x=cleaned_df["trip_distance"], y=cleaned_df["fare_amount"], alpha=0.6)
plt.title("Trip Distance vs Fare Amount", fontsize=14, fontweight='bold')
plt.xlabel("Trip Distance (miles)", fontsize=12)
plt.ylabel("Fare Amount ($)", fontsize=12)
plt.grid(True, linestyle='--', alpha=0.5)
plt.show()
```



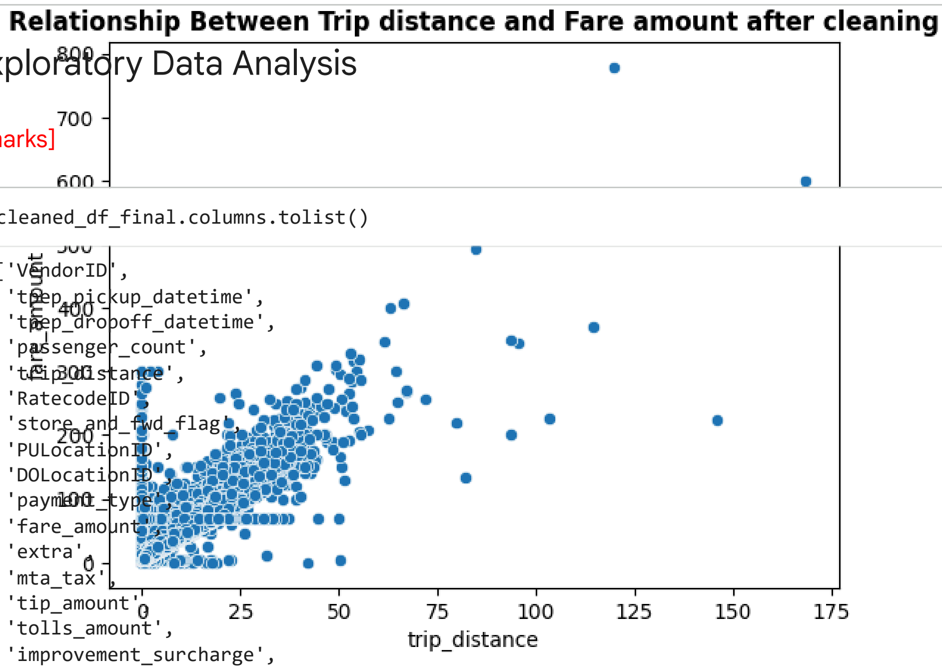

```
# Continue with outlier handling

# Relationship Between Trip distance and Fare amount before cleaning
sns.scatterplot(x=cleaned_df["trip_distance"], y= df["fare_amount"])
plt.title('Relationship Between Trip distance and Fare amount before cleaning', fontsize=12, fontweight='bold')
plt.show()

# Step 1: Remove entries where trip_distance is less than 5 and fare_amount is unusually high (>300)
outliers_trip_fare = cleaned_df[(cleaned_df['trip_distance'] < 5) & (cleaned_df['fare_amount'] > 300)]
print("Step 1: Count of records identified as outliers:", outliers_trip_fare.shape)
cleaned_step1 = cleaned_df[~((cleaned_df['trip_distance'] < 5) & (cleaned_df['fare_amount'] > 300))]
print("Step 1: Count of records after outlier removal:", cleaned_step1.shape)

# Step 2: Remove entries where trip_distance and fare_amount are 0 but pickup and dropoff zones differ
outliers_zero_trip = cleaned_step1[
    (cleaned_step1['trip_distance'] == 0) &
    (cleaned_step1['fare_amount'] == 0) &
    (cleaned_step1['PULocationID'] != cleaned_step1['DOLocationID'])
]
print("Step 2: Count of records identified as outliers:", outliers_zero_trip.shape)
cleaned_step2 = cleaned_step1[~(
    (cleaned_step1['trip_distance'] == 0) &
```

```
    (cleaned_step1['fare_amount'] == 0) &
    (cleaned_step1['PULocationID'] != cleaned_step1['DOLocationID'])
)]
print("Step 2: Count of records after outlier removal:", cleaned_step2.shape)

# Step 3: Remove entries where trip_distance > 250 (extremely rare long trips)
outliers_long_trips = cleaned_step2[cleaned_step2['trip_distance'] > 250]
print("Step 3: Count of records identified as outliers:", outliers_long_trips.shape)
cleaned_step3 = cleaned_step2[~(cleaned_step2['trip_distance'] > 250)]
print("Step 3: Count of records after outlier removal:", cleaned_step3.shape)

# Step 4: Remove entries with invalid payment_type (e.g., 0)
outliers_invalid_payment = cleaned_step3[cleaned_step3['payment_type'] == 0]
print("Step 4: Count of records identified as outliers:", outliers_invalid_payment.shape)
cleaned_step4 = cleaned_step3[cleaned_step3['payment_type'] != 0]
print("Step 4: Count of records after outlier removal:", cleaned_step4.shape)

# Step 5: Remove entries with invalid RatecodeID (valid: 1-6)
invalid_ratecode = cleaned_step4[~cleaned_step4['RatecodeID'].isin([1,2,3,4,5,6])]
print("Step 5: Count of records with invalid RatecodeID:", invalid_ratecode.shape)
cleaned_df_final = cleaned_step4[cleaned_step4['RatecodeID'].isin([1,2,3,4,5,6])]
print("Step 5: Count of records after removing invalid RatecodeID:", cleaned_df_final.shape)

# Relationship Between Trip distance and Fare amount after cleaning
sns.scatterplot(x=cleaned_df_final["trip_distance"], y= cleaned_df_final["fare_amount"])
plt.title('Relationship Between Trip distance and Fare amount after cleaning', fontsize=12, fontweight='bold')
plt.show()
```




```
# Do any columns need standardising?

print(cleaned_df_final.describe().T[['min', 'max', 'mean', 'std']])
```

```
                  min     max        mean        std
VendorID          1.0    2.00    1.744531   0.436125
passenger_count   1.0    6.00    1.388669   0.881890
trip_distance     0.0  168.53    3.433199   4.577812
RatecodeID        1.0    5.00    1.075763   0.399904
PULocationID      1.0  265.00  165.835379  63.507857
DOLocationID      1.0  265.00  164.666880  69.709243
payment_type      1.0    4.00    1.206181   0.466165
[illegible]
```



3 Exploratory Data Analysis

[90 marks]

```
cleaned_df_final.columns.tolist()
```

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

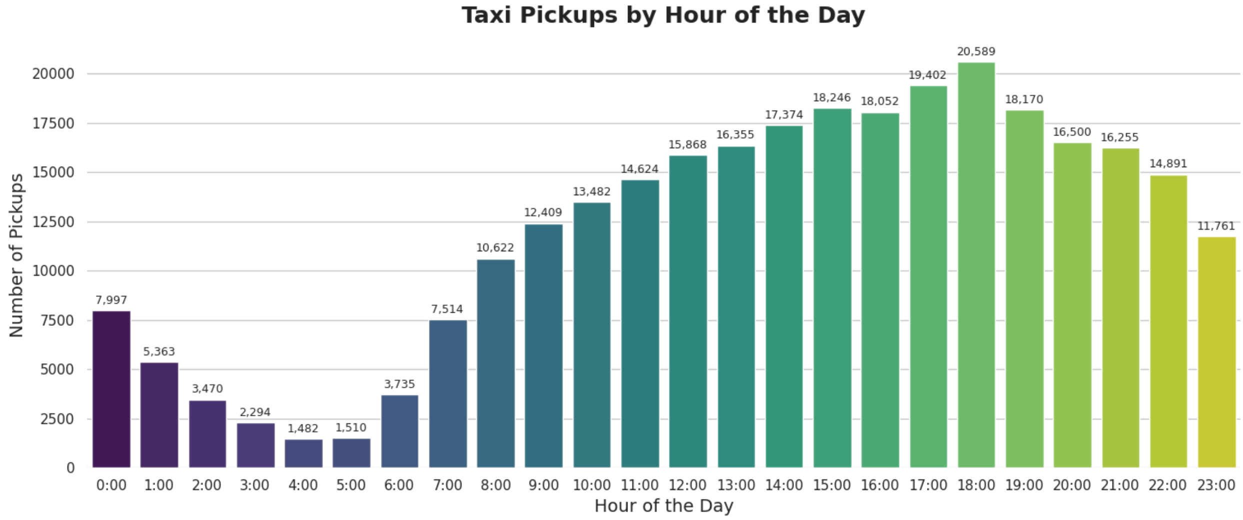
Temporal Analysis

3.1.2 [5 marks]

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

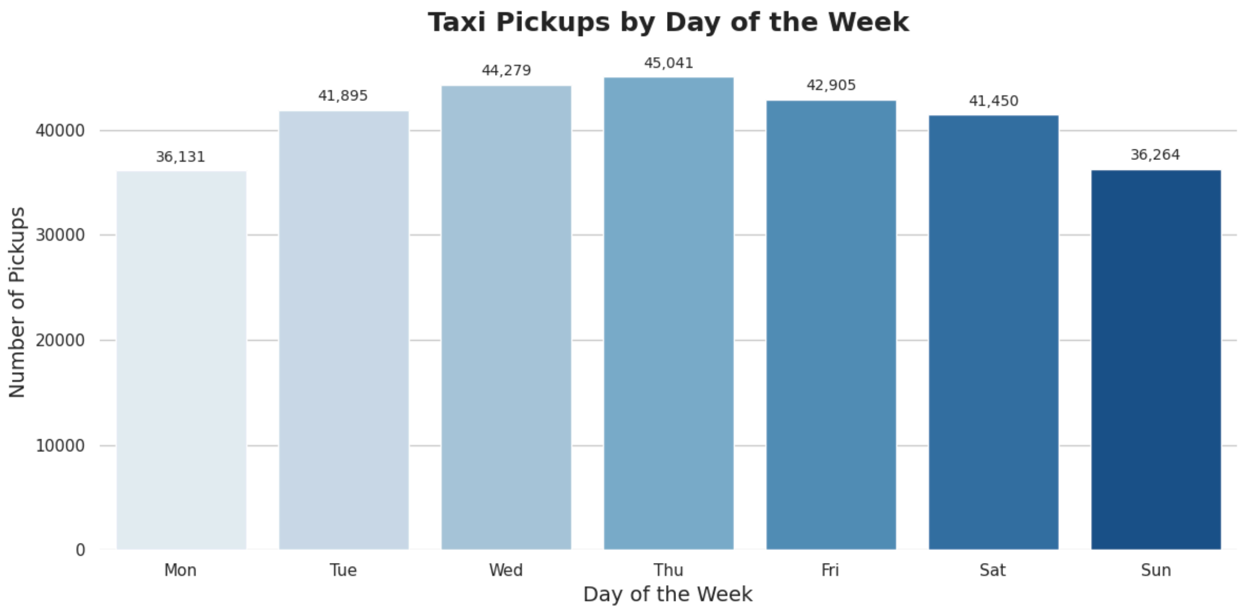
```
# Find and show the hourly trends in taxi pickups
cleaned_df_final['tpep_pickup_datetime'] = pd.to_datetime(
    cleaned_df_final['tpep_pickup_datetime'], errors='coerce'
)

# Extract time features
cleaned_df_final['pickup_hour'] = cleaned_df_final['tpep_pickup_datetime'].dt.hour
cleaned_df_final['pickup_day_of_week'] = cleaned_df_final['tpep_pickup_datetime'].dt.dayofweek
cleaned_df_final['pickup_month'] = cleaned_df_final['tpep_pickup_datetime'].dt.month

# Count pickups by hour (ensure all 24 hours appear)
pickup_counts_by_hour = (
    cleaned_df_final
    .groupby('pickup_hour')
    .size()
    .reindex(range(24), fill_value=0)
    .reset_index(name='count')
)

# Styling
sns.set_theme(style="whitegrid")

plt.figure(figsize=(14, 6))
ax = sns.barplot(
    x='pickup_hour',
    y='count',
    data=pickup_counts_by_hour,
    palette=sns.color_palette("viridis", 24)
)

# Titles & labels
ax.set_title('Taxi Pickups by Hour of the Day', fontsize=18, fontweight='bold', pad=15)
ax.set_xlabel('Hour of the Day', fontsize=14)
ax.set_ylabel('Number of Pickups', fontsize=14)

# Improve ticks
ax.set_xticks(range(24))
ax.set_xticklabels([f'{h}:00' for h in range(24)])
```

```
# Add value labels on top of bars
for p in ax.patches:
    ax.annotate(
        f'{int(p.get_height()):,}',
        (p.get_x() + p.get_width() / 2, p.get_height()),
        ha='center',
        va='bottom',
        fontsize=9,
        xytext=(0, 3),
        textcoords='offset points'
    )

# Clean up
sns.despine(left=True, bottom=True)
plt.tight_layout()
plt.show()
```

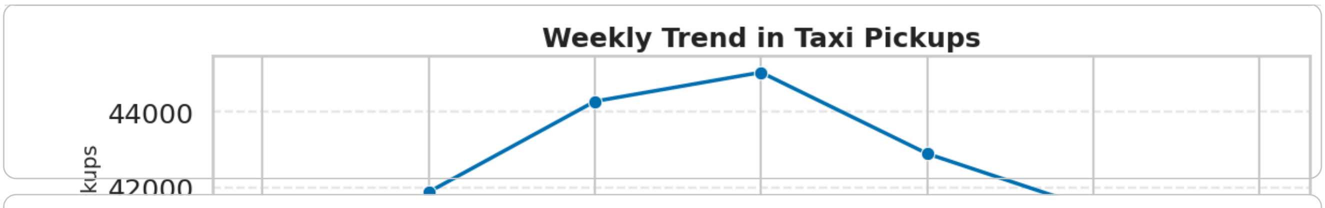



```
# Find and show the daily trends in taxi pickups (days of the week)

day_order = list(range(7))
day_labels = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun']

pickup_counts_by_day = (
    cleaned_df_final
    .groupby('pickup_day_of_week')
    .size()
    .reindex(day_order, fill_value=0)
    .reset_index(name='count')
)

sns.set_theme(style="whitegrid")

plt.figure(figsize=(12, 6))
ax = sns.barplot(
    x='pickup_day_of_week',
    y='count',
    data=pickup_counts_by_day,
    palette=sns.color_palette("Blues", 7)
)

# Titles and labels
ax.set_title('Taxi Pickups by Day of the Week', fontsize=18, fontweight='bold', pad=15)
ax.set_xlabel('Day of the Week', fontsize=14)
ax.set_ylabel('Number of Pickups', fontsize=14)

# Ticks
ax.set_xticks(day_order)
ax.set_xticklabels(day_labels)

# Value labels
for p in ax.patches:
    ax.annotate(
        f'{int(p.get_height()):,}',
```

```
            (p.get_x() + p.get_width() / 2, p.get_height()),
            ha='center',
            va='bottom',
            fontsize=10,
            xytext=(0, 4),
            textcoords='offset points'
        )

sns.despine(left=True, bottom=True)
plt.tight_layout()
plt.show()
```




```
sns.set_theme(
    style="whitegrid",
    context="talk",
    font_scale=1.1
)

plt.figure(figsize=(12, 5))
sns.lineplot(
    x='pickup_day_of_week',
    y='count',
    data=pickup_counts_by_day,
    marker='o',
    linewidth=2.5,
    color="#0072B2"  # clean, high-contrast blue
)

plt.title('Weekly Trend in Taxi Pickups', fontsize=18, fontweight='bold')
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Number of Pickups', fontsize=14)
plt.xticks(day_order, day_labels)
plt.grid(axis='y', linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```




```
# Show the monthly trends in pickups
month_order = list(range(1, 13))
month_labels = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

pickup_counts_by_month = (
    cleaned_df_final
    .groupby('pickup_month')
    .size()
    .reindex(month_order, fill_value=0)
    .reset_index(name='count')
)

sns.set_theme(style="whitegrid", context="talk")

plt.figure(figsize=(13, 6))
plt.fill_between(
    pickup_counts_by_month['pickup_month'],
    pickup_counts_by_month['count'],
    color="#56B4E9",
    alpha=0.6
)

plt.plot(
    pickup_counts_by_month['pickup_month'],
    pickup_counts_by_month['count'],
    color="#0072B2",
    linewidth=3,
    marker='o'
)

plt.title('Monthly Trend in Taxi Pickups', fontsize=18, fontweight='bold', pad=15)
plt.xlabel('Month', fontsize=14)
plt.ylabel('Number of Pickups', fontsize=14)

plt.xticks(month_order, month_labels)
```

```
([<matplotlib.axis.XTick at 0x7e1a0025ffb0>,
  <matplotlib.axis.XTick at 0x7e1a29b0da60>,
  <matplotlib.axis.XTick at 0x7e1a29b35d60>,
  <matplotlib.axis.XTick at 0x7e1a0029c320>,
  <matplotlib.axis.XTick at 0x7e1a0029cdd0>,
  <matplotlib.axis.XTick at 0x7e1a0029d880>,
  <matplotlib.axis.XTick at 0x7e1a00277a70>,
  <matplotlib.axis.XTick at 0x7e1a0029e0f0>,
  <matplotlib.axis.XTick at 0x7e1a0029eae0>,
  <matplotlib.axis.XTick at 0x7e1a0029f590>,
  <matplotlib.axis.XTick at 0x7e1a0029ffe0>,
  <matplotlib.axis.XTick at 0x7e1a0029ee10>],
 [Text(1, 0, 'Jan'),
  Text(2, 0, 'Feb'),
  Text(3, 0, 'Mar'),
  Text(4, 0, 'Apr'),
  Text(5, 0, 'May'),
  Text(6, 0, 'Jun'),
  Text(7, 0, 'Jul'),
  Text(8, 0, 'Aug'),
  Text(9, 0, 'Sep'),
  Text(10, 0, 'Oct'),
  Text(11, 0, 'Nov'),
  Text(12, 0, 'Dec')])
```

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

# Look for zero/negative values

cols_to_check = ['fare_amount', 'tip_amount', 'total_amount', 'trip_distance']

zero_counts = (cleaned_df_final[cols_to_check] == 0).sum()
negative_counts = (cleaned_df_final[cols_to_check] < 0).sum()

analysis = pd.DataFrame({
    "Zero Values": zero_counts,
    "Negative Values": negative_counts
})

print("Analysis of zero and negative values in financial and trip distance columns:\n")
print(analysis)
```

```
Analysis of zero and negative values in financial and trip distance columns:

               Zero Values  Negative Values
fare_amount             72                0
tip_amount           63117                0
total_amount            38                0
trip_distance         3272                0
```

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

```
df_clean  = cleaned_df_final.copy()
```

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.

cols_to_filter = ['fare_amount', 'tip_amount', 'total_amount']

for col in cols_to_filter:
    df_clean = df_clean[df_clean[col] != 0]

df_clean = df_clean[~(
    (df_clean['trip_distance'] == 0) &
    (df_clean['PULocationID'] != df_clean['DOLocationID'])
)]
```

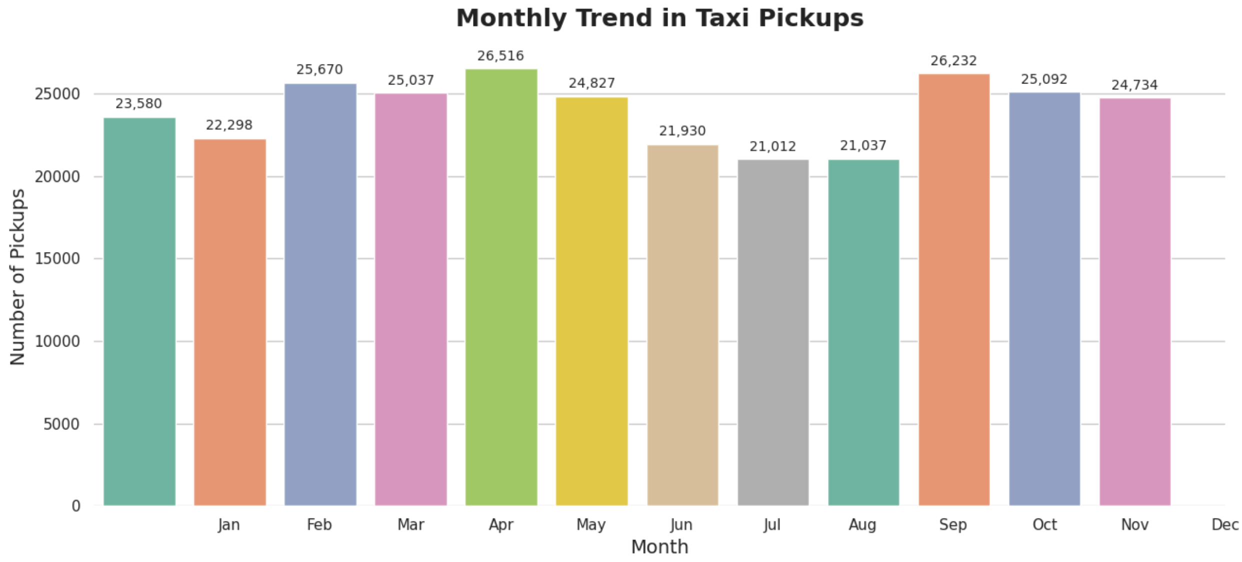
```
print("Original DataFrame shape:", cleaned_df_final.shape)
print("Non zeos and filtered DataFrame shape:", df_clean.shape)
```

```
Original DataFrame shape: (287965, 23)
Non zeos and filtered DataFrame shape: (224403, 23)
```

3.1.4 [3 marks]

Analyse the monthly revenue (`total_amount`) trend

```
# Group data by month and analyse monthly revenue
month_order = list(range(1, 13))
month_labels = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
                'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

pickup_counts_by_month = (
    cleaned_df_final
    .groupby('pickup_month')
    .size()
    .reindex(month_order, fill_value=0)
    .reset_index(name='count')
)

sns.set_theme(style="whitegrid")

plt.figure(figsize=(13, 6))
ax = sns.barplot(
    x='pickup_month',
    y='count',
    data=pickup_counts_by_month,
    palette=sns.color_palette("Set2", 12)
)

ax.set_title('Monthly Trend in Taxi Pickups', fontsize=18, fontweight='bold', pad=15)
ax.set_xlabel('Month', fontsize=14)
ax.set_ylabel('Number of Pickups', fontsize=14)

ax.set_xticks(month_order)
ax.set_xticklabels(month_labels)

# Value labels
for p in ax.patches:
    ax.annotate(
        f'{int(p.get_height()):,}',
        (p.get_x() + p.get_width() / 2, p.get_height()),
        ha='center',
        va='bottom',
        fontsize=10,
        xytext=(0, 4),
        textcoords='offset points'
    )

sns.despine(left=True, bottom=True)
plt.tight_layout()
plt.show()
```



3.1.5 [3 marks]

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

```
# Calculate proportion of each quarter
def determineQuarter(month):
    if month in [1, 2, 3]:
        return 'Q1'
    elif month in [4, 5, 6]:
        return 'Q2'
    elif month in [7, 8, 9]:
        return 'Q3'
    else:
        return 'Q4'

# Create quarter column
df_clean['quarter'] = df_clean['pickup_month'].apply(determineQuarter)

# Calculate quarterly revenue
quarterly_revenue = (
    df_clean
    .groupby('quarter')['total_amount']
    .sum()
    .reset_index()
)

# Ensure correct quarter order
quarter_order = ['Q1', 'Q2', 'Q3', 'Q4']
quarterly_revenue['quarter'] = pd.Categorical(
    quarterly_revenue['quarter'],
    categories=quarter_order,
    ordered=True
)
quarterly_revenue = quarterly_revenue.sort_values('quarter')

# Calculate proportion
total_revenue = quarterly_revenue['total_amount'].sum()
quarterly_revenue['proportion'] = quarterly_revenue['total_amount'] / total_revenue

print("Quarterly revenue and proportion of total revenue:\n")
print(quarterly_revenue)

# -------------------- DONUT CHART --------------------
sns.set_style("white")

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

colors = ['#66c2a5', '#fc8d62', '#8da0cb', '#e78ac3']

wedges, texts, autotexts = plt.pie(
    quarterly_revenue['proportion'],
```

```python
    labels=quarterly_revenue['quarter'],
    autopct='%1.1f%%',
    startangle=90,
    colors=colors,
    explode=(0.03, 0.03, 0.03, 0.03),
    wedgeprops={'edgecolor': 'white', 'linewidth': 2},
    textprops={'fontsize': 12}
)

# Create center circle for donut
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
plt.gca().add_artist(centre_circle)

for autotext in autotexts:
    autotext.set_fontweight('bold')

plt.title(
    'Proportion of Revenue by Quarter',
    fontsize=16,
    fontweight='bold',
    pad=20
)

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

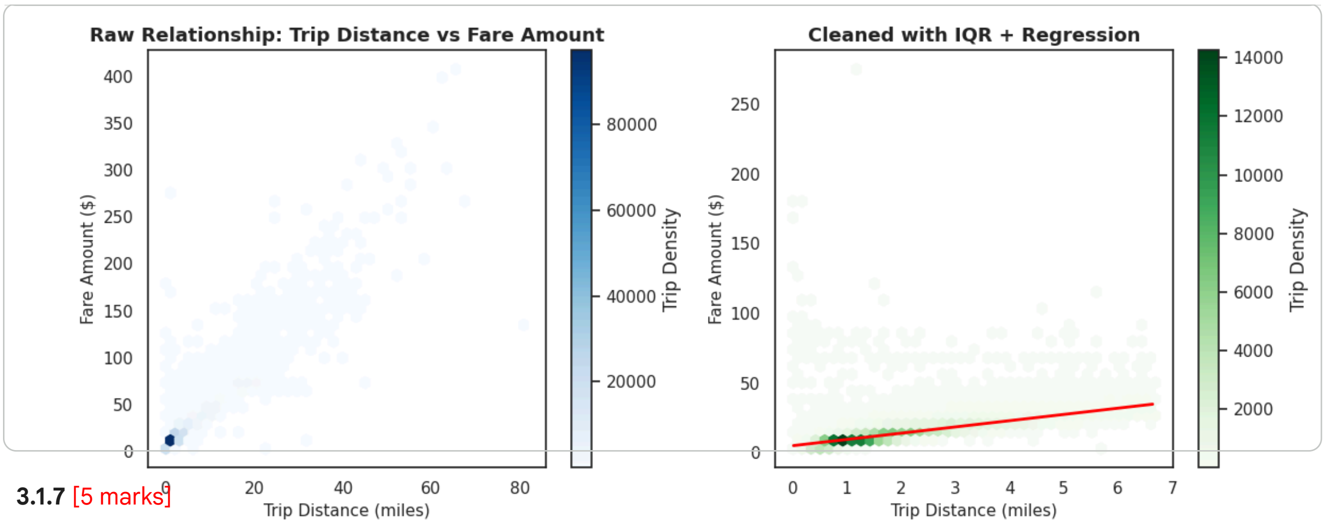
```
Quarterly revenue and proportion of total revenue:

  quarter  total_amount  pro
0      Q1    1599839.60  0.240278
1      Q2    1772396.50  0.266194
2      Q3    1479544.98  0.222211
3      Q4    1806510.63  0.271317
```

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

df_clean['tpep_pickup_datetime'] = pd.to_datetime(df_clean['tpep_pickup_datetime'], errors='coerce')
filteredDF = df_clean[(df_clean['trip_distance'] > 0) & (df_clean['fare_amount'] > 0)]

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.hexbin(
    filteredDF['trip_distance'],
    filteredDF['fare_amount'],
    gridsize=40,
    cmap='Blues',
    mincnt=1
)
plt.colorbar(label='Trip Density')
plt.title("Raw Relationship: Trip Distance vs Fare Amount", fontsize=13, fontweight='bold')
plt.xlabel("Trip Distance (miles)", fontsize=11)
plt.ylabel("Fare Amount ($)", fontsize=11)

# IQR-based outlier removal
Q1 = filteredDF['trip_distance'].quantile(0.25)
Q3 = filteredDF['trip_distance'].quantile(0.75)
IQR = Q3 - Q1
lowerBound = Q1 - 1.5 * IQR
upperBound = Q3 + 1.5 * IQR

cleanedData = filteredDF[
    (filteredDF['trip_distance'] >= lowerBound) &
    (filteredDF['trip_distance'] <= upperBound)
]

plt.subplot(1, 2, 2)
sns.regplot(
    data=cleanedData.sample(min(5000, len(cleanedData)), random_state=42),
    x='trip_distance', y='fare_amount',
    scatter=False,
    line_kws={'color': 'red', 'linewidth': 2}
)

plt.hexbin(
    cleanedData['trip_distance'],
    cleanedData['fare_amount'],
    gridsize=40,
    cmap='Greens',
    mincnt=1
)
plt.colorbar(label='Trip Density')
plt.title("Cleaned with IQR + Regression", fontsize=13, fontweight='bold')
plt.xlabel("Trip Distance (miles)", fontsize=11)
plt.ylabel("Fare Amount ($)", fontsize=11)

plt.tight_layout()
plt.show()

raw_corr = filteredDF['trip_distance'].corr(filteredDF['fare_amount'])
clean_corr = cleanedData['trip_distance'].corr(cleanedData['fare_amount'])

print(f"Correlation (Raw data): {raw_corr:.2f}")
print(f"Correlation (After IQR cleaning): {clean_corr:.2f}")
```




```
Correlation (Raw data): 0.95
Correlation (After IQR cleaning): 0.81
```

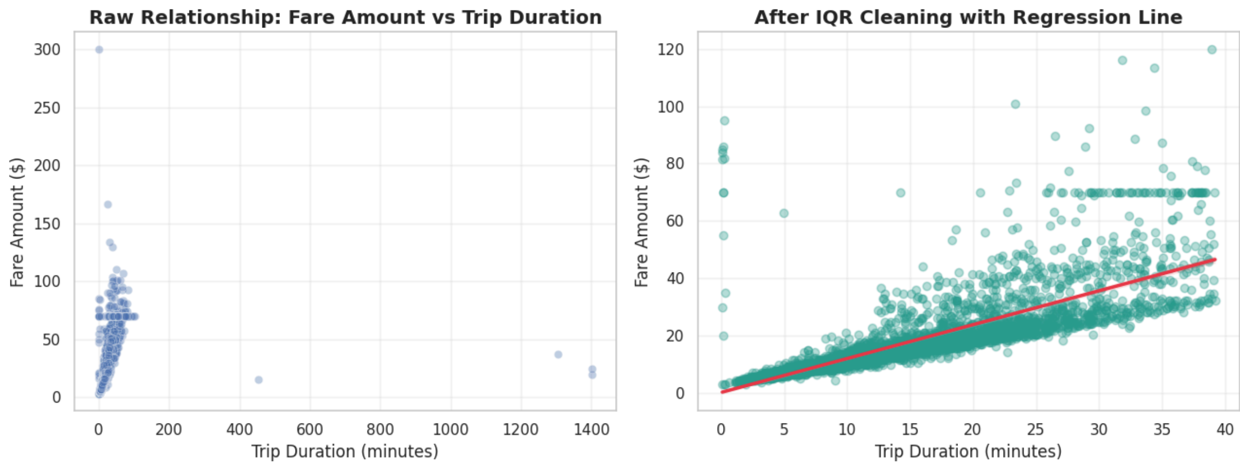
3.1.7 [5 marks]

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`

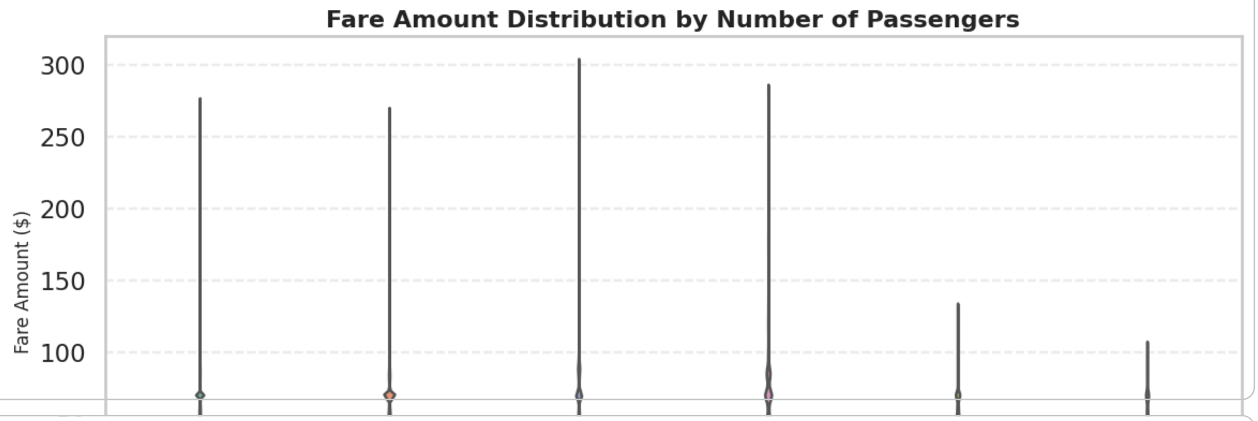
```
# Show relationship between fare and trip duration

# Ensure datetime columns are properly formatted
df_clean['tpep_pickup_datetime'] = pd.to_datetime(df_clean['tpep_pickup_datetime'], errors='coerce')
df_clean['tpep_dropoff_datetime'] = pd.to_datetime(df_clean['tpep_dropoff_datetime'], errors='coerce')

# Calculate trip duration in minutes
df_clean['trip_duration'] = (df_clean['tpep_dropoff_datetime'] - df_clean['tpep_pickup_datetime']).dt.total_sec

# Filter for positive fare and trip duration
filteredDF = df_clean[(df_clean['fare_amount'] > 0) & (df_clean['trip_duration'] > 0)]

sns.set_theme(style="whitegrid")

plt.figure(figsize=(13, 5))

# -------- Left: Raw relationship --------
plt.subplot(1, 2, 1)
sns.scatterplot(
    data=filteredDF.sample(min(5000, len(filteredDF)), random_state=42),
    x='trip_duration',
    y='fare_amount',
    alpha=0.35,
    color='#4C72B0'
)
plt.title("Raw Relationship: Fare Amount vs Trip Duration", fontsize=14, fontweight='bold')
plt.xlabel("Trip Duration (minutes)", fontsize=12)
plt.ylabel("Fare Amount ($)", fontsize=12)
plt.grid(alpha=0.25)

# -------- IQR outlier removal --------
Q1 = filteredDF['trip_duration'].quantile(0.25)
Q3 = filteredDF['trip_duration'].quantile(0.75)
IQR = Q3 - Q1
lowerBound = Q1 - 1.5 * IQR
upperBound = Q3 + 1.5 * IQR

cleanedData = filteredDF[
    (filteredDF['trip_duration'] >= lowerBound) &
    (filteredDF['trip_duration'] <= upperBound)
]

# -------- Right: Cleaned + regression --------
plt.subplot(1, 2, 2)
sns.regplot(
    data=cleanedData.sample(min(5000, len(cleanedData)), random_state=42),
    x='trip_duration',
    y='fare_amount',
    scatter_kws={'alpha': 0.35, 'color': '#2A9D8F'},
```

```
    line_kws={'color': '#E63946', 'linewidth': 2.5}
)
plt.title("After IQR Cleaning with Regression Line", fontsize=14, fontweight='bold')
plt.xlabel("Trip Duration (minutes)", fontsize=12)
plt.ylabel("Fare Amount ($)", fontsize=12)
plt.grid(alpha=0.25)

plt.tight_layout()
plt.show()

# Correlation comparison
raw_corr = filteredDF['trip_duration'].corr(filteredDF['fare_amount'])
clean_corr = cleanedData['trip_duration'].corr(cleanedData['fare_amount'])

print(f"Correlation (Raw data): {raw_corr:.2f}")
print(f"Correlation (After IQR cleaning): {clean_corr:.2f}")
```

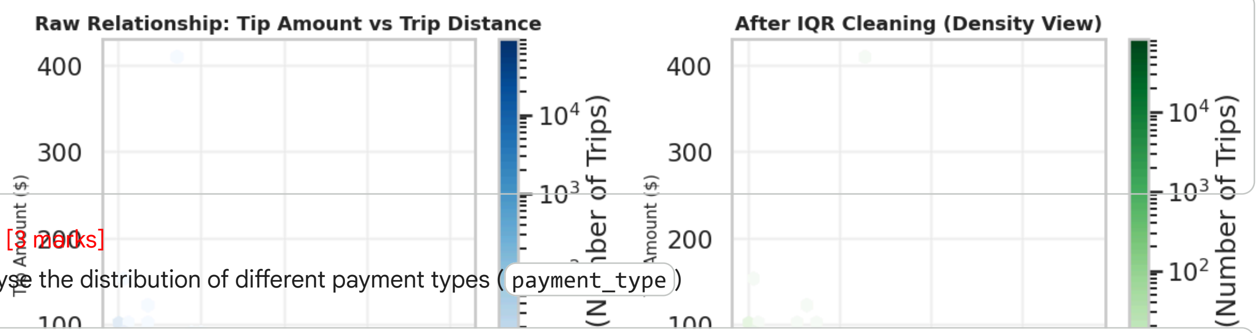



```
Correlation (Raw data): 0.34
Correlation (After IQR cleaning): 0.77
```

```
# Show relationship between fare and number of passengers
plt.figure(figsize=(12, 6))

sns.violinplot(
    data=cleanedData,
    x='passenger_count',
    y='fare_amount',
    palette='Set2',
    inner='quartile'
)

plt.title(
    'Fare Amount Distribution by Number of Passengers',
    fontsize=16,
    fontweight='bold'
)
plt.xlabel('Number of Passengers', fontsize=12)
plt.ylabel('Fare Amount ($)', fontsize=12)
plt.grid(axis='y', alpha=0.3, linestyle='--')

plt.tight_layout()
plt.show()

corr_passenger_fare = cleanedData['fare_amount'].corr(cleanedData['passenger_count'])
print(f'Correlation between Fare Amount and Passenger Count: {corr_passenger_fare:.2f}')
```




```
# Show relationship between tip and trip distance
plt.figure(figsize=(12, 5))

# -------- Left: Raw data (high-contrast hexbin) --------
plt.subplot(1, 2, 1)
plt.hexbin(
    filteredDF['trip_distance'],
    filteredDF['tip_amount'],
    gridsize=35,
    cmap='Blues',   # clean, professional
    mincnt=1,
    bins='log'
)
plt.colorbar(label='Log(Number of Trips)')
plt.title("Raw Relationship: Tip Amount vs Trip Distance", fontsize=13, fontweight='bold')
plt.xlabel("Trip Distance (miles)", fontsize=11)
plt.ylabel("Tip Amount ($)", fontsize=11)
plt.grid(alpha=0.2)

# -------- Right: Cleaned data (high-contrast hexbin) --------
plt.subplot(1, 2, 2)
plt.hexbin(
    cleanedData['trip_distance'],
    cleanedData['tip_amount'],
    gridsize=35,
    cmap='Greens',   # clean, professional
    mincnt=1,
    bins='log'
)
plt.colorbar(label='Log(Number of Trips)')
plt.title("After IQR Cleaning (Density View)", fontsize=13, fontweight='bold')
plt.xlabel("Trip Distance (miles)", fontsize=11)
plt.ylabel("Tip Amount ($)", fontsize=11)
plt.grid(alpha=0.2)

plt.tight_layout()
plt.show()

raw_corr = filteredDF['trip_distance'].corr(filteredDF['tip_amount'])
clean_corr = cleanedData['trip_distance'].corr(cleanedData['tip_amount'])

print(f"Correlation (Raw data): {raw_corr:.2f}")
print(f"Correlation (After IQR cleaning): {clean_corr:.2f}")
```

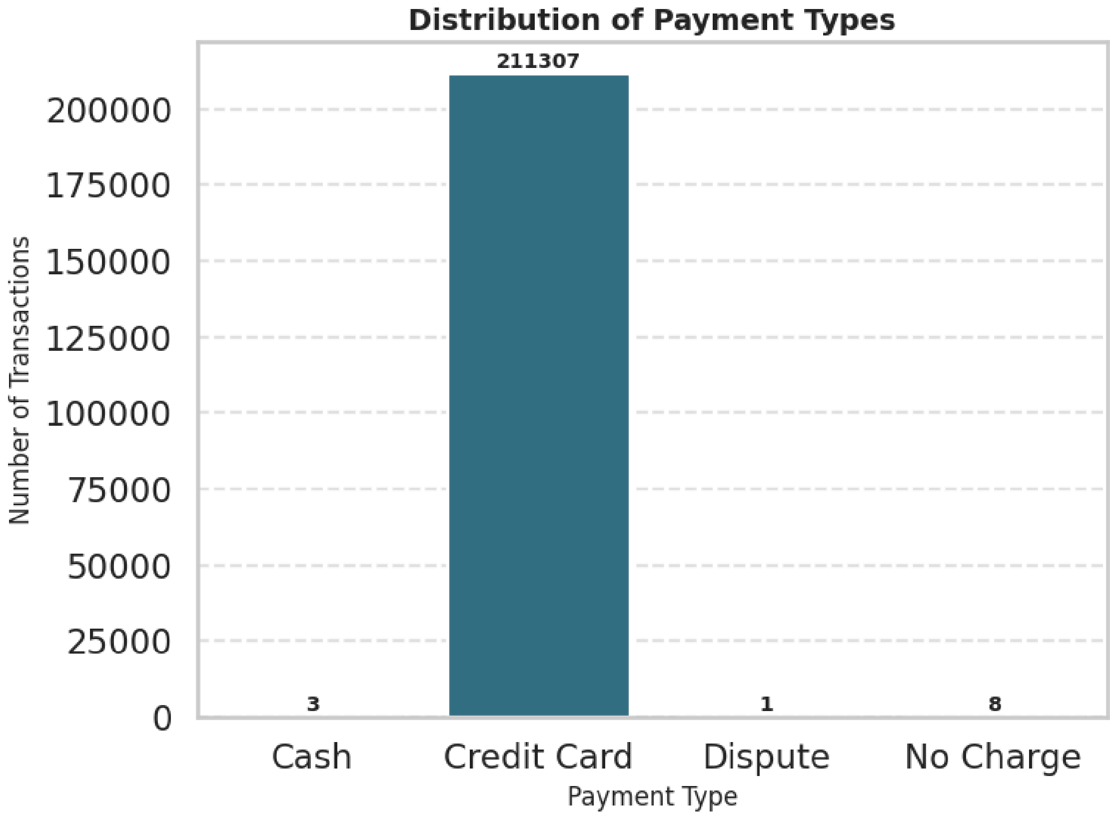


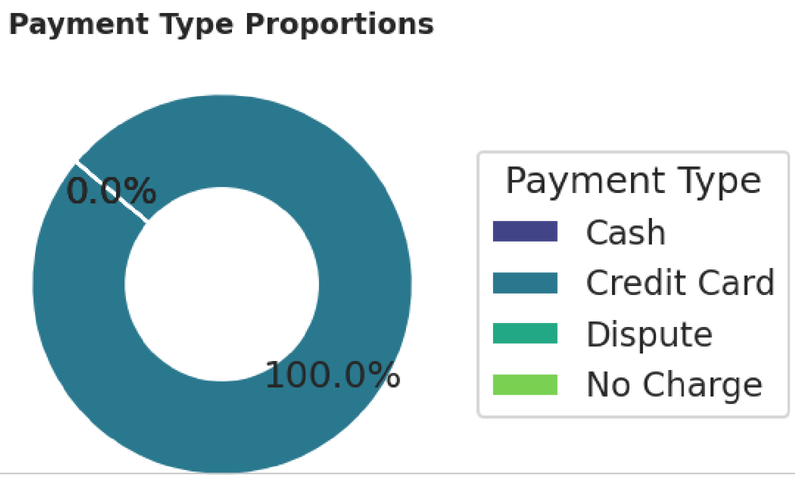
3.1.8 [3 marks]

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

```
# Analyse the distribution of different payment types (payment_type).
payment_type_mapping = {
    1: "Credit Card",
    2: "Cash",
    3: "No Charge",
    4: "Dispute"
}
cleanedData['payment_type_name'] = cleanedData['payment_type'].map(payment_type_mapping)

# Calculate counts and percentages
payment_counts = cleanedData['payment_type_name'].value_counts().sort_index()
payment_percent = cleanedData['payment_type_name'].value_counts(normalize=True).sort_index() * 100

print("Counts of Payment Types:")
print(payment_counts)
print("\nPercentage Distribution of Payment Types:")
print(payment_percent.round(2))

plt.figure(figsize=(8,6))
sns.barplot(x=payment_counts.index, y=payment_counts.values, palette="viridis")
plt.title("Distribution of Payment Types", fontsize=14, fontweight="bold")
plt.xlabel("Payment Type", fontsize=12)
plt.ylabel("Number of Transactions", fontsize=12)

# Annotate bar counts
for i, val in enumerate(payment_counts.values):
    plt.text(i, val + (0.01 * payment_counts.max()), str(val),
             ha='center', fontsize=10, fontweight="bold")

plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
plt.figure(figsize=(6,6))
colors = sns.color_palette("viridis", len(payment_counts))

wedges, texts, autotexts = plt.pie(
    payment_percent,
    labels=None,
    autopct='%1.1f%%',
    startangle=140,
    colors=colors,
    pctdistance=0.75
)

centre_circle = plt.Circle((0,0),0.50,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.legend(
    wedges,
    payment_counts.index,
    title="Payment Type",
    loc="center left",
    bbox_to_anchor=(1, 0, 0.5, 1)
)

plt.title("Payment Type Proportions", fontsize=14, fontweight="bold")
plt.tight_layout()
plt.show()
```

```
Counts of Payment Types:
payment_type_name
Cash                3
Credit Card    211307
Dispute             1
No Charge           8
Name: count, dtype: int64

Percentage Distribution of Payment Types:
payment_type_name
Cash            0.00
Credit Card    99.99
Dispute         0.00
No Charge       0.00
Name: proportion, dtype: float64
```





- 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 /usr/local/lib/python3.12/dist-packages (1.1.2)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.12/dist-packages (from geopandas) (2.0.2)
Requirement already satisfied: pyogrio>=0.7.2 in /usr/local/lib/python3.12/dist-packages (from geopandas) (0.1
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from geopandas) (26.0)
Requirement already satisfied: pandas>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from geopandas) (2.2.
Requirement already satisfied: pyproj>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from geopandas) (3.7.
Requirement already satisfied: shapely>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from geopandas) (2.1
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas>
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=2.0.0->ge
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=2.0.0->
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from pyogrio>=0.7.2->geopan
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.
```

3.1.9 [2 marks]

Load the shapefile and display it.

```
import geopandas as gpd


# Read the shapefile using geopandas
zones = gpd.read_file('/content/drive/My Drive/EDA/taxi_zones/taxi_zones.shp')
zones.head()
```

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

Next steps: Generate code with zones New interactive sheet

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

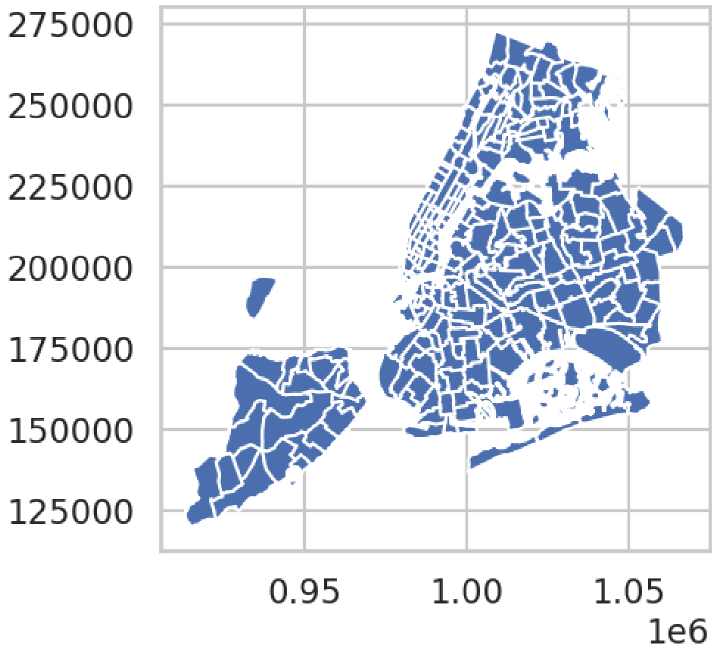
Now, the `locationID` here is also what we are using to mark pickup and drop zones in the trip records.

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
mergedData = pd.merge(
    cleanedData,
    zones,
    left_on='PULocationID',
    right_on='LocationID',
    how='inner'
)

print("Merged DataFrame preview:")
mergedData.head()
```

Merged DataFrame preview:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_anc
0	2	2023-10-02 18:54:59	2023-10-02 19:04:20	1.0	2.10	1.0	
1	2	2023-03-15 17:18:43	2023-03-15 17:44:54	1.0	4.38	1.0	
2	1	2023-05-10 07:19:08	2023-05-10 07:30:21	1.0	4.20	1.0	
3	1	2023-11-08 15:48:24	2023-11-08 16:01:36	1.0	0.60	1.0	
4	2	2023-02-21 16:09:34	2023-02-21 16:20:37	1.0	0.78	1.0	

5 rows × 33 columns

3.1.11 [3 marks]

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

```
# Group data by location and calculate the number of trips
trip_count_by_location = mergedData.groupby('LocationID').size().reset_index(name='total_num_trips')
print("Total trips per pickup location:")
trip_count_by_location
```

Total trips per pickup location:

	LocationID	total_num_trips
0	1	24
1	4	200
2	6	1
3	7	45
4	8	1
...	...	...
170	259	2
171	260	20
172	261	973
173	262	3012
174	263	4297

175 rows × 2 columns

Next steps: Generate code with `trip_count_by_location` New interactive sheet

3.1.12 [2 marks]

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

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

```
# Merge trip counts back to the zones GeoDataFrame
mergedTripCountsData = pd.merge(zones, trip_count_by_location, how='left', on='LocationID')
mergedTripCountsData
```

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry	total_num_trip
0	1	0.116357	0.000782	Newark Airport	1	EWR	POLYGON ((933100.918 192536.086, 933091.011 19...	24.
1	2	0.433470	0.004866	Jamaica Bay	2	Queens	MULTIPOLYGON (((1033269.244 172126.008, 103343...	NaI
2	3	0.084341	0.000314	Allerton/Pelham Gardens	3	Bronx	POLYGON ((1026308.77 256767.698, 1026495.593 2...	NaI
3	4	0.043567	0.000112	Alphabet City	4	Manhattan	POLYGON ((992073.467 203714.076, 992068.667 20...	200.
4	5	0.092146	0.000498	Arden Heights	5	Staten Island	POLYGON ((935843.31 144283.336, 936046.565 144...	NaI
...	...	...	...	...	...	...	...	.
258	259	0.126750	0.000395	Woodlawn/Wakefield	259	Bronx	POLYGON ((1025414.782 270986.139, 1025138.624 ...	2.
259	260	0.133514	0.000422	Woodside	260	Queens	POLYGON ((1011466.966 216463.005, 1011545.889 ...	20.
260	261	0.027120	0.000034	World Trade Center	261	Manhattan	POLYGON ((980555.204 196138.486, 980570.792 19...	973.
261	262	0.049064	0.000122	Yorkville East	262	Manhattan	MULTIPOLYGON (((999804.795 224498.527, 999824....	3012.
262	263	0.037017	0.000066	Yorkville West	263	Manhattan	POLYGON ((997493.323 220912.386, 997355.264 22...	4297.

263 rows × 8 columns

Next steps: Generate code with `mergedTripCountsData` New interactive sheet

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

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

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

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

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

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

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

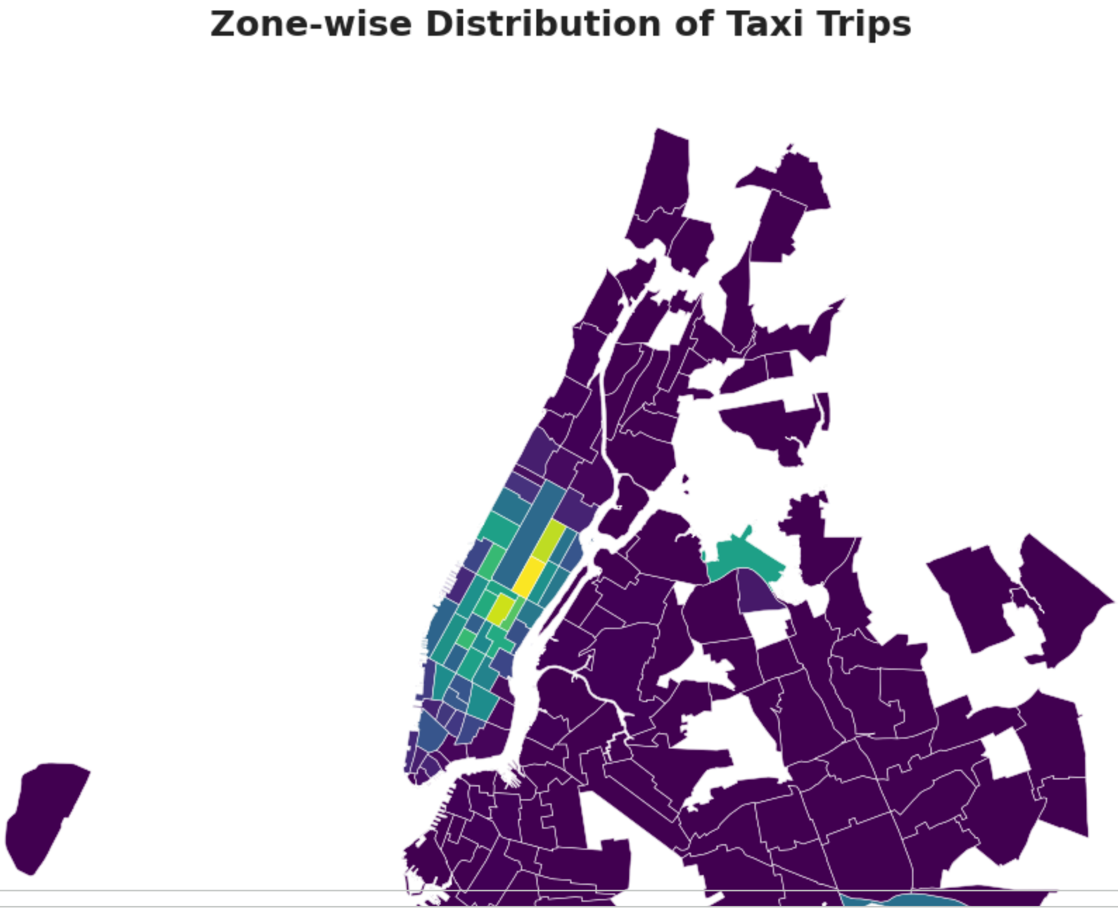
```
column = '',
ax = ax,
legend = True,
legend_kwds = {'label': "label", 'orientation': "<horizontal/vertical>"}
```

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

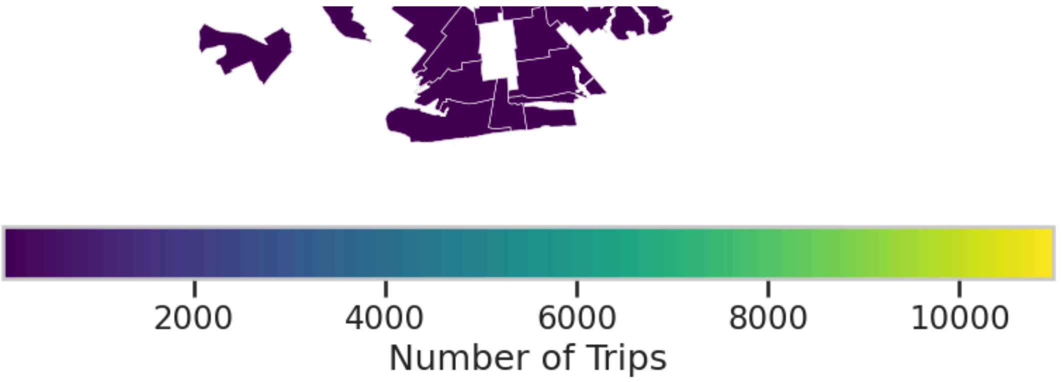
3.1.13 [3 marks]

Plot a color-coded map showing zone-wise trips

```
# Define figure and axis
fig, ax = plt.subplots(1, 1, figsize=(13, 11))

# Plot choropleth map
mergedTripCountsData.plot(
    column='total_num_trips',
    ax=ax,
    cmap='viridis',                    # clean, professional, smooth gradient
    legend=True,
    linewidth=0.4,
    edgecolor='white',                 # lighter edge for modern look
    legend_kwds={
        'label': "Number of Trips",
        'orientation': "horizontal",
        'shrink': 0.6,
        'pad': 0.02
    }
)

# Remove axes for cleaner map
ax.set_axis_off()

# Title
ax.set_title(
    "Zone-wise Distribution of Taxi Trips",
    fontsize=18,
    fontweight='bold',
    pad=20
)

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




```
# can you try displaying the zones DF sorted by the number of trips?
sortedZones = mergedTripCountsData.sort_values(by='total_num_trips', ascending=False)
sortedZones
```



	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry	total_num_trips
236	237	0.042213	0.000096	Upper East Side South	237	Manhattan	POLYGON ((993633.442 216961.016, 993507.232 21...	10983.0
160	161	0.035804	0.000072	Midtown Center	161	Manhattan	POLYGON ((991081.026 214453.698, 990952.644 21...	10155.0
235	236	0.044252	0.000103	Upper East Side North	236	Manhattan	POLYGON ((995940.048 221122.92, 995812.322 220...	9996.0
161	162	0.035270	0.000048	Midtown East	162	Manhattan	POLYGON ((992224.354 214415.293, 992096.999 21...	8134.0
185	186	0.024696	0.000037	Penn Station/Madison Sq West	186	Manhattan	POLYGON ((986752.603 210853.699, 986627.863 21...	7500.0
...	...	...	...	...	...	...	...	...
244	245	0.095983	0.000466	West Brighton	245	Staten Island	POLYGON ((957085.564 172591.26, 957142.385 172...	NaN
249	250	0.079626	0.000241	Westchester Village/Unionport	250	Bronx	POLYGON ((1026991.885 245048.602, 1027255.054 ...	NaN
250	251	0.137711	0.000626	Westerleigh	251	Staten Island	POLYGON ((947868.004 169247.734, 948000.981 16...	NaN
251	252	0.158004	0.001025	Whitestone	252	Queens	POLYGON ((1033946.683 231157.996,	NaN

Next steps: Generate code with sortedZones New interactive sheet

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
# Convert pickup and dropoff datetime columns to proper datetime format
mergedData['tpep_pickup_datetime'] = pd.to_datetime(mergedData['tpep_pickup_datetime'])
mergedData['tpep_dropoff_datetime'] = pd.to_datetime(mergedData['tpep_dropoff_datetime'])


mergedData['pickup_hour_of_day'] = mergedData['tpep_pickup_datetime'].dt.hour
mergedData['trip_duration_min'] = (mergedData['tpep_dropoff_datetime'] - mergedData['tpep_pickup_datetime']).dt

hourly_route_stats = mergedData.groupby(
    ['pickup_hour_of_day', 'PULocationID', 'DOLocationID']
).agg(
    avg_duration=('trip_duration_min', 'mean'),
    avg_distance=('trip_distance', 'mean')
).reset_index()

# Compute average speed for each route
hourly_route_stats['avg_speed_miles_per_min'] = hourly_route_stats['avg_distance'] / hourly_route_stats['avg_du
```

```
# Sort by hour and speed to identify slowest routes
slowest_routes_by_hour = hourly_route_stats.sort_values(by=['pickup_hour_of_day', 'avg_speed_miles_per_min'], a

# Select top 5 slowest routes per hour
top5_slowest_routes = slowest_routes_by_hour.groupby('pickup_hour_of_day').head(5)

# Display results
print("Slowest routes for each hour of the day (avg speed in miles/min):")
top5_slowest_routes[['pickup_hour_of_day', 'PULocationID', 'DOLocationID', 'avg_speed_miles_per_min']]
```

```
Slowest routes for each hour of the day (avg speed in miles/min):
```

	pickup_hour_of_day	PULocationID	DOLocationID	avg_speed_miles_per_min
29	0	24	24	0.0
35	0	41	41	0.0
39	0	42	42	0.0
339	0	87	87	0.0
711	0	129	129	0.0
...	...	...	...	...
47952	23	1	1	0.0
48005	23	42	42	0.0
48915	23	133	133	0.0
49727	23	167	167	0.0
50142	23	232	232	0.0

120 rows × 4 columns

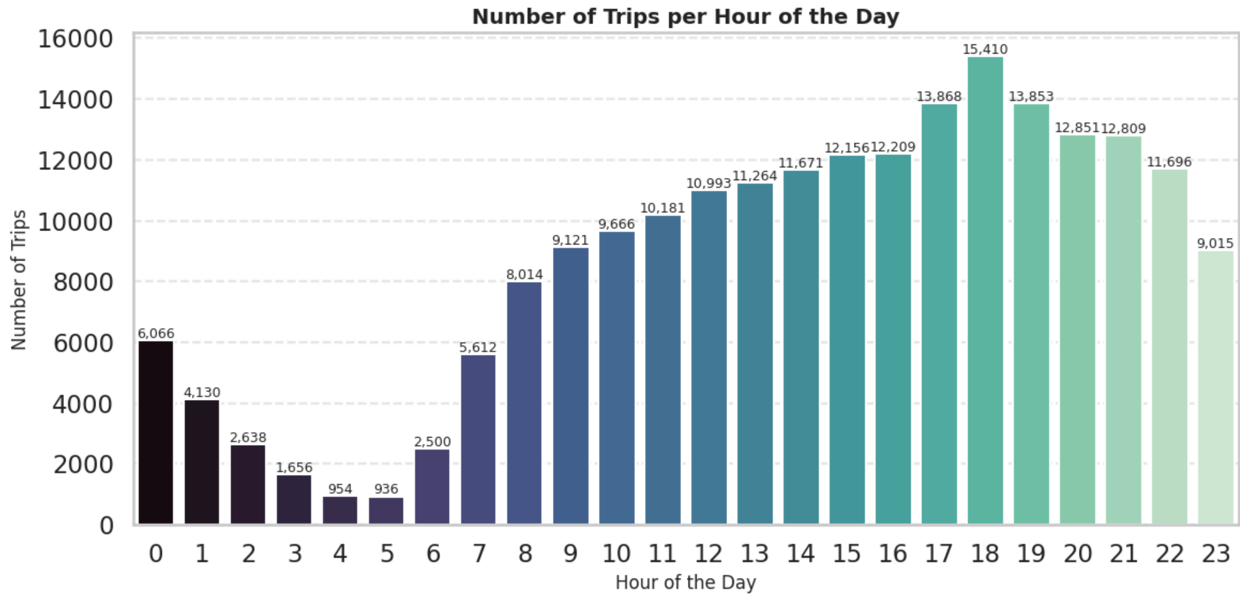
How does identifying high-traffic, high-demand routes help us?

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
hourly_trips = mergedData.groupby('pickup_hour_of_day').size().reset_index(name='num_trips')

# Convert hour values to integer
hourly_trips['pickup_hour_of_day'] = hourly_trips['pickup_hour_of_day'].astype(int)

# Visualize the number of trips per hour
plt.figure(figsize=(12, 6))
sns.barplot(
    x='pickup_hour_of_day',
    y='num_trips',
    data=hourly_trips,
    palette='mako'  # clean, modern sequential palette
)

plt.title('Number of Trips per Hour of the Day', fontsize=14, fontweight='bold')
plt.xlabel('Hour of the Day', fontsize=12)
plt.ylabel('Number of Trips', fontsize=12)
plt.xticks(range(24))
plt.grid(axis='y', linestyle='--', alpha=0.4)  # lighter, cleaner grid

# Bar annotation with value
for index, row in hourly_trips.iterrows():
    plt.text(
        row['pickup_hour_of_day'],
        row['num_trips'] + (0.005 * hourly_trips['num_trips'].max()),
        f"{row['num_trips']:,}",
        ha='center',
        fontsize=9
    )
```

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

# Find the busiest hour
busiest = hourly_trips.loc[hourly_trips['num_trips'].idxmax()]
print(f"Peak activity detected: Hour {busiest['pickup_hour_of_day']} with {busiest['num_trips']:,} trips.")
```




```
Peak activity detected: Hour 18 with 15,410 trips.
```

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

3.2.3 [2 mark]

Find the actual number of trips in the five busiest hours

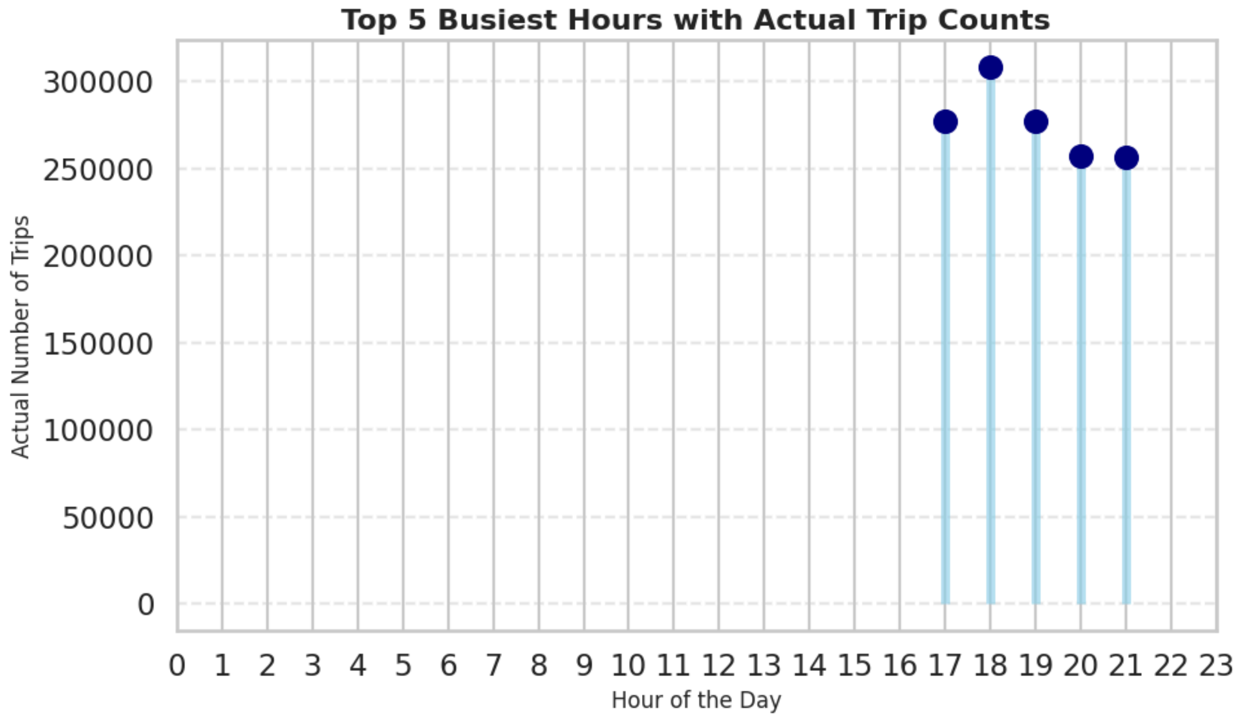
```
hourly_trips.columns.tolist()
```

```
['pickup_hour_of_day', 'num_trips']
```

```
# Scale up the number of trips
samplingRatio = 0.05

# Scale the number of trips up by the sampling ratio
hourly_trips['actual_num_trips'] = hourly_trips['num_trips'] / samplingRatio
busiest_hours = hourly_trips.sort_values(by='actual_num_trips', ascending=False).head(5)

print("The five busiest hours with actual trip counts are:")
print(busiest_hours[['pickup_hour_of_day', 'actual_num_trips']])

# -------- Lollipop-style plot --------
plt.figure(figsize=(10, 6))

# Draw lines
plt.vlines(
    x=busiest_hours['pickup_hour_of_day'],
    ymin=0,
    ymax=busiest_hours['actual_num_trips'],
    color='skyblue',
    linewidth=5,
    alpha=0.6
)

# Draw points on top
plt.scatter(
    busiest_hours['pickup_hour_of_day'],
    busiest_hours['actual_num_trips'],
    color='navy',
    s=150,   # bigger markers
```

```
    s=150,  # bigger markers
    zorder=5
)

plt.title('Top 5 Busiest Hours with Actual Trip Counts', fontsize=16, fontweight='bold')
plt.xlabel('Hour of the Day', fontsize=12)
plt.ylabel('Actual Number of Trips', fontsize=12)
plt.xticks(range(24))
plt.grid(axis='y', linestyle='--', alpha=0.4)

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

```
The five busiest hours with actual trip counts are:
    pickup_hour_of_day  actual_num_trips
18                  18          308200.0
17                  17          277360.0
19                  19          277060.0
20                  20          257020.0
21                  21          256180.0
```

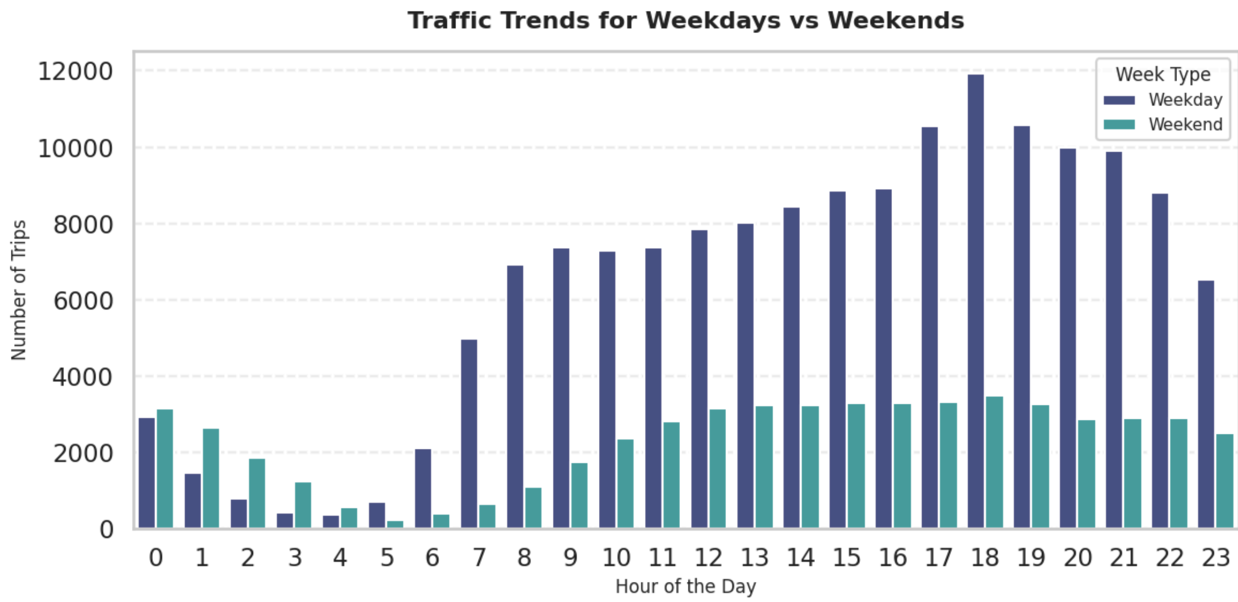


3.2.4 [3 marks]

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

```
# Compare traffic trends for the week days and weekends
mergedData['day_of_week'] = mergedData['tpep_pickup_datetime'].dt.weekday

# Categorize into weekdays (0-4) and weekends (5-6)
mergedData['week_type'] = mergedData['day_of_week'].apply(lambda x: 'Weekday' if x < 5 else 'Weekend')

tripsByWeektype = mergedData.groupby(['week_type', 'pickup_hour_of_day']).size().reset_index(name='num_trips_we
trips_pivot = tripsByWeektype.pivot(index='pickup_hour_of_day', columns='week_type', values='num_trips_week_typ

# -------- Cleaned Grouped Bar Plot --------
plt.figure(figsize=(12, 6))
sns.set_theme(style="whitegrid", context="talk")  # clean background and font sizing

sns.barplot(
    x='pickup_hour_of_day',
    y='num_trips_week_type',
    hue='week_type',
    data=tripsByWeektype,
    palette='mako'  # clean, professional sequential colors
)

plt.title('Traffic Trends for Weekdays vs Weekends', fontsize=16, fontweight='bold', pad=15)
```

```
plt.title('Traffic Trends for Weekdays vs Weekends', fontsize=16, fontweight='bold', pad=15)
plt.xlabel('Hour of the Day', fontsize=12)
plt.ylabel('Number of Trips', fontsize=12)
plt.xticks(range(24))
plt.grid(axis='y', linestyle='--', alpha=0.3)  # subtle grid
plt.legend(title='Week Type', fontsize=11, title_fontsize=12)
plt.tight_layout()
plt.show()
```



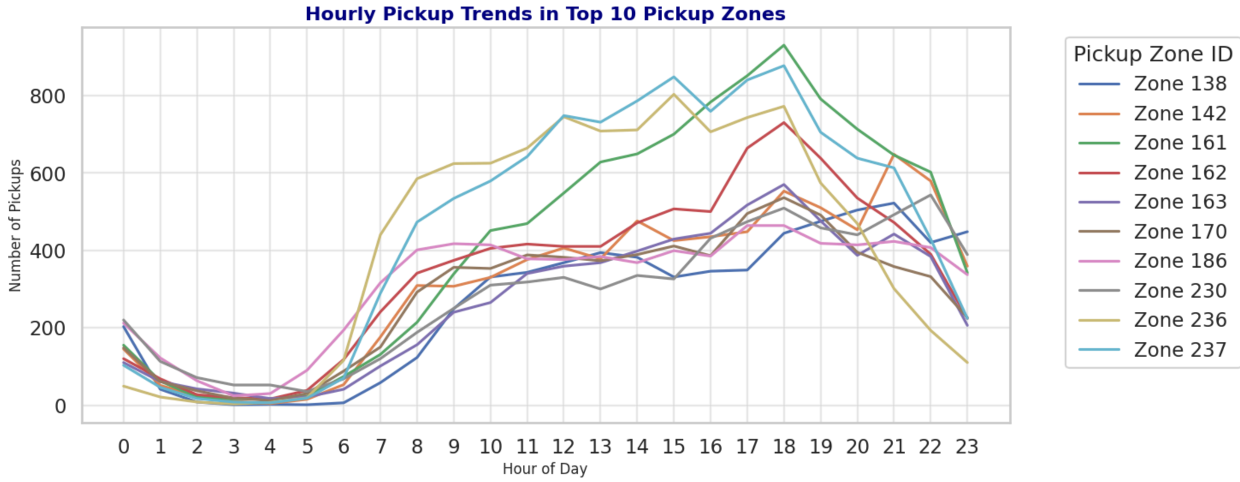
What can you infer from the above patterns? How will finding busy and quiet hours for each day help us?

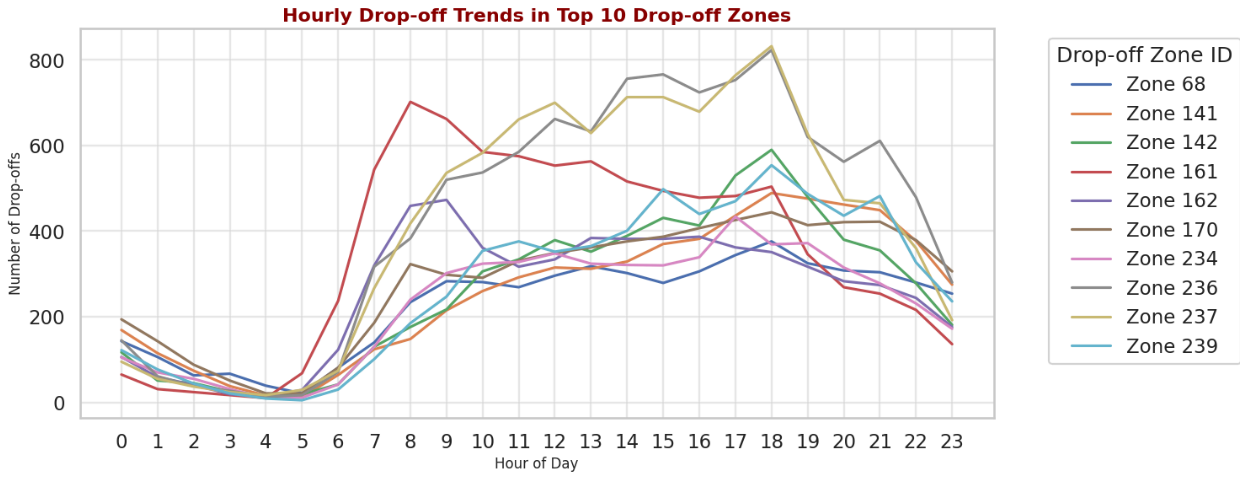
3.2.5 [3 marks]

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

```
# Find top 10 pickup and dropoff zones
# Find top 10 pickup zones overall
pickup_counts = mergedData.groupby("PULocationID").size().reset_index(name="pickup_count")
top10_pickup_zones = pickup_counts.sort_values(by="pickup_count", ascending=False).head(10)["PULocationID"]

# Find top 10 drop-off zones overall
dropoff_counts = mergedData.groupby("DOLocationID").size().reset_index(name="dropoff_count")
top10_dropoff_zones = dropoff_counts.sort_values(by="dropoff_count", ascending=False).head(10)["DOLocationID"]

# Filter dataset for only these top zones
pickup_trends = mergedData[mergedData["PULocationID"].isin(top10_pickup_zones)]
dropoff_trends = mergedData[mergedData["DOLocationID"].isin(top10_dropoff_zones)]

# Hourly pickup trends
pickup_hourly = pickup_trends.groupby(["pickup_hour_of_day", "PULocationID"]).size().reset_index(name="pickup_c
pickup_pivot = pickup_hourly.pivot(index="pickup_hour_of_day", columns="PULocationID", values="pickup_count").f

# Hourly drop-off trends
dropoff_hourly = dropoff_trends.groupby(["pickup_hour_of_day", "DOLocationID"]).size().reset_index(name="dropof
dropoff_pivot = dropoff_hourly.pivot(index="pickup_hour_of_day", columns="DOLocationID", values="dropoff_count"

# Plot pickups
plt.figure(figsize=(14,6))
for col in pickup_pivot.columns:
    plt.plot(pickup_pivot.index, pickup_pivot[col], label=f"Zone {col}")
plt.title("Hourly Pickup Trends in Top 10 Pickup Zones", fontsize=16, fontweight="bold", color="navy")
plt.xlabel("Hour of Day", fontsize=12)
plt.ylabel("Number of Pickups", fontsize=12)
plt.xticks(range(0,24))
plt.legend(title="Pickup Zone ID", bbox_to_anchor=(1.05, 1), loc="upper left")
plt.grid(alpha=0.4)
plt.show()
```

```
# Plot drop-offs
plt.figure(figsize=(14,6))
for col in dropoff_pivot.columns:
    plt.plot(dropoff_pivot.index, dropoff_pivot[col], label=f"Zone {col}")
plt.title("Hourly Drop-off Trends in Top 10 Drop-off Zones", fontsize=16, fontweight="bold", color="darkred")
plt.xlabel("Hour of Day", fontsize=12)
plt.ylabel("Number of Drop-offs", fontsize=12)
plt.xticks(range(0,24))
plt.legend(title="Drop-off Zone ID", bbox_to_anchor=(1.05, 1), loc="upper left")
plt.grid(alpha=0.4)
plt.show()
```





3.2.6 [3 marks]

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

```
# Find the top 10 and bottom 10 pickup/dropoff ratios
pickup_summary = mergedData.groupby('PULocationID').size().reset_index(name='pickup_count')
dropoff_summary = mergedData.groupby('DOLocationID').size().reset_index(name='dropoff_count')

# Merge pickup and dropoff counts on the location IDs
pickup_dropoff_summary = pd.merge(
    pickup_summary,
    dropoff_summary,
    left_on='PULocationID',
    right_on='DOLocationID',
    how='outer'  # Ensure all zones are included, even if one of the counts is missing
)

# Fill missing values with 0
pickup_dropoff_summary.fillna(0, inplace=True)

# Calculate the pickup-to-dropoff ratio
pickup_dropoff_summary['pickup_dropoff_ratio'] = (
```

```
    pickup_dropoff_summary['pickup_count'] / pickup_dropoff_summary['dropoff_count'].replace(0, float('inf'))
)

ratio_sorted = pickup_dropoff_summary.sort_values(by='pickup_dropoff_ratio', ascending=False)

top_10_ratios = ratio_sorted.head(10)
bottom_10_ratios = ratio_sorted.tail(10)

# Display results
print("Top 10 zones with the highest Pickup-to-Dropoff Ratios:")
print(top_10_ratios[['PULocationID', 'pickup_dropoff_ratio']])

print("\nBottom 10 zones with the lowest Pickup-to-Dropoff Ratios:")
print(bottom_10_ratios[['PULocationID', 'pickup_dropoff_ratio']])
```

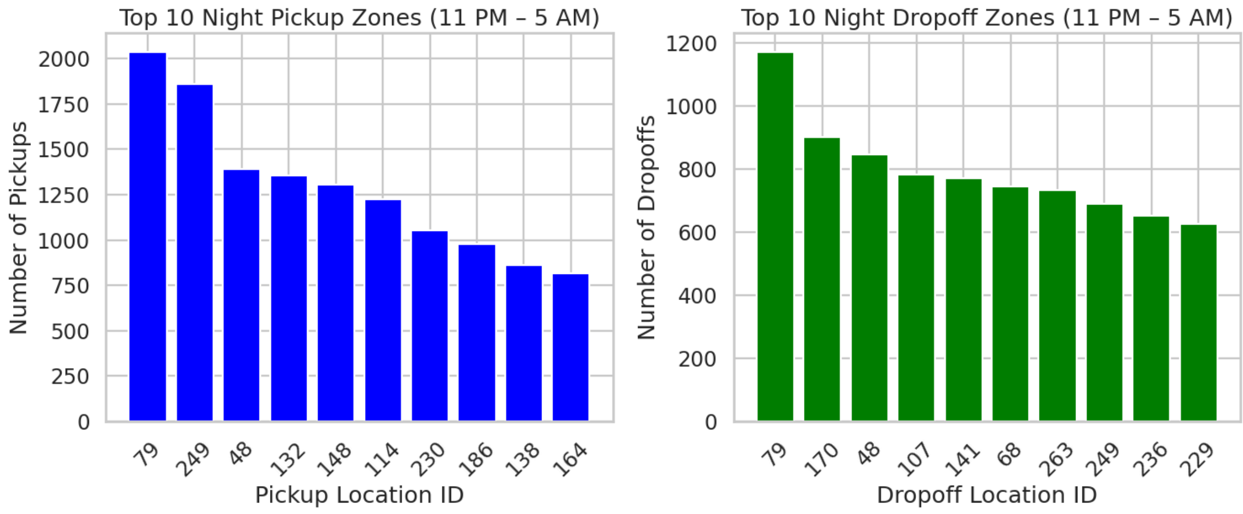
```
Top 10 zones with the highest Pickup-to-Dropoff Ratios:
     PULocationID  pickup_dropoff_ratio
65           70.0             10.391892
118         132.0              5.818565
124         138.0              2.870253
169         186.0              1.665556
87           93.0              1.500000
39           43.0              1.421655
101         114.0              1.398452
227         249.0              1.364810
147         162.0              1.312571
93          100.0              1.294436

Bottom 10 zones with the lowest Pickup-to-Dropoff Ratios:
     PULocationID  pickup_dropoff_ratio
187           0.0                   0.0
192           0.0                   0.0
182           0.0                   0.0
218           0.0                   0.0
223           0.0                   0.0
230           0.0                   0.0
229           0.0                   0.0
228           0.0                   0.0
241           0.0                   0.0
242           0.0                   0.0
```

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

mergedData['pickup_hour'] = mergedData['tpep_pickup_datetime'].dt.hour
mergedData['dropoff_hour'] = mergedData['tpep_dropoff_datetime'].dt.hour

# Filter trips between 11 PM (23) and 5 AM (inclusive)
night_trips = mergedData[
    ((mergedData['pickup_hour'] >= 23) | (mergedData['pickup_hour'] <= 5)) |
    ((mergedData['dropoff_hour'] >= 23) | (mergedData['dropoff_hour'] <= 5))
]

# Group separately for pickups and dropoffs
night_pickups = night_trips.groupby('PULocationID').size().reset_index(name='night_pickups')
night_dropoffs = night_trips.groupby('DOLocationID').size().reset_index(name='night_dropoffs')
top_10_night_pickups = night_pickups.sort_values(by='night_pickups', ascending=False).head(10)
top_10_night_dropoffs = night_dropoffs.sort_values(by='night_dropoffs', ascending=False).head(10)

plt.figure(figsize=(14, 6))

# Plot pickups
plt.subplot(1, 2, 1)
plt.bar(top_10_night_pickups['PULocationID'].astype(str), top_10_night_pickups['night_pickups'], color='blue')
plt.title('Top 10 Night Pickup Zones (11 PM - 5 AM)')
plt.xlabel('Pickup Location ID')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=45)

# Plot dropoffs
plt.subplot(1, 2, 2)
```

```
plt.bar(top_10_night_dropoffs['DOLocationID'].astype(str), top_10_night_dropoffs['night_dropoffs'], color='Gree
plt.title('Top 10 Night Dropoff Zones (11 PM – 5 AM)')
plt.xlabel('Dropoff Location ID')
plt.ylabel('Number of Dropoffs')
plt.xticks(rotation=45)

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

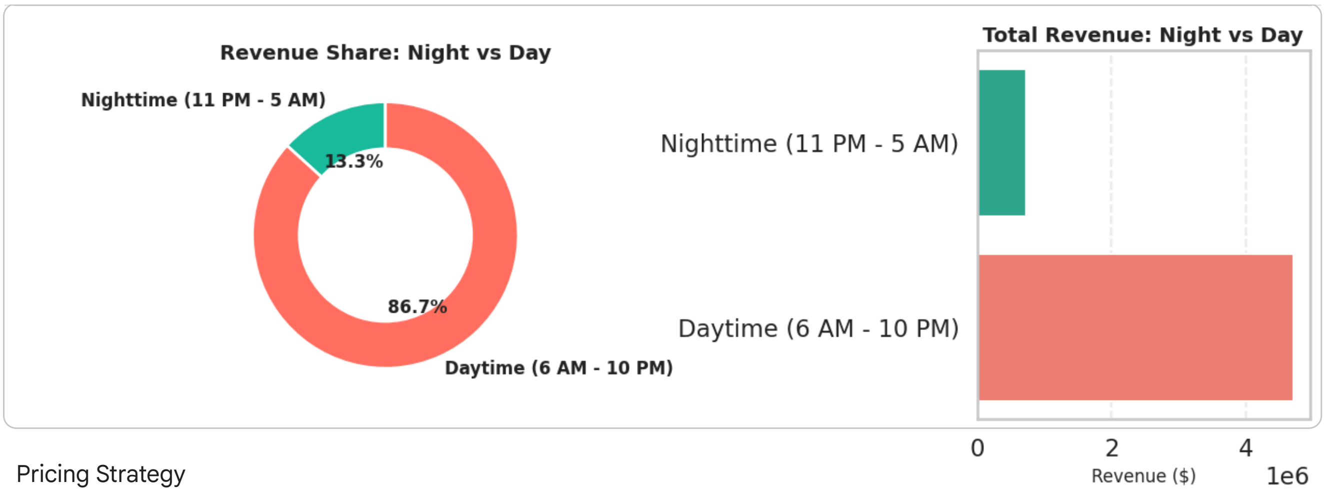


Now, let us find the revenue share for the night time hours and the day time hours. After this, we will move to deciding a pricing strategy.

3.2.8 [2 marks]

Find the revenue share for nighttime and daytime hours.

```
labels = ['Nighttime (11 PM - 5 AM)', 'Daytime (6 AM - 10 PM)']
values = [nighttimeRevenue, daytimeRevenue]
colors = ['#1ABC9C', '#FF6F61']  # Teal for night, coral for day

plt.figure(figsize=(12, 5))

# -------- Donut Pie Chart --------
plt.subplot(1, 2, 1)
wedges, texts, autotexts = plt.pie(
    values,
    labels=labels,
    autopct='%1.1f%%',
    startangle=90,
    colors=colors,
    wedgeprops={'edgecolor': 'white', 'linewidth': 2},
    textprops={'fontsize': 12, 'fontweight': 'bold'}
)
# Draw white circle for donut
centre_circle = plt.Circle((0,0),0.65,fc='white')
plt.gca().add_artist(centre_circle)
plt.title('Revenue Share: Night vs Day', fontsize=14, fontweight='bold')

# -------- Horizontal Bar Chart --------
plt.subplot(1, 2, 2)
sns.barplot(x=values, y=labels, palette=colors)
plt.title('Total Revenue: Night vs Day', fontsize=14, fontweight='bold')
plt.xlabel('Revenue ($)', fontsize=12)
plt.ylabel('')
plt.grid(axis='x', linestyle='--', alpha=0.3)

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



Pricing Strategy

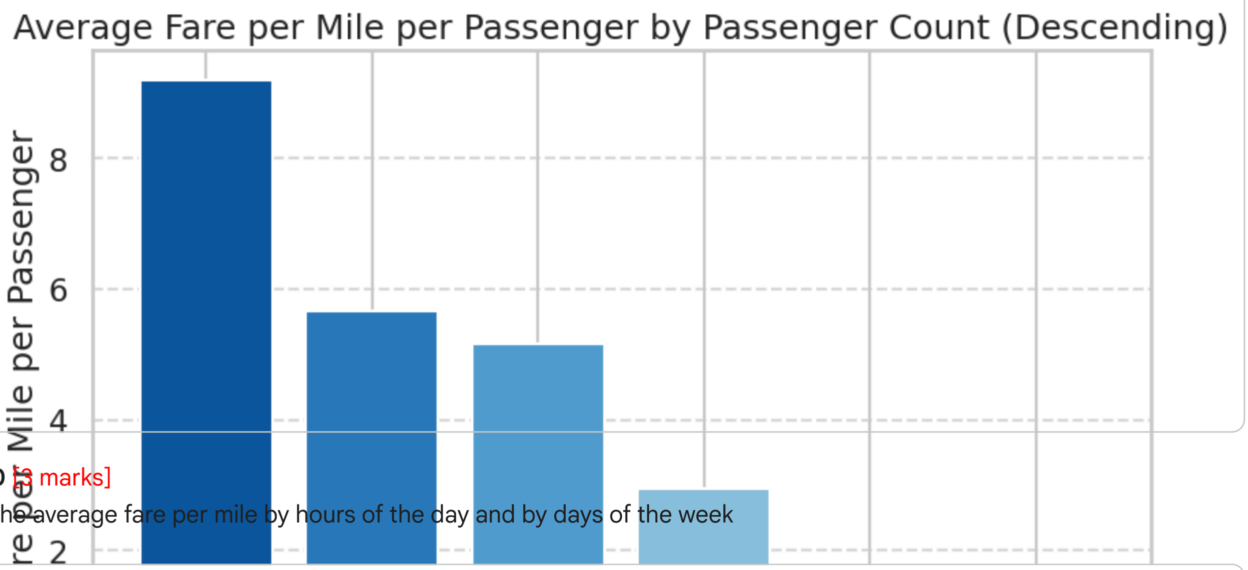
3.2.9 [2 marks]

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

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

```
# Analyse the fare per mile per passenger for different passenger counts
# Filter out trips with zero distance
mergedData = mergedData[mergedData['trip_distance'] > 0]

# Calculate the fare per mile for each trip
mergedData['fare_per_mile'] = mergedData['fare_amount'] / mergedData['trip_distance']

# Calculate fare per mile per passenger
mergedData['fare_per_mile_per_passenger'] = mergedData['fare_per_mile'] / mergedData['passenger_count']

# Group by passenger count and compute the average fare per mile per passenger
avg_fare_per_passenger = mergedData.groupby('passenger_count')['fare_per_mile_per_passenger'].mean().reset_inde

avg_fare_per_passenger_sorted = avg_fare_per_passenger.sort_values(by='fare_per_mile_per_passenger', ascending=

# Display the sorted average fare per mile per passenger
print("Average fare per mile per passenger by passenger count (descending order):")
print(avg_fare_per_passenger_sorted)

# Plot the results as a bar chart
plt.figure(figsize=(10,6))
palette = sns.color_palette("Blues_r", n_colors=len(avg_fare_per_passenger_sorted))
plt.bar(avg_fare_per_passenger_sorted['passenger_count'].astype(str),
        avg_fare_per_passenger_sorted['fare_per_mile_per_passenger'],
        color=palette)

plt.title('Average Fare per Mile per Passenger by Passenger Count (Descending)')
plt.xlabel('Passenger Count')
plt.ylabel('Fare per Mile per Passenger')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
Average fare per mile per passenger by passenger count (descending order):
   passenger_count  fare_per_mile_per_passenger
0              1.0                     9.197826
3              4.0                     5.658404
1              2.0                     5.158399
2              3.0                     2.957985
4              5.0                     1.529912
5              6.0                     1.339084
```

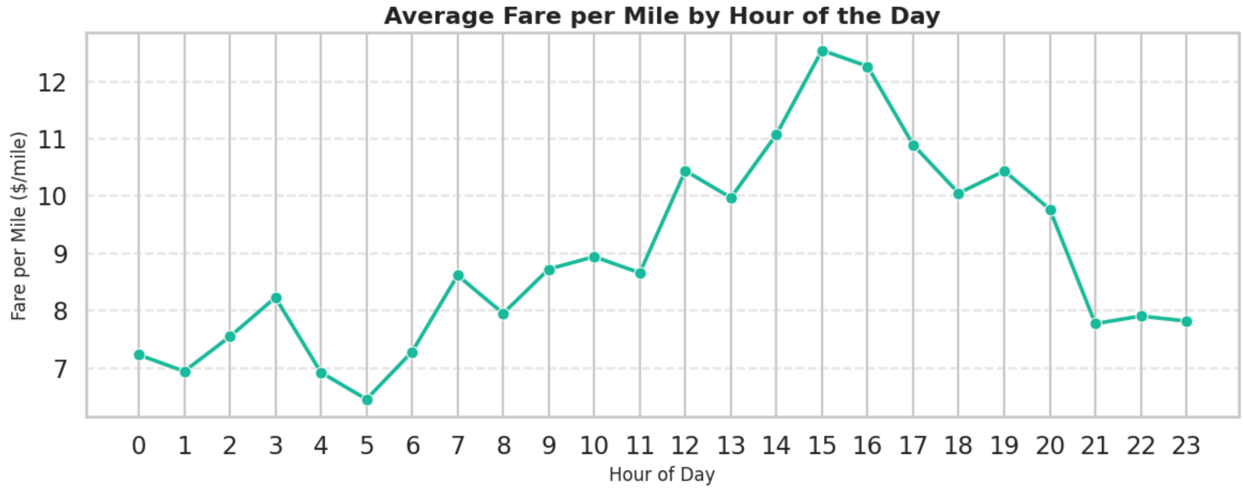


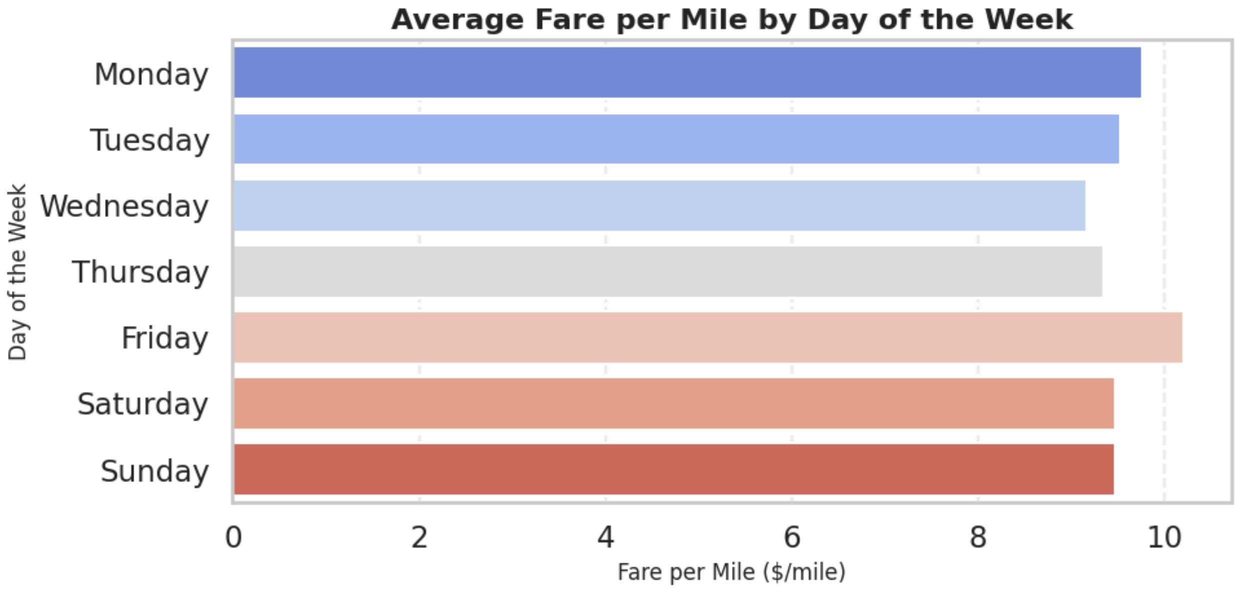
3.2.10 [3 marks]

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

```
# Plot: Average fare per mile by hour of the day
# Extract hour of the day and day of the week
mergedData['hour_of_day'] = mergedData['tpep_pickup_datetime'].dt.hour
mergedData['day_of_week'] = mergedData['tpep_pickup_datetime'].dt.dayofweek  # Monday=0, Sunday=6

# Calculate fare per mile
mergedData['fare_per_mile'] = mergedData['fare_amount'] / mergedData['trip_distance']

# Average fare per mile by hour of the day
avgFarePerMileHour = mergedData.groupby('hour_of_day')['fare_per_mile'].mean().reset_index()

# Average fare per mile by day of the week
avgFarePerMileDay = mergedData.groupby('day_of_week')['fare_per_mile'].mean().reset_index()

# Map day of the week to string names
avgFarePerMileDay['day_of_week'] = avgFarePerMileDay['day_of_week'].map({
    0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday',
    4: 'Friday', 5: 'Saturday', 6: 'Sunday'
})

# -----------------------------
# Clean plots
# -----------------------------

import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style("whitegrid")

# Plot: Average fare per mile by hour of the day
plt.figure(figsize=(12,5))
sns.lineplot(
    x='hour_of_day',
    y='fare_per_mile',
    data=avgFarePerMileHour,
    marker='o',
    markersize=8,
    linewidth=2.5,
    color='#1ABC9C'  # teal
)
plt.title('Average Fare per Mile by Hour of the Day', fontsize=16, fontweight='bold')
plt.xlabel('Hour of Day', fontsize=12)
plt.ylabel('Fare per Mile ($/mile)', fontsize=12)
plt.xticks(range(0,24))
plt.grid(axis='y', linestyle='--', alpha=0.4)
plt.tight_layout()
plt.show()
```

```
plt.show()

# Plot: Average fare per mile by day of the week (horizontal bar)
plt.figure(figsize=(10,5))
palette = sns.color_palette("coolwarm", n_colors=len(avgFarePerMileDay))
sns.barplot(
    y='day_of_week',  # horizontal
    x='fare_per_mile',
    data=avgFarePerMileDay,
    palette=palette
)
plt.title('Average Fare per Mile by Day of the Week', fontsize=16, fontweight='bold')
plt.ylabel('Day of the Week', fontsize=12)
plt.xlabel('Fare per Mile ($/mile)', fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.3)
plt.tight_layout()
plt.show()
```





3.2.11 [3 marks]

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

```
# Compare fare per mile for different vendors
# Calculate fare per mile
mergedData['fare_per_mile'] = mergedData['fare_amount'] / mergedData['trip_distance']

# Group by VendorID and hour_of_day to calculate average fare per mile
avgFarePerMileByVendorHour = mergedData.groupby(['VendorID', 'hour_of_day'])['fare_per_mile'].mean().reset_inde

# Pivot the table for visualization (VendorID as rows, hour_of_day as columns)
farePivotTable = avgFarePerMileByVendorHour.pivot(index='VendorID', columns='hour_of_day', values='fare_per_mil
```

```python
# Display the pivot table
print("Average Fare per Mile by Vendor and Hour of the Day:")
print(farePivotTable)

# Plot the results for each vendor
plt.figure(figsize=(12, 8))
colors = ['steelblue', 'darkorange']  # Different colors for each vendor
for i, vendorId in enumerate(farePivotTable.index):
    plt.plot(
        farePivotTable.columns,
        farePivotTable.loc[vendorId],
        marker='o',
        color=colors[i % len(colors)],
        label=f"Vendor {vendorId}"
    )

plt.title('Average Fare per Mile by Vendor and Hour of the Day', fontsize=14, fontweight='bold')
plt.xlabel('Hour of the Day', fontsize=12)
plt.ylabel('Average Fare per Mile ($)', fontsize=12)
plt.xticks(range(0, 24))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend(title='Vendor ID')
plt.tight_layout()
plt.show()
```

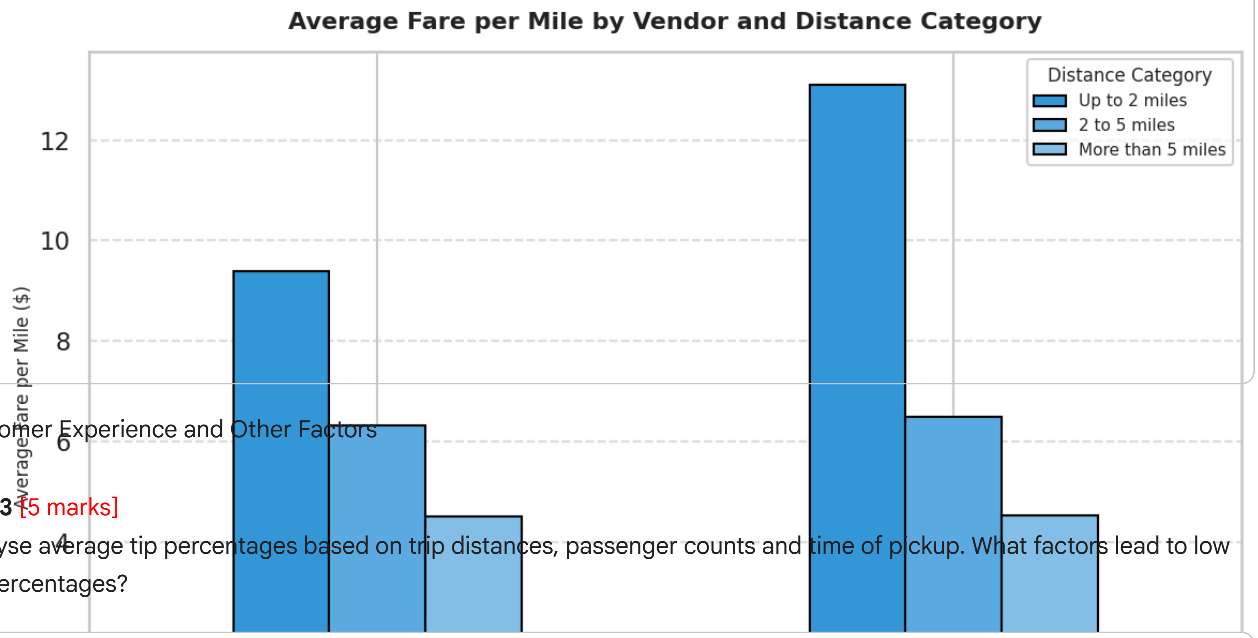
```
Average Fare per Mile by Vendor and Hour of the Day:
hour_of_day         0         1         2         3         4         5   \
VendorID
1            6.656528  6.609495  6.545370  6.515084  6.790786  6.694899
2            7.392130  7.034066  7.842532  8.734327  6.958702  6.360676

hour_of_day         6         7         8         9   ...        14   \
VendorID                                               ...
1            6.966081  7.663538  7.855709  8.104259   ...   8.709522
2            7.407113  9.236139  7.969013  8.904303   ...  11.925303
```

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.

```python
# Defining distance tiers
distance_labels = ['Up to 2 miles', '2 to 5 miles', 'More than 5 miles']

# Create a new column for distance category
mergedData['distance_category'] = pd.cut(
    mergedData['trip_distance'],
    bins=[0, 2, 5, float('inf')],
    labels=distance_labels,
    right=True
)

# Calculate average fare per mile by Vendor and distance category
avg_fare_per_mile_by_vendor = mergedData.groupby(['VendorID', 'distance_category'])['fare_per_mile'].mean().res
fare_pivot_table = avg_fare_per_mile_by_vendor.pivot(
    index='VendorID',
    columns='distance_category',
    values='fare_per_mile'
)

print("Average Fare per Mile by Vendor and Distance Category:")
print(fare_pivot_table)

# ---------- Plot ----------
plt.figure(figsize=(12, 8))
fare_pivot_table.plot(
    kind='bar',
    figsize=(12, 8),
    color=['#3498DB', '#5DADE2', '#85C1E9'],
    edgecolor='black'
)

plt.title(
    'Average Fare per Mile by Vendor and Distance Category',
    fontsize=16,
    fontweight='bold',
    pad=15
)
plt.xlabel('Vendor ID', fontsize=12)
plt.ylabel('Average Fare per Mile ($)', fontsize=12)
plt.xticks(rotation=0)
plt.legend(title='Distance Category', fontsize=11, title_fontsize=12)
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
Average Fare per Mile by Vendor and Distance Category:
distance_category  Up to 2 miles  2 to 5 miles  More than 5 miles
VendorID
1                       9.391686      6.317473           4.493202
2                      13.112045      6.493276           4.529479
<Figure size 1200x800 with 0 Axes>
```



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

# Calculate tip percentage
mergedData['tip_percentage'] = (mergedData['tip_amount'] / mergedData['fare_amount']) * 100

# Categorize trip distances
distance_labels = ['Up to 2 miles', '2 to 5 miles', 'More than 5 miles']
mergedData['distance_category'] = pd.cut(
    mergedData['trip_distance'],
    bins=[0, 2, 5, float('inf')],
    labels=distance_labels,
    right=True
)

# Average tip percentage by distance category
avg_tip_by_distance = mergedData.groupby('distance_category')['tip_percentage'].mean().reset_index()

# Average tip percentage by passenger count
avg_tip_by_passenger = mergedData.groupby('passenger_count')['tip_percentage'].mean().reset_index()

# Average tip percentage by hour of pickup
avg_tip_by_hour = mergedData.groupby('hour_of_day')['tip_percentage'].mean().reset_index()

import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")

# Plot: Tip percentage by distance
plt.figure(figsize=(10, 6))
sns.barplot(
    x='distance_category',
    y='tip_percentage',
    data=avg_tip_by_distance,
    palette='pastel',
    edgecolor='black'
)
plt.title('Average Tip Percentage by Trip Distance', fontsize=14, fontweight='bold')
plt.xlabel('Trip Distance Category')
plt.ylabel('Average Tip Percentage')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Plot: Tip percentage by passenger
plt.figure(figsize=(10, 6))
sns.lineplot(
```

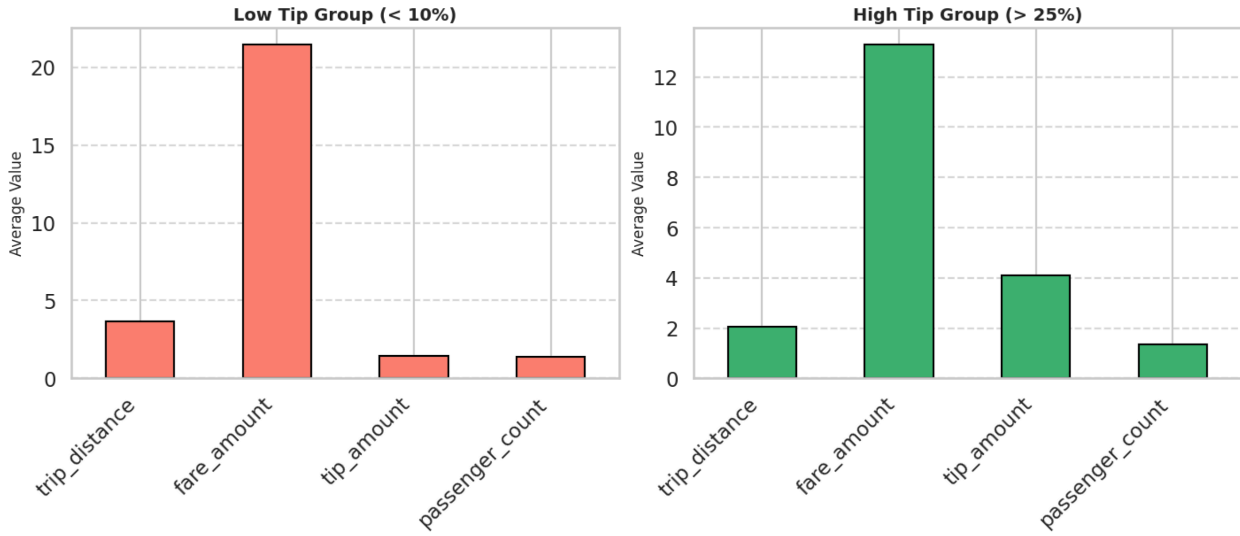
```
    x='passenger_count',
    y='tip_percentage',
    data=avg_tip_by_passenger,
    marker='o',
    linewidth=2.5,
    markersize=8,
    color='#1ABC9C'
)
plt.title('Average Tip Percentage by Passenger Count', fontsize=14, fontweight='bold')
plt.xlabel('Passenger Count')
plt.ylabel('Average Tip Percentage')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Plot: Tip percentage by hour of day
plt.figure(figsize=(10, 6))
sns.lineplot(
    x='hour_of_day',
    y='tip_percentage',
    data=avg_tip_by_hour,
    marker='o',
    linewidth=2.5,
    markersize=8,
    color='#FF6F61'
)
plt.title('Average Tip Percentage by Hour of Pickup', fontsize=14, fontweight='bold')
plt.xlabel('Hour of the Day')
plt.ylabel('Average Tip Percentage')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Correlation
tip_correlation = mergedData[['trip_distance', 'passenger_count', 'hour_of_day', 'tip_percentage']].corr()
print("Correlation matrix for factors affecting tip percentage:")
print(tip_correlation)
```

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%

# Compare trips with tip percentage < 10% to trips with tip percentage > 25%
mergedData['tip_percentage'] = (mergedData['tip_amount'] / mergedData['fare_amount']) * 100

# Filter trips based on tip percentage
low_tip_df = mergedData[mergedData['tip_percentage'] < 10]
high_tip_df = mergedData[mergedData['tip_percentage'] > 25]

# Compute average statistics for both groups
low_tip_stats = low_tip_df[['trip_distance', 'fare_amount', 'tip_amount', 'passenger_count']].mean()
high_tip_stats = high_tip_df[['trip_distance', 'fare_amount', 'tip_amount', 'passenger_count']].mean()

# Display statistics for comparison
print("Low Tip Group (tip percentage < 10%)")
print(low_tip_stats.to_frame().T)
print("\nHigh Tip Group (tip percentage > 25%)")
print(high_tip_stats.to_frame().T)
```

```
print(high_tip_stats.to_frame().T)

# Visualize comparison
fig, axes = plt.subplots(1, 2, figsize=(14, 6), constrained_layout=True)

# Low tip group plot
low_tip_stats.plot(kind='bar', ax=axes[0], color='salmon', edgecolor='black')
axes[0].set_title('Low Tip Group (< 10%)', fontsize=14, fontweight='bold')
axes[0].set_ylabel('Average Value', fontsize=12)
axes[0].set_xticklabels(low_tip_stats.index, rotation=45, ha='right')
axes[0].grid(axis='y', linestyle='--', alpha=0.7)

# High tip group plot
high_tip_stats.plot(kind='bar', ax=axes[1], color='mediumseagreen', edgecolor='black')
axes[1].set_title('High Tip Group (> 25%)', fontsize=14, fontweight='bold')
axes[1].set_ylabel('Average Value', fontsize=12)
axes[1].set_xticklabels(high_tip_stats.index, rotation=45, ha='right')
axes[1].grid(axis='y', linestyle='--', alpha=0.7)

plt.show()
```

```
Low Tip Group (tip percentage < 10%)
   trip_distance  fare_amount  tip_amount  passenger_count
0       3.687293    21.492958    1.42122         1.369158

High Tip Group (tip percentage > 25%)
   trip_distance  fare_amount  tip_amount  passenger_count
0       2.037586    13.308439    4.109486         1.364281
```

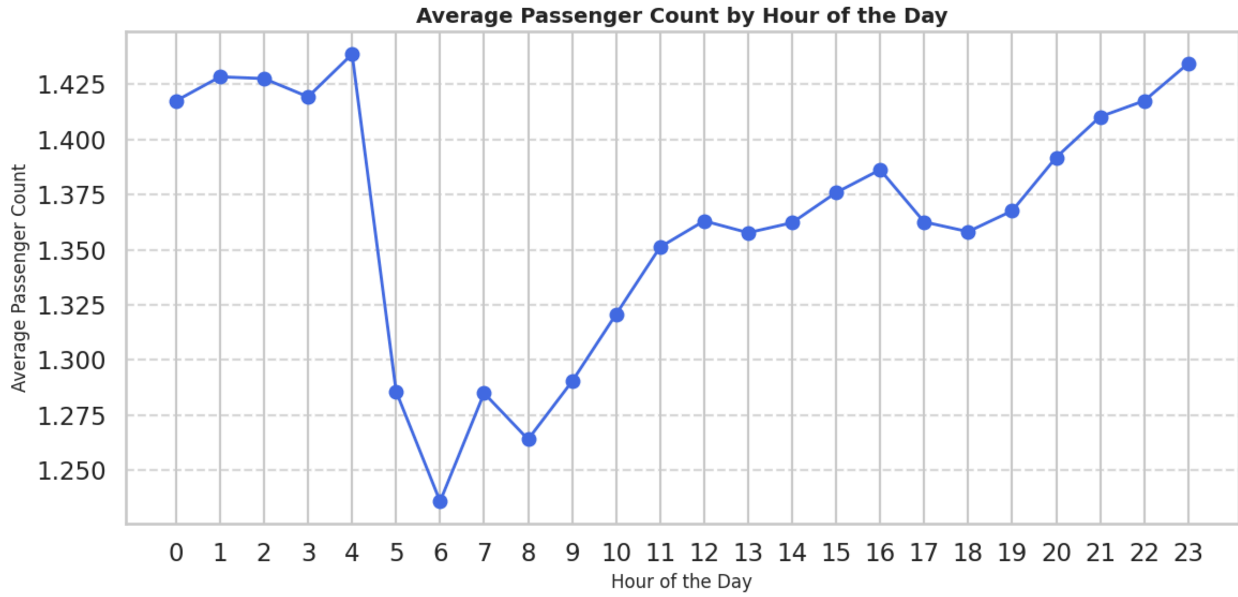


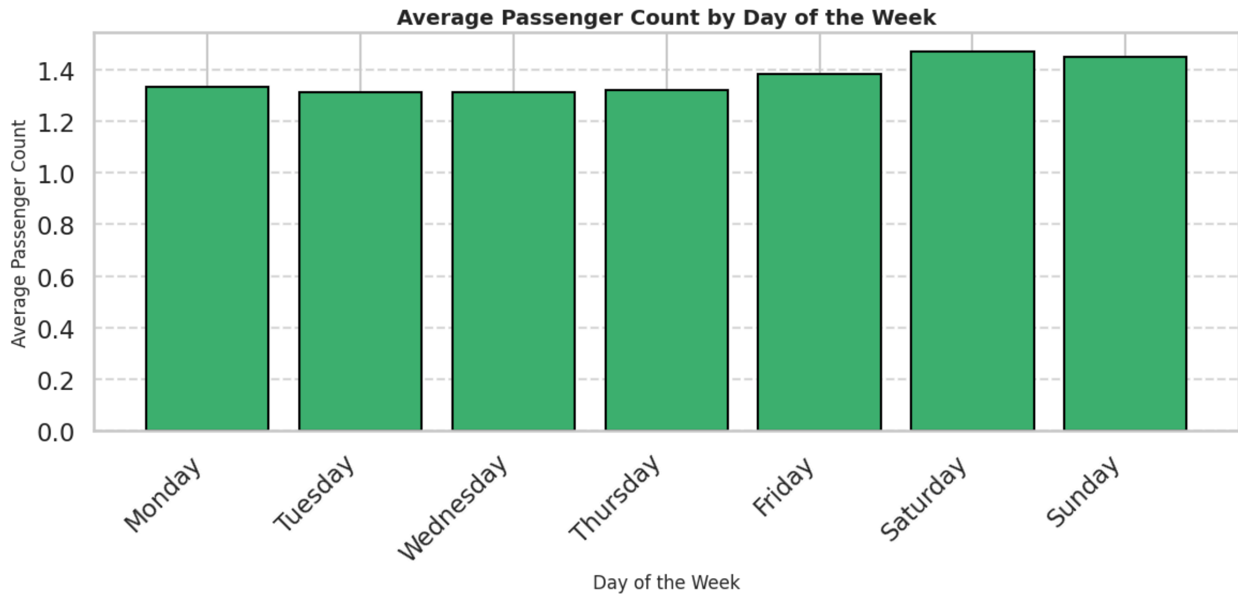
3.2.14 [3 marks]

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

```
# See how passenger count varies across hours and days
# Average passenger count by hour of the day
avg_passengers_hour = mergedData.groupby('hour_of_day')['passenger_count'].mean().reset_index()

# Average passenger count by day of the week
avg_passengers_day = mergedData.groupby('day_of_week')['passenger_count'].mean().reset_index()
avg_passengers_day = avg_passengers_day.sort_values('day_of_week')
avg_passengers_day['day_of_week'] = avg_passengers_day['day_of_week'].map({
    0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday',
    4: 'Friday', 5: 'Saturday', 6: 'Sunday'
})

plt.figure(figsize=(12, 6))
plt.plot(
    avg_passengers_hour['hour_of_day'],
    avg_passengers_hour['passenger_count'],
    marker='o', color='royalblue', linewidth=2
)
plt.title('Average Passenger Count by Hour of the Day', fontsize=14, fontweight='bold')
plt.xlabel('Hour of the Day', fontsize=12)
plt.ylabel('Average Passenger Count', fontsize=12)
plt.xticks(range(0, 24, 1))
```

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

plt.figure(figsize=(12, 6))
plt.bar(
    avg_passengers_day['day_of_week'],
    avg_passengers_day['passenger_count'],
    color='mediumseagreen', edgecolor='black'
)
plt.title('Average Passenger Count by Day of the Week', fontsize=14, fontweight='bold')
plt.xlabel('Day of the Week', fontsize=12)
plt.ylabel('Average Passenger Count', fontsize=12)
plt.xticks(rotation=45, ha='right')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```





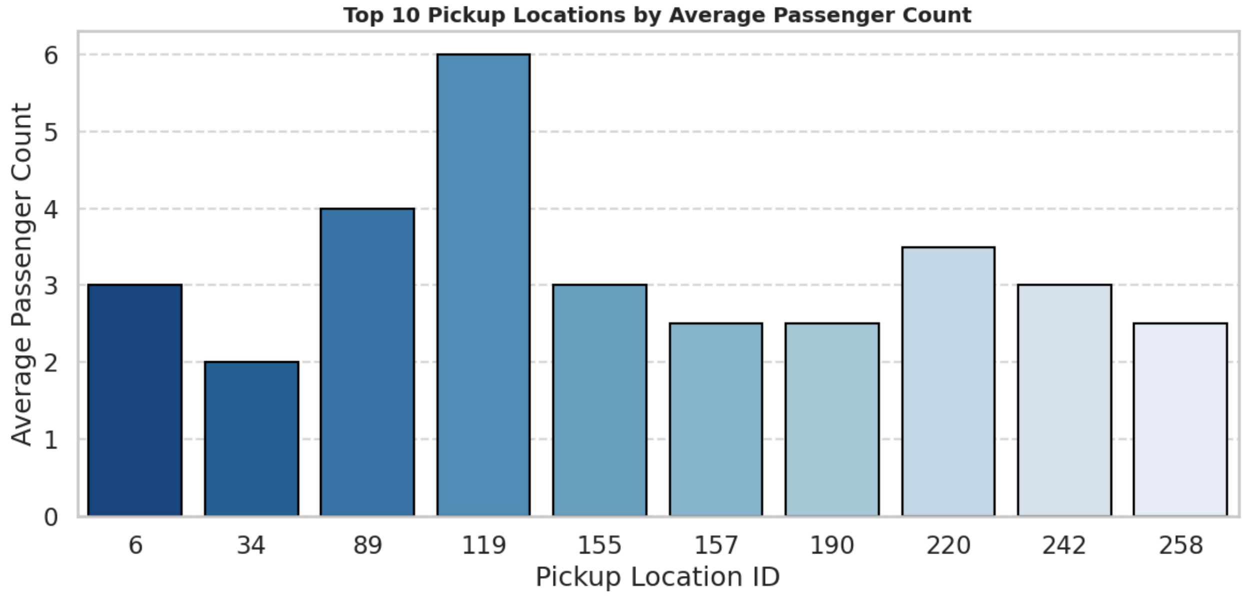
3.2.15 [2 marks]

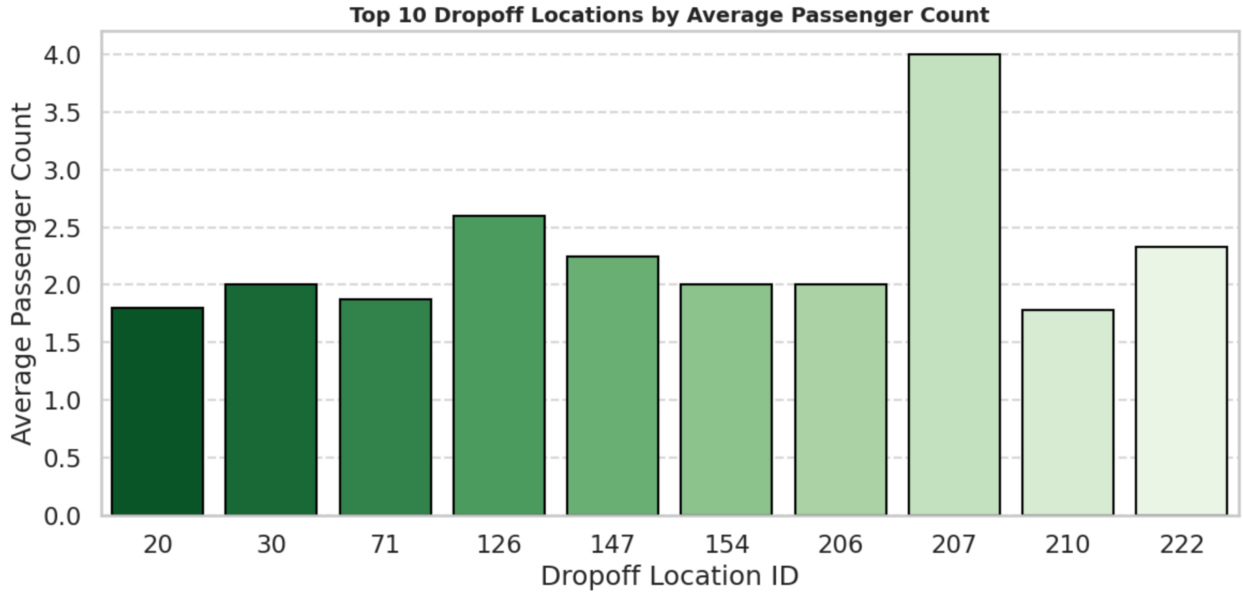
Analyse the variation of passenger counts across zones

```
# How does passenger count vary across zones

# Average passenger count by pickup location
```

```python
avgPassengerCountPickup = mergedData.groupby('PULocationID')['passenger_count'].mean().reset_index()
avgPassengerCountPickup = avgPassengerCountPickup.sort_values(by='passenger_count', ascending=False)

# Average passenger count by dropoff location
avgPassengerCountDropoff = mergedData.groupby('DOLocationID')['passenger_count'].mean().reset_index()
avgPassengerCountDropoff = avgPassengerCountDropoff.sort_values(by='passenger_count', ascending=False)

# Display top 10 pickup locations
print("Top 10 Pickup Locations by Average Passenger Count")
print(avgPassengerCountPickup.head(10))

# Display top 10 dropoff locations
print("\nTop 10 Dropoff Locations by Average Passenger Count")
print(avgPassengerCountDropoff.head(10))


# Top 10 pickup locations
sns.set_style("whitegrid")
plt.figure(figsize=(12,6))
sns.barplot(
    x='PULocationID',
    y='passenger_count',
    data=avgPassengerCountPickup.head(10),
    palette='Blues_r',
    edgecolor='black'
)
plt.title('Top 10 Pickup Locations by Average Passenger Count', fontsize=14, fontweight='bold')
plt.xlabel('Pickup Location ID')
plt.ylabel('Average Passenger Count')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()

# Top 10 dropoff locations
plt.figure(figsize=(12,6))
sns.barplot(
    x='DOLocationID',
    y='passenger_count',
    data=avgPassengerCountDropoff.head(10),
    palette='Greens_r',
    edgecolor='black'
)
plt.title('Top 10 Dropoff Locations by Average Passenger Count', fontsize=14, fontweight='bold')
plt.xlabel('Dropoff Location ID')
plt.ylabel('Average Passenger Count')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
Top 10 Pickup Locations by Average Passenger Count
     PULocationID  passenger_count
72            119              6.0
56             89              4.0
136           220              3.5
97            155              3.0
2               6              3.0
154           242              3.0
98            157              2.5
116           190              2.5
164           258              2.5
19             34              2.0

Top 10 Dropoff Locations by Average Passenger Count
     DOLocationID  passenger_count
186           207         4.000000
111           126         2.600000
199           222         2.333333
132           147         2.250000
139           154         2.000000
185           206         2.000000
25             30         2.000000
65             71         1.875000
16             20         1.800000
189           210         1.785714
```

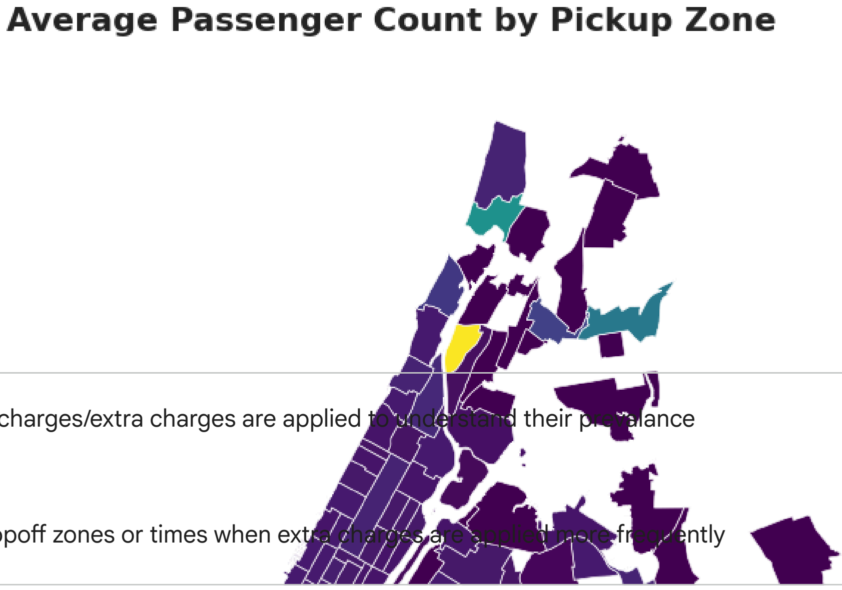





```
# For a more detailed analysis, we can use the zones_with_trips GeoDataFrame
# Create a new column for the average passenger count in each zone.

avg_passenger_pickup = mergedData.groupby('PULocationID')['passenger_count'].mean().reset_index()
avg_passenger_pickup.rename(columns={'PULocationID': 'LocationID', 'passenger_count': 'avg_passenger_pickup'},
```

```
avg_passenger_dropoff = mergedData.groupby('DOLocationID')['passenger_count'].mean().reset_index()
avg_passenger_dropoff.rename(columns={'DOLocationID': 'LocationID', 'passenger_count': 'avg_passenger_dropoff'}

zones_with_trips = zones.copy()
zones_with_trips = zones_with_trips.merge(avg_passenger_pickup, on='LocationID', how='left')
zones_with_trips = zones_with_trips.merge(avg_passenger_dropoff, on='LocationID', how='left')

print(zones_with_trips[['zone', 'borough', 'avg_passenger_pickup', 'avg_passenger_dropoff']].head(10))

# Plotting the pickup zones with average passenger count
fig, ax = plt.subplots(1, 1, figsize=(12, 10))

# Use a more modern, clean colormap
zones_with_trips.plot(
    column='avg_passenger_pickup',
    ax=ax,
    legend=True,
    cmap='viridis',  # visually appealing and clear
    edgecolor='white',  # clean boundaries
    linewidth=0.5,
    legend_kwds={
        'label': "Avg Passenger Count (Pickup)",
        'orientation': "horizontal",
        'shrink': 0.6,
        'pad': 0.02
    }
)

# Clean up axes
ax.set_axis_off()

# Title
ax.set_title("Average Passenger Count by Pickup Zone", fontsize=16, fontweight='bold', pad=20)

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

```
                    zone        borough  avg_passenger_pickup  \
0         Newark Airport            EWR              1.250000
1            Jamaica Bay         Queens                   NaN
2  Allerton/Pelham Gardens        Bronx                   NaN
3          Alphabet City      Manhattan              1.454545
4          Arden Heights  Staten Island                   NaN
5  Arrochar/Fort Wadsworth  Staten Island            3.000000
6                Astoria         Queens              1.348837
7           Astoria Park         Queens              1.000000
8             Auburndale         Queens                   NaN
9           Baisley Park         Queens              1.631579

   avg_passenger_dropoff
0               1.708589
1                    NaN
2               1.000000
3               1.348724
4                    NaN
5                    NaN
6               1.325103
7               1.000000
8               1.000000
9               1.485714
```



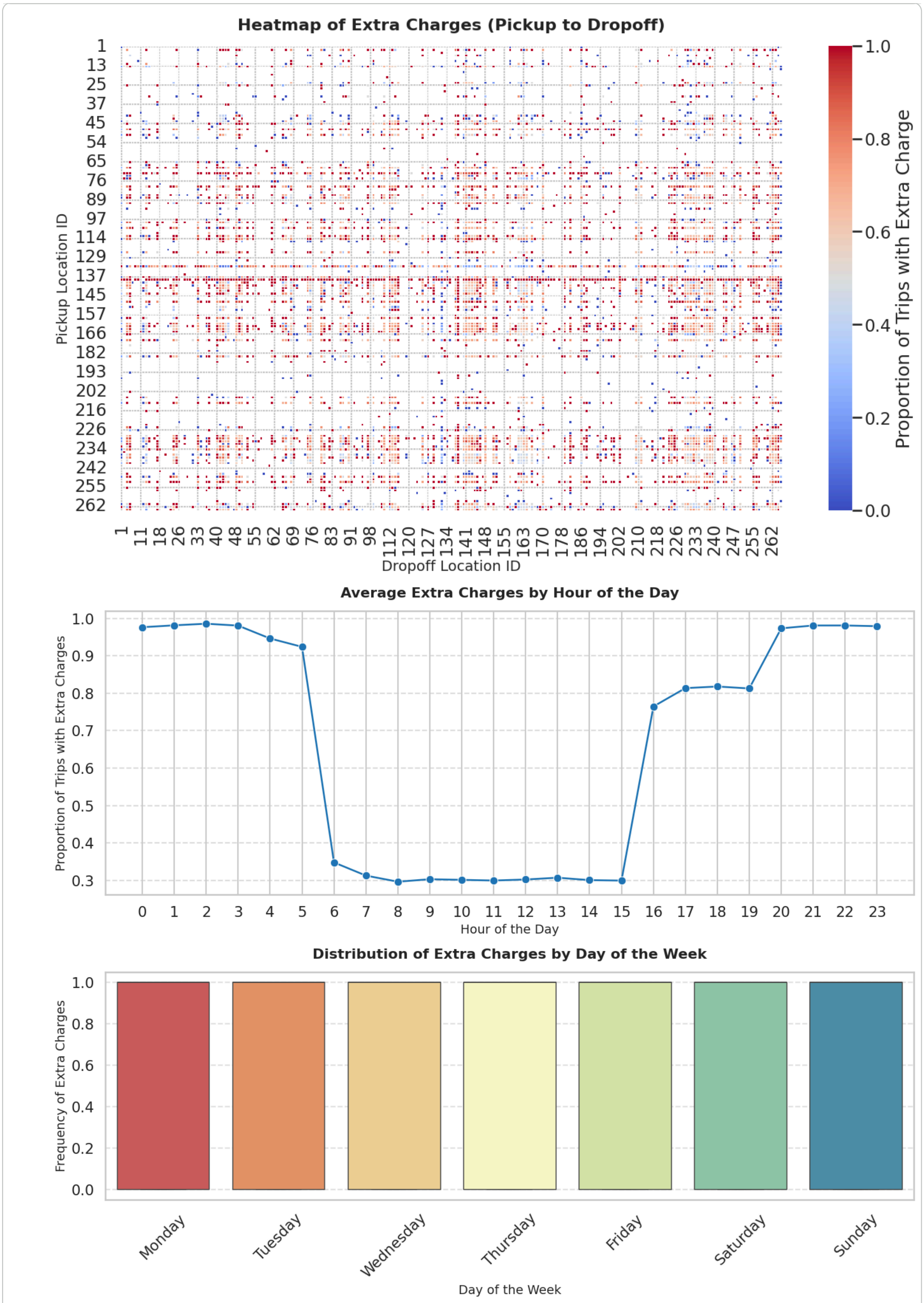
Find out how often surcharges/extra charges are applied to understand their prevalance

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?
# ---------- Flag rows where extra charge was applied ----------
mergedData['extraChargeApplied'] = (mergedData['extra'] > 0)

# ---------- Heatmap: Pickup to Dropoff ----------
plt.figure(figsize=(14, 8))
extraChargeHeatmapData = mergedData.pivot_table(
    values='extraChargeApplied',
    index='PULocationID',
    columns='DOLocationID',
    aggfunc='mean'
)
sns.heatmap(
    extraChargeHeatmapData,
    cmap='coolwarm',   # smoother, visually appealing
    linewidths=0.5,
    cbar_kws={'label': 'Proportion of Trips with Extra Charge'},
    square=True,
    robust=True
)
plt.title("Heatmap of Extra Charges (Pickup to Dropoff)", fontsize=16, fontweight='bold', pad=15)
plt.xlabel("Dropoff Location ID", fontsize=14)
plt.ylabel("Pickup Location ID", fontsize=14)
plt.tight_layout()
plt.show()

# ---------- Average Extra Charges by Hour ----------
plt.figure(figsize=(14, 6))
avg_extra = mergedData.groupby('hour_of_day')['extraChargeApplied'].mean().reset_index()
sns.lineplot(
    x='hour_of_day',
```

```python
    y='extraChargeApplied',
    data=avg_extra,
    marker='o',
    color='#1f77b4',  # clean, professional blue
    linewidth=2
)
plt.title('Average Extra Charges by Hour of the Day', fontsize=16, fontweight='bold', pad=15)
plt.xlabel('Hour of the Day', fontsize=14)
plt.ylabel('Proportion of Trips with Extra Charges', fontsize=14)
plt.xticks(range(0, 24))
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()

# ---------- Distribution of Extra Charges by Day of the Week ----------
mergedData['dayOfWeek'] = mergedData['tpep_pickup_datetime'].dt.strftime('%A')
daysOfWeekOrder = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
mergedData['dayOfWeek'] = pd.Categorical(mergedData['dayOfWeek'], categories=daysOfWeekOrder, ordered=True)

plt.figure(figsize=(14, 6))
sns.boxenplot(  # changed from boxplot → boxenplot for better distribution visibility
    x='dayOfWeek',
    y='extraChargeApplied',
    data=mergedData,
    palette='Spectral'  # colorful but readable
)
plt.title('Distribution of Extra Charges by Day of the Week', fontsize=16, fontweight='bold', pad=15)
plt.xlabel('Day of the Week', fontsize=14)
plt.ylabel('Frequency of Extra Charges', fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



4 Conclusion