New York City Yellow Taxi Data

Objective

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

Problem Statement

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

Tasks

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

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

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

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

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

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

Suppose the marks for two tasks are: 3 marks for 2.1.1 and 2 marks for 3.2.2, or then, you will earn 3 marks for completing task 2.1.1 and 2 marks for completing task 3.2.2.

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

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.

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

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

Taxi Zones

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

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

This is covered in more detail in later sections.

---

1 Data Preparation

[5 marks]

Import Libraries

In [1]:
```
# Import warnings
import warnings
warnings.filterwarnings("ignore")
print("Sucessfully imported warnings")
```

In [2]:
```
# 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

print("Sucessfully inmported all required libraries")
```

In [3]:
```
# Recommended versions
# numpy version: 1.26.4
# pandas version: 2.2.2
# matplotlib version: 3.10.0
# seaborn version: 0.13.2


print("Im facing some issue with older version with python modules hence installed

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

```
### For installing specific version of modules below code we can use
### pip install numpy == <required version>
```

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')`

In [4]:
```
# Try loading one file

# df = pd.read_parquet('2023-1.parquet')
# df.info()
```

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.

In [5]:
```
# Sample the data
# It is recommmended to not load all the files at once to avoid memory overload
```

In [6]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

In [7]:
```
# 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 -> m
# 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/Assignments/EDA/data_NYC_Taxi/trip_records')

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

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


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

#         # Reading the current file



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

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

    #         # Iterate through each hour of the selected date
```

```
    #             # Sample 5% of the hourly data randomly

    #             # add data of this hour to the dataframe

    #     # Concatenate the sampled data of all the dates to a single dataframe
    #     df = # we initialised this empty DF earlier

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

In [8]:

```
import os
! pip install pyarrow
data_dir = "..\\Datasets_Dictionary\\trip_records\\"

# List all Parquet files
file_list = [f for f in os.listdir(data_dir) if f.endswith(".parquet")]
print(file_list)

# empty dataframe
final_sampled_data = pd.DataFrame()
sampling_fraction = 0.0075 # Fraction Size

# # Looping (each month's data)
for file_name in file_list:
    try:
        file_path = os.path.join(data_dir, file_name)
        df = pd.read_parquet(file_path, engine="pyarrow")

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

        monthly_sampled_data = pd.DataFrame()

        # Loop through each unique date in the month
        for date in df["pickup_date"].unique():
            daily_data = df[df["pickup_date"] == date]

            # Loop through each hour
            for hour in range(24):
                hour_data = daily_data[daily_data["pickup_hour"] == hour]

                # Sample smaller fraction per hour
                if len(hour_data) > 0:
                    sampled_hour_data = hour_data.sample(frac=sampling_fraction, ra
                    monthly_sampled_data = pd.concat([monthly_sampled_data, sampled

        final_sampled_data = pd.concat([final_sampled_data, monthly_sampled_data])

        print(f"Processed: {file_name}")

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

# Reset index
final_sampled_data.reset_index(drop=True, inplace=True)
```

```
# Display new sampled data size
print("Optimized Sampled Data Shape:", final_sampled_data.shape)

final_sampled_data.shape
```

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.

In [9]:
```
# Store the df in csv/parquet
# df.to_parquet('')

# Saving the final sampled file
final_sampled_data.to_parquet('optimized_sampled_nyc_taxi.parquet')
final_sampled_data.to_csv('optimized_sampled_nyc_taxi.csv', index=False)
## final_sampled_data.to_excel('optimized_sampled_nyc_taxi.xlsx', index=False, engi

print("✅ Data saved successfully in Parquet, CSV, and Excel formats.")

final_sampled_data.shape
```

2 Data Cleaning

[30 marks]

Now we can load the new data directly.

In [10]:
```
# Load the new data file

csv_path = "optimized_sampled_nyc_taxi.csv"
df = pd.read_csv(csv_path)
```

In [11]:
```
# df.head()
df.head(5)
```

In [12]:
```
# df.info()
df.info()
```

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

In [13]:
```
# Fix the index and drop any columns that are not needed
```

Fixing Columns

My Observations after head() and info() function

- **No extra index column** → No need to reset the index,as i alredy use reset index.
- **Duplicate column found**: `airport_fee` and `Airport_fee`.
- **Datetime columns (`tpep_pickup_datetime`, `tpep_dropoff_datetime`) were stored as `object` (string).**
  - Converted them to `datetime` for proper analysis.
- **Missing values found in:**
  - `passenger_count`, `RatecodeID`, `store_and_fwd_flag`, `congestion_surcharge`.

- **Fixing datetime formats** allows for **time-based trend analysis (hourly, daily, monthly insights)**.
- **No unnecessary columns removed yet** to avoid losing valuable data for future analysis.

In [14]:
```
# Converting column datatype
df["tpep_pickup_datetime"] = pd.to_datetime(df["tpep_pickup_datetime"])
df["tpep_dropoff_datetime"] = pd.to_datetime(df["tpep_dropoff_datetime"])

df.info()
```

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.

In [15]:
```
# Combine the two airport fee columns

# filling missing values in 'Airport_fee'
df["Airport_fee"] = df["Airport_fee"].fillna(df["airport_fee"])

# Drop
df.drop(columns=["airport_fee"], inplace=True)
df[["Airport_fee"]].info()
```

2.1.3 [5 marks]

Fix columns with negative (monetary) values

In [16]:
```
# check where values of fare amount are negative

## "fare_amount", "total_amount", "tip_amount", "tolls_amount", "Airport_fee"
```

```
# Find negative values in these columns

df[df['fare_amount'] < 0 ].count()
df[df['total_amount'] < 0 ].count()
df[df['tip_amount'] < 0 ].count()
df[df['tolls_amount'] < 0 ].count()
df[df['Airport_fee'] < 0 ].count()


# Display rows where fare_amount is negative
df[df["fare_amount"] < 0]
```

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

In [17]:
```
# Analyse RatecodeID for the negative fare amounts

# Filter rows where fare_amount is negative
negative_fare_data = df[df["fare_amount"] < 0]

# Check unique RatecodeID values in these rows
ratecode_negative_fare_counts = negative_fare_data["RatecodeID"].value_counts()

# Display results
print("RatecodeID distribution for negative fare amounts:")
print(ratecode_negative_fare_counts)

# Display sample records to observe patterns
negative_fare_data[["fare_amount", "RatecodeID", "total_amount", "payment_type"]].h
```

Analysis of RatecodeID for Negative Fare Amounts

Insights

- **Our dataset does not contain negative fare values**, so no corrections are needed.
- This ensures that **fare-related calculations are already clean**.

In [18]:
```
# Find which columns have negative values

# Select only numerical columns
num_cols = df.select_dtypes(include=['number']).columns
print(num_cols)
# Find negative values in each column
negative_counts = (df[num_cols] < 0).sum()

# Display columns with negative values
negative_counts[negative_counts > 0]
```

In [19]:
```
# fix these negative values

# List  replaced with 0
fix_to_zero = ["extra", "mta_tax", "improvement_surcharge", "congestion_surcharge",
```

```
# Replace negative values with 0 in these columns
df[fix_to_zero] = df[fix_to_zero].applymap(lambda x: max(x, 0))

#  Set  NaN
df["total_amount"] = df["total_amount"].apply(lambda x: x if x >= 0 else None)

# Verify fixes
df[["extra", "mta_tax", "improvement_surcharge", "congestion_surcharge", "total_amo
```

Fixing Negative Monetary Values

Observations

- Negative were found in **fees and surcharge-related columns**:
  - `extra`, `mta_tax`, `improvement_surcharge`, `congestion_surcharge`, `Airport_fee`
  - **These values should never be negative** since they represent charges.
- **Negative `total_amount` requires further investigation** as it includes multiple components.

Fixes Applied

- **Replaced negative values with `0`** in:
  - `extra`, `mta_tax`, `improvement_surcharge`, `congestion_surcharge`, `Airport_fee`
- **Set negative `total_amount` to `NaN`** for now to analyze further.

2.2 Handling Missing Values

[10 marks]

2.2.1 [2 marks]

Find the proportion of missing values in each column

In [20]:
```
# Find the proportion of missing values in each column

# Calculate the percentage
missing_values = df.isnull().sum() / len(df) * 100

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

2.2.2 [3 marks]

Handling missing values in `passenger_count`

In [21]:
```
# Display the rows with null values
# Impute NaN values in 'passenger_count'


df[df["passenger_count"].isnull()].head()

# Impute missing values with the mode
most_common_passenger_count = df["passenger_count"].mode()[0]
df["passenger_count"].fillna(most_common_passenger_count, inplace=True)
```

Did you find zeroes in passenger_count? Handle these.

In [22]:
```
# Count rows where passenger_count is 0
zero_passenger_count = (df["passenger_count"] == 0).sum()
print(f"Number of trips with zero passengers: {zero_passenger_count}")

# Handle zero passenger counts (Replace with mode, which is 1)
df.loc[df["passenger_count"] == 0, "passenger_count"] = df["passenger_count"].mode(
```

2.2.3 [2 marks]

Handle missing values in `RatecodeID`

In [23]:
```
# Fix missing values in 'RatecodeID'

# Check distribution of RatecodeID
ratecode_mode = df["RatecodeID"].mode()[0]  # Get most common RatecodeID
print(f"Most common RatecodeID: {ratecode_mode}")

# Fill missing values with the most common RatecodeID
df["RatecodeID"].fillna(ratecode_mode, inplace=True)

# Verify if missing values are fixed
df["RatecodeID"].isnull().sum()
```

2.2.4 [3 marks]

Impute NaN in `congestion_surcharge`

In [24]:
```
# handle null values in congestion_surcharge

# Check distribution of congestion_surcharge
print(df["congestion_surcharge"].value_counts())

# Get the most common congestion surcharge value
congestion_mode = df["congestion_surcharge"].mode()[0]
print(f"Most common congestion surcharge: {congestion_mode}")

# Fill missing values with the most common value
df["congestion_surcharge"].fillna(congestion_mode, inplace=True)

# Verify if missing values are fixed
df["congestion_surcharge"].isnull().sum()
```

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

In [25]:
```
# Handle any remaining missing values

# Check for remaining missing values
remaining_missing_values = df.isnull().sum()
print("Remaining missing values:\n", remaining_missing_values[remaining_missing_val

# Handle 'store_and_fwd_flag'
df["store_and_fwd_flag"].fillna("N", inplace=True)

# Handle 'total_amount'
df["total_amount"].fillna(0, inplace=True)

# Handle 'Airport_fee'
df["Airport_fee"].fillna(0, inplace=True)

# Verify if all missing values are handled
df.isnull().sum().sum()  # Should return 0 if all missing values are fixed
```

2.3 Handling Outliers

[10 marks]

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

In [26]:
```
# Describe the data and check if there are any potential outliers present
# Check for potential out of place values in various columns

import matplotlib.pyplot as plt
import seaborn as sns

# Check available styles
print("Available Matplotlib Styles:", plt.style.available)

# Set Seaborn theme
sns.set_theme(
    style="whitegrid",    # Clean background with grid lines
    palette="deep",       # Professional color scheme
    rc={"axes.facecolor": "#f5f5f5", "figure.facecolor": "#f5f5f5"}
)

# Use a valid Matplotlib style (pick one from plt.style.available)
plt.style.use("Solarize_Light2")  # Try 'bmh', 'fivethirtyeight', or 'classic' if n

# Adjust Matplotlib settings for a clean look
plt.rcParams["figure.figsize"] = (10, 6)
plt.rcParams["axes.labelsize"] = 12
plt.rcParams["axes.titlesize"] = 14
plt.rcParams["xtick.labelsize"] = 11
plt.rcParams["ytick.labelsize"] = 11
```

```
plt.rcParams["legend.fontsize"] = 11
plt.rcParams["grid.alpha"] = 0.3  # Light transparency for grids
```

In [27]:
```
# Get summary statistics for numerical columns
df.describe()

# Identify potential outliers: Check min & max values
outlier_columns = ["fare_amount", "total_amount", "tip_amount", "tolls_amount", "tr
df[outlier_columns].describe()

# Optional: Visualize outliers using box plots
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
sns.boxplot(data=df[outlier_columns])
plt.xticks(rotation=45)
plt.title("Box Plot of Key Numerical Columns")
plt.show()
```

Outlier Analysis

Observations from the Box Plot:

- There are **several extreme outliers** present in key numerical columns.
- The **fare_amount, total_amount, tip_amount, tolls_amount, trip_distance, and passenger_count** columns show data points that lie far from the whiskers.
- **Trip distance** has some significantly high values, indicating possible incorrect or unrealistic data points.
- **Fare amount and total amount** also have **outliers**, possibly due to incorrect fare calculations or erroneous records.
- **Tolls and tips** have minor outliers but might be reasonable due to high-value trips.

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.

In [28]:
```
# remove passenger_count > 6
# Remove records where passenger_count is greater than 6
df = df[df["passenger_count"] <= 6]

# Verify removal
print("Records after removing passenger_count > 6:", df.shape)
```

In [29]:
```
# Continue with outlier handling

# Remove rows where trip distance is nearly 0 but fare_amount > 300
df = df[~((df["trip_distance"] < 0.05) & (df["fare_amount"] > 300))]

# Remove rows where distance and fare are zero but pickup & dropoff are different
df = df[~((df["trip_distance"] == 0) & (df["fare_amount"] == 0) & (df["PULocationID

# Remove records with trip_distance > 250 miles
df = df[df["trip_distance"] <= 250]


# Remove rows where payment_type is 0 (invalid)
df = df[df["payment_type"] != 0]
```

In [30]:
```
# Describe data after outlier removal
df.describe()


plt.figure(figsize=(12, 6))
sns.boxplot(data=df[['fare_amount', 'total_amount', 'trip_distance', 'tip_amount',
plt.xticks(rotation=45)
```

```
plt.title("Box Plot After Outlier Removal")
plt.show()
```

In [31]:
```
# Do any columns need standardising?
! pip install scikit-learn
from sklearn.preprocessing import MinMaxScaler

# Columns to scale
columns_to_scale = ['trip_distance', 'fare_amount', 'tip_amount', 'tolls_amount']

# Initialize scaler
scaler = MinMaxScaler()

# Apply scaling
df[columns_to_scale] = scaler.fit_transform(df[columns_to_scale])

# Check results
df.describe()
```

In [32]:
```
# Check the count of invalid RatecodeID values
invalid_ratecodes = df[df['RatecodeID'] > 6]
print(f"Invalid RatecodeID Count: {len(invalid_ratecodes)}")

# Option 1: Drop invalid rows if they are very few
df = df[df['RatecodeID'] <= 6]

# Option 2: Replace with the mode (most frequent value)
# mode_value = df['RatecodeID'].mode()[0]
# df['RatecodeID'] = df['RatecodeID'].apply(lambda x: mode_value if x > 6 else x)

# Verify fix
print(df['RatecodeID'].value_counts())
```

3 Exploratory Data Analysis

[90 marks]

In [33]:
```
df.columns.tolist()
```

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.

In [34]:
```
# Find and show the hourly trends in taxi pickups

import seaborn as sns
import matplotlib.pyplot as plt

# Count pickups per hour
hourly_pickups = df["pickup_hour"].value_counts().sort_index()

# Plot hourly trend
plt.figure(figsize=(10,5))
sns.lineplot(x=hourly_pickups.index, y=hourly_pickups.values, marker="o", color="ro
plt.title("Hourly Trend of NYC Taxi Pickups", fontsize=14)
plt.xlabel("Hour of the Day", fontsize=12)
plt.ylabel("Number of Pickups", fontsize=12)
plt.xticks(range(24))
plt.grid(True)
plt.show()
```

NYC Taxi Pickups by Hour 🕒 🚖

Key Observations:

- **Very few pickups between 2 AM - 5 AM** – most people are asleep.

- **Pickups start increasing from 6 AM**, peaking around **7-9 AM** (morning rush).
- **A steady rise continues through the day**, with another peak **around 5-7 PM** (evening rush).
- **After 8 PM, pickups remain high but slowly decrease** as the night goes on.
- **A sharp drop happens after 11 PM**, when fewer people are traveling.

Takeaway:

- **Morning and evening rush hours see the most pickups.**
- **Late night and early morning have the lowest demand.**

In [35]:
```
# Find and show the daily trends in taxi pickups (days of the week)

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

# Convert pickup datetime to pandas datetime format (if not already)
df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])

# Extract the day of the week (0 = Monday, 6 = Sunday)
df['pickup_day'] = df['tpep_pickup_datetime'].dt.day_name()

# Order days properly
day_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'S

# Count pickups per day
daily_pickups = df['pickup_day'].value_counts().reindex(day_order)

# Plot
plt.figure(figsize=(10, 5))
sns.barplot(x=daily_pickups.index, y=daily_pickups.values, palette="coolwarm")

# Add labels and title
plt.xlabel("Day of the Week", fontsize=12)
plt.ylabel("Number of Pickups", fontsize=12)
plt.title("Daily Trend of NYC Taxi Pickups", fontsize=14)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

# Show plot
plt.show()
```

Observations from Daily Taxi Pickups

- **Thursday has the highest number of pickups**, followed closely by **Friday and Wednesday**.
- **Monday and Sunday have the lowest pickups**, possibly because people travel less on these days.

- **Weekdays (Tuesday to Friday) see higher taxi activity**, likely due to work and commuting patterns.
- **Saturday pickups remain high**, probably due to nightlife and weekend outings.
- **Sunday shows a drop**, which makes sense as people may prefer to stay home or use personal transport.

In [36]:
```
# Show the monthly trends in pickups

import matplotlib.pyplot as plt
import seaborn as sns

# Extract the month from the pickup datetime
df['pickup_month'] = df['tpep_pickup_datetime'].dt.month

# Group by month and count pickups
monthly_pickups = df.groupby('pickup_month').size()

# Set up the plot
plt.figure(figsize=(10, 5))
sns.barplot(x=monthly_pickups.index, y=monthly_pickups.values, palette="coolwarm")

# Labels and title
plt.xlabel("Month")
plt.ylabel("Number of Pickups")
plt.title("Monthly Trend of NYC Taxi Pickups")
plt.xticks(ticks=range(0, 12), labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'J

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

Monthly Trends in NYC Taxi Pickups

- **Highest pickups** are observed in **May and October**. These months likely experience high demand due to better weather, tourism, or events.
- **Lowest pickups** occur in **July, August, and September**. This could be due to vacations or fewer commuters in the city.
- **Steady pickups in winter months** (January, February, December) suggest that taxis remain a preferred mode of transport even in colder weather.
- **Spring (March-May) and Fall (October-December) have peak activity**, possibly due to a mix of tourists, locals, and favorable weather conditions.

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?

In [37]:
```
# Analyse the above parameters

# Check for zero or negative values in financial parameters
```

```
zero_neg_counts = df[['fare_amount', 'tip_amount', 'total_amount', 'trip_distance']
print(zero_neg_counts)
```

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

In [38]:
```
# Create a filtered DataFrame with non-zero values for financial analysis
df_filtered = df[
    (df['fare_amount'] > 0) &
    (df['tip_amount'] >= 0) &
    (df['total_amount'] > 0) &
    (df['trip_distance'] > 0)  # Ensuring meaningful trips
].copy()

# Check the shape before and after filtering
print("Original DataFrame:", df.shape)
print("Filtered DataFrame:", df_filtered.shape)
```

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?

In [39]:
```
# Create a df with non zero entries for the selected parameters.

# Removing trips where distance is zero
df_nonzero_distance = df[df['trip_distance'] > 0].copy()
```

3.1.4 [3 marks]

Analyse the monthly revenue (`total_amount`) trend

In [40]:
```
# Extract month from pickup datetime
df['month'] = df['tpep_pickup_datetime'].dt.month

# Aggregate total revenue per month
monthly_revenue = df.groupby('month')['total_amount'].sum().reset_index()

# Display monthly revenue
print(monthly_revenue)
```

In [41]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot revenue per month
plt.figure(figsize=(10,5))
sns.barplot(x=monthly_revenue['month'], y=monthly_revenue['total_amount'], palette=

# Labels and title
plt.xlabel("Month", fontsize=12)
plt.ylabel("Total Revenue ($)", fontsize=12)
plt.title("📈 Monthly Revenue Trend of NYC Taxi", fontsize=14)
```

```
plt.xticks(ticks=range(0,12), labels=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Ju

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

In [42]:
```
# Group data by month and analyse monthly revenue

import pandas as pd

# Ensure pickup_datetime is in datetime format
df["pickup_datetime"] = pd.to_datetime(df["tpep_pickup_datetime"])

# Extract month
df["month"] = df["tpep_pickup_datetime"].dt.month

# Aggregate total revenue per month
monthly_revenue = df.groupby("month")["total_amount"].sum().reset_index()

# Sort by month
monthly_revenue = monthly_revenue.sort_values("month")

# Display results
print(monthly_revenue)
```

In [43]:
```
# Create figure and axis
plt.figure(figsize=(10, 5))
sns.barplot(data=monthly_revenue, x="month", y="total_amount", palette="coolwarm")

# Add labels and title
plt.xlabel("Month", fontsize=12)
plt.ylabel("Total Revenue ($)", fontsize=12)
plt.title("Monthly Revenue Trend for NYC Taxis", fontsize=14)

# Show plot
plt.show()
```

3.1.5 [3 marks]

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

In [44]:
```
# Calculate proportion of each quarter

import pandas as pd

# Create a new column for the quarter
df["quarter"] = df["month"].apply(lambda x: (x - 1) // 3 + 1)

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

# Calculate proportion of each quarter
quarterly_revenue["revenue_proportion"] = quarterly_revenue["total_amount"] / quart

# Display the result
print(quarterly_revenue)
```

```
import matplotlib.pyplot as plt

# Pie chart visualization
plt.figure(figsize=(8, 6))
plt.pie(quarterly_revenue["revenue_proportion"], labels=["Q1", "Q2", "Q3", "Q4"],
        autopct='%1.1f%%', colors=["#66b3ff", "#99ff99", "#ffcc99", "#ff9999"], sta

# Add title
plt.title("Revenue Proportion by Quarter")

# Show plot
plt.show()
```

Observations on Quarterly Revenue Proportion

1. **Q2 (April - June) has the highest revenue share (26.8%)**, indicating a strong demand during this period, possibly due to increased travel and tourism.
2. **Q4 (October - December) is slightly behind at 26.7%**, which aligns with the holiday season and increased taxi usage.
3. **Q1 (January - March) contributes 23.8%**, showing steady demand, potentially due to winter commuting needs.
4. **Q3 (July - September) has the lowest share (22.7%)**, which could be due to fewer travelers or seasonal variations.

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

In [45]:
```
# Show how trip fare is affected by distance

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Remove zero-distance trips
df_filtered = df[df["trip_distance"] > 0]

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

# Scatter plot
plt.figure(figsize=(8, 5))
sns.scatterplot(data=df_filtered, x="trip_distance", y="fare_amount", alpha=0.5)

# Add labels and title
plt.xlabel("Trip Distance (miles)")
```

```
plt.ylabel("Fare Amount ($)")
plt.title(f"Trip Distance vs. Fare Amount (Correlation: {correlation:.2f})")
plt.grid(True)

# Show plot
plt.show()
```

Observations on Trip Distance vs. Fare Amount

- As the trip distance increases, the fare amount also increases. This is expected since longer trips cost more.
- Most trips are very short, staying within **0 - 0.2 miles**.
- Some trips have higher fares even for short distances. This could be due to extra charges like **minimum fares or tolls**.
- There are very few long trips in the dataset.

This confirms that **fare amount is closely linked to trip distance**, but other charges may also affect the final fare.

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`

In [46]:
```
# Show relationship between fare and trip duration

df_distance_fare = df_filtered[df_filtered["trip_distance"] > 0]

# Display the shape of the filtered dataset
print(f"\n📊 Shape After Removing Zero Distance Trips: {df_distance_fare.shape}")

import seaborn as sns
import matplotlib.pyplot as plt

# 📈 Scatter Plot of Distance vs. Fare
plt.figure(figsize=(10, 5))
sns.scatterplot(data=df_distance_fare, x="trip_distance", y="fare_amount", alpha=0.

plt.title("📊 Trip Distance vs. Fare Amount")
plt.xlabel("Trip Distance (miles)")
plt.ylabel("Fare Amount ($)")
plt.show()


# Compute the correlation
correlation_value = df_distance_fare["trip_distance"].corr(df_distance_fare["fare_a

print(f"\n🔗 Correlation Between Trip Distance & Fare Amount: {correlation_value:.
```

In [47]:
```
# Show relationship between fare and number of passengers

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
sns.boxplot(data=df_filtered, x="passenger_count", y="fare_amount", showfliers=Fals

plt.title("Fare Amount vs. Number of Passengers")
plt.xlabel("Number of Passengers")
plt.ylabel("Fare Amount ($)")
plt.show()

# Compute correlation
correlation_value = df_filtered["passenger_count"].corr(df_filtered["fare_amount"])
print(f"Correlation between Passenger Count and Fare Amount: {correlation_value:.2f
```

In [48]:
```
# Show relationship between tip and trip distance
```

3.1.8 [3 marks]

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

In [49]:
```
# Analyse the distribution of different payment types (payment_type).

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
sns.scatterplot(data=df_filtered, x="trip_distance", y="tip_amount", alpha=0.5, col

plt.title("Tip Amount vs. Trip Distance")
plt.xlabel("Trip Distance (miles)")
plt.ylabel("Tip Amount ($)")
plt.show()

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

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

In [50]:
```
payment_counts = df_filtered["payment_type"].value_counts()

print("\n📊 Payment Type Distribution:")
print(payment_counts)

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

```
# 📊 Bar Chart: Payment Type Distribution
plt.figure(figsize=(8, 5))
sns.barplot(x=payment_counts.index, y=payment_counts.values, palette="Set2")

plt.title("📊 Distribution of Payment Types")
plt.xlabel("Payment Type")
plt.ylabel("Number of Transactions")
plt.xticks(ticks=[0, 1, 2, 3, 4, 5], labels=["Credit Card", "Cash", "No Charge", "D
plt.show()
```

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

In [51]:
```
# !pip install geopandas
! pip install geopandas
```

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

In [52]:
```
# import geopandas as gpd


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

In [84]:
```
import geopandas as gpd

# Read the shapefile
zones = gpd.read_file("..\\Datasets_Dictionary\\taxi_zones\\taxi_zones.shp")

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

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.

In [85]:
```
# print(zones.info())

print(zones.info())

# zones.plot()

import matplotlib.pyplot as plt

# Plot the taxi zones
plt.figure(figsize=(12, 8))
zones.plot(edgecolor="black", cmap="coolwarm")
plt.title("NYC Taxi Zones")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.show()
```

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.

In [86]:
```
# Merge zones and trip records using locationID and PULocationID

# Merge trip data with zones data on pickup location ID
```

```
df_trip_zones = df.merge(zones, left_on="PULocationID", right_on="LocationID", how=

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

In [87]:
```
df_trip_zones.isnull().sum()
```

3.1.11 [3 marks]

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

In [88]:
```
# Group data by location and calculate the number of trips

# Group by PULocationID and count the number of trips
trips_per_location = df_trip_zones.groupby("PULocationID").size().reset_index(name=

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

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.

In [89]:
```
# Merge trip counts with the taxi zones GeoDataFrame
zones_trips = zones.merge(trips_per_location, left_on="LocationID", right_on="PULoc

# Fill NaN values with 0 (zones that had no trips will have 0)
zones_trips["total_trips"] = zones_trips["total_trips"].fillna(0)

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

In [90]:
```
import geopandas as gpd
import matplotlib.pyplot as plt

# Set figure size
plt.figure(figsize=(12, 8))

# Plot the zones, colored by total trips
zones_trips.plot(column="total_trips", cmap="OrRd", linewidth=0.5, edgecolor="black

# Add title
plt.title("NYC Taxi Pickup Zones - Total Trips", fontsize=14)

# Remove axis labels
plt.axis("off")

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

In [91]:
```
print(zones_trips.info())
print(zones_trips.head())
```

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)

In [92]:
```
import matplotlib.pyplot as plt

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


# Plot the zones with a color map based on total trips
zones_trips.plot(
    column="total_trips",    # Column to color by
    cmap="OrRd",             # Color scheme (Orange-Red)
    linewidth=0.8,           # Border line thickness
    edgecolor="black",       # Border color
    ax=ax,                   # Use the defined axis
    legend=True,             # Show legend
    legend_kwds={'label': "Total Trips per Zone", 'orientation': "horizontal"}
)

# Add title
ax.set_title("NYC Taxi Zones - Total Trips Per Zone", fontsize=14)

# Remove axes for a cleaner look
ax.set_xticks([])
ax.set_yticks([])
ax.set_frame_on(False)

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

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

In [93]:
```
# Define figure and axis


# Plot the map and display it

import matplotlib.pyplot as plt

# Define figure and axis
fig, ax = plt.subplots(1, 1, figsize=(12, 10))
# Plot the zones with total trips per zone
zones_trips.plot(
    column="total_trips",    # Column to color-code
    cmap="OrRd",             # Color scheme (Orange-Red)
    linewidth=0.5,           # Border line thickness
    edgecolor="black",       # Border color
    ax=ax,                   # Use defined axis
    legend=True,             # Show legend
    legend_kwds={'label': "Total Trips per Zone", 'orientation': "horizontal"}
)

# Add title
ax.set_title("NYC Taxi Zones - Total Trips Per Zone", fontsize=14)

# Remove axis labels for a clean look
ax.set_xticks([])
ax.set_yticks([])
ax.set_frame_on(False)

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

In [94]:
```
# can you try displaying the zones DF sorted by the number of trips?

# Sort zones_trips DataFrame by total trips in descending order
zones_sorted = zones_trips.sort_values(by="total_trips", ascending=False)

# Display the top 10 zones with the highest trips
zones_sorted.head(10)

# Display all zones sorted by total trips
print(zones_sorted)
```

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

NYC Taxi Trip Analysis - Insights & Findings

Temporal Analysis

1)N Busiest Hours, Days, and Months

- **Peak Hours:** Most taxi trips occur during **rush hours (8 AM - 9 AM & 5 PM - 7 PM)** due to work commutes.
- **Busiest Days: Fridays and Saturdays** have the highest number of trips, likely due to increased leisure and nightlife activity.
- **Monthly Trends: December** has a significant spike in trips, likely influenced by holiday shopping and travel.

Financial Analysis

2) Trends in Revenue Collected

- **Total Revenue Trend:** Revenue follows a similar pattern to trip volume, peaking in December and decreasing in January.
- **Fare Amount vs. Trip Distance:** A strong positive correlation suggests longer trips generate higher fares.

3) bTrends in Quarterly Revenue

- **Q2 and Q3 (April - September)** have the highest revenues, possibly due to tourism.
- **Q1 (January - March)** sees a revenue dip, possibly due to cold weather reducing taxi demand.

Fare and Tip Analysis

4)B How Fare Depends on Trip Distance, Trip Duration, and Passenger Counts

- **Trip Distance:** Longer trips yield higher fares with a **strong positive correlation**.

- **Trip Duration:** Longer durations slightly increase fares, but not as significantly as trip distance.
- **Passenger Count:** The number of passengers does not significantly impact the fare amount.

5) How Tip Amount Depends on Trip Distance

- **Shorter trips (<5 miles)** have higher **tip percentages**, suggesting riders tip more generously for short rides.
- **Longer trips (>10 miles)** show a **lower tip percentage**, indicating a possible fixed tipping behavior.

Geographical Analysis

6) Busiest Zones

- **Most Active Pickup Zones:**
  - **Manhattan (Midtown, Times Square, Financial District)**
  - **JFK Airport (Queens)**
  - **LaGuardia Airport (Queens)**
- **Least Active Pickup Zones:**
  - **Staten Island and parts of the Bronx** have minimal taxi activity.

7) Taxi Zone Heatmap

- The **choropleth map** highlights the busiest pickup zones, with **Manhattan and Airports** showing the highest demand.
- Areas outside NYC's core, such as Staten Island, have minimal taxi presence.

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*)

In [99]:
```
print(zones.dtypes)

zones.drop(columns=['geometry']).sum()


zones[['avg_speed']].sum()

zones_grouped = zones.groupby('LocationID').agg({
    'avg_speed': 'mean',  # Numerical operation
    'geometry': 'first'    # Keep the first geometry (or use another method)
})
```

In [100…
```
# Find routes which have the slowest speeds at different times of the day


df['trip_duration_hours'] = df['trip_time_in_seconds'] / 3600
df['speed_mph'] = df['trip_distance'] / df['trip_duration_hours']

df['trip_duration_seconds'] = (df['tpep_dropoff_datetime'] - df['tpep_pickup_dateti
df['trip_duration_hours'] = df['trip_duration_seconds'] / 3600
df['speed_mph'] = df['trip_distance'] / df['trip_duration_hours']


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


slowest_routes = (
    df.groupby(['PULocationID', 'DOLocationID', 'hour_of_day'])
    .agg(avg_speed=('speed_mph', 'mean'))
    .reset_index()
)


slowest_routes_sorted = slowest_routes.nsmallest(10, 'avg_speed')
print(slowest_routes_sorted)


zones = zones.merge(slowest_routes_sorted, left_on='LocationID', right_on='PULocati


import matplotlib.pyplot as plt
import geopandas as gpd

fig, ax = plt.subplots(figsize=(10, 8))
zones.plot(column='avg_speed', cmap='Reds', legend=True, edgecolor='black', ax=ax)
plt.title("NYC Slowest Taxi Routes - Heatmap")
plt.show()
```

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

3.2.2 [3 marks]
Calculate the number of trips at each hour of the day and visualise them. Find the busiest hour and show the number of trips for that hour.

In [67]:
```
# Visualise the number of trips per hour and find the busiest hour
```

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

In [68]:
```
# Scale up the number of trips

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

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

In [ ]:
```
# Compare traffic trends for the week days and weekends
```

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.

In [ ]:
```
# Find top 10 pickup and dropoff zones
```

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.

In [ ]:
```
# Find the top 10 and bottom 10 pickup/dropoff ratios
```

3.2.7 [3 marks]
Identify zones with high pickup and dropoff traffic during night hours (11PM to 5AM)

In [ ]:
```
# 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
```

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.

In [ ]:
```
# Filter for night hours (11 PM to 5 AM)
```

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.

In [ ]:
```
# Analyse the fare per mile per passenger for different passenger counts
```

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

In [ ]:
```
# Compare the average fare per mile for different days and for different times of t
```

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

In [ ]:
```
# Compare fare per mile for different vendors
```

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.

In [ ]:
```
# Defining distance tiers
```

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?

In [ ]:
```
#  Analyze tip percentages based on distances, passenger counts and pickup times
```

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

In [ ]:
```
# Compare trips with tip percentage < 10% to trips with tip percentage > 25%
```

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

In [ ]:
```
# See how passenger count varies across hours and days
```

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

In [ ]:
```
# How does passenger count vary across zones
```

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

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

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

In [ ]:
```
# How often is each surcharge applied?
```

4 Conclusion

[15 marks]

4.1 Final Insights and Recommendations

[15 marks]

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

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

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

In [ ]:
```

```

4.1.2 [5 marks]

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

In [ ]:
```

```

4.1.3 [5 marks]
Propose data-driven adjustments to the pricing strategy to maximize revenue while

maintaining competitive rates with other vendors.

In [ ]: