New York City Yellow Taxi Data

Objective

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

Problem Statement

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

Tasks

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

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

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

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

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

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

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

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

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

Data Understanding

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

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

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

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

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

Data Description

You can find the data description here: Data Dictionary

Trip Records

Field Name	description
VendorID	A code indicating the TPEP provider that provided the record. 1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.
tpep_pickup_datetime	The date and time when the meter was engaged.
tpep_dropoff_datetime	The date and time when the meter was disengaged.
Passenger_count	The number of passengers in the vehicle. This is a driver-entered value.
Trip_distance	The elapsed trip distance in miles reported by the taximeter.
PULocationID	TLC Taxi Zone in which the taximeter was engaged
DOLocationID	TLC Taxi Zone in which the taximeter was disengaged
RateCodeID	The final rate code in effect at the end of the trip. 1 = Standard rate 2 = JFK 3 = Newark 4 = Nassau or Westchester

Field Name	description
	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 [ ]:
```
# Import warnings
import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
# Import the libraries you will be using for analysis

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

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

# Check versions
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

print("numpy version:", np.__version__)
print("pandas version:", pd.__version__)
print("matplotlib version:", plt.matplotlib.__version__)
print("seaborn version:", sns.__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 [295…
```
# Try loading one file

# df = pd.read_parquet('2023-1.parquet')
# df.info()
import pandas as pd
df = pd.read_parquet('2023-3.parquet')
```

```
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3275796 entries, 0 to 3307233
Data columns (total 19 columns):
 #   Column                 Dtype
---  ------                 -----
 0   VendorID               int32
 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           int32
 8   DOLocationID           int32
 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), int32(3), int64(1), object(1)
memory usage: 462.4+ MB
```

Out[295...

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_
count	3.275796e+06	3275796	3275796	3.176486e+06	3.275
mean	1.734740e+00	2023-06-15 18:06:20.461846	2023-06-15 18:24:05.545134	1.369032e+00	4.381
min	1.000000e+00	2002-12-31 23:03:19	2002-12-31 23:06:17	0.000000e+00	0.000
25%	1.000000e+00	2023-06-08 09:09:51.500000	2023-06-08 09:27:12	1.000000e+00	1.090
50%	2.000000e+00	2023-06-15 14:10:44	2023-06-15 14:31:23	1.000000e+00	1.820
75%	2.000000e+00	2023-06-23 02:12:43.750000	2023-06-23 02:28:10.500000	1.000000e+00	3.540
max	6.000000e+00	2023-07-01 00:42:13	2023-07-03 16:31:24	9.000000e+00	1.852
std	4.491018e-01	NaN	NaN	8.970379e-01	2.291

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

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

In [297…
```
# 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
import pandas as pd

# Change directory to where the parquet files are stored
os.chdir(r'C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records')

# Get only .parquet files and ignore system files
file_list = [f for f in os.listdir() if f.endswith('.parquet') and os.path.isfile(f

# Initialize final dataframe
df = pd.DataFrame()

for file_name in file_list:
    try:
        file_path = os.path.join(os.getcwd(), file_name)
        print(f"Processing: {file_path}")

        # Read parquet file
        data = pd.read_parquet(file_path)

        # Ensure datetime column exists and is of datetime type
        if 'tpep_pickup_datetime' not in data.columns:
            print(f"Skipped {file_name} - Missing 'tpep_pickup_datetime'")
            continue

        if not pd.api.types.is_datetime64_any_dtype(data['tpep_pickup_datetime']):
            data['tpep_pickup_datetime'] = pd.to_datetime(data['tpep_pickup_datetim

        # Drop rows with invalid datetime
        data = data.dropna(subset=['tpep_pickup_datetime'])

        # Extract date and hour
        data['pickup_date'] = data['tpep_pickup_datetime'].dt.date
        data['pickup_hour'] = data['tpep_pickup_datetime'].dt.hour

        # Collect sampled data from each hour
        sampled_data = pd.DataFrame()
        for date_val in data['pickup_date'].unique():
            day_data = data[data['pickup_date'] == date_val]
            for hr in range(24):
                hr_data = day_data[day_data['pickup_hour'] == hr]
                if not hr_data.empty:
                    hr_sample = hr_data.sample(frac=0.05, random_state=42)
                    sampled_data = pd.concat([sampled_data, hr_sample], ignore_inde

        # Add to final dataframe
        df = pd.concat([df, sampled_data], ignore_index=True)

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

# Reset index and show result
df.reset_index(drop=True, inplace=True)
print(df.head())
```

```
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-1.par
quet
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-10.pa
rquet
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-11.pa
rquet
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-12.pa
rquet
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-2.par
quet
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-3.par
quet
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-4.par
quet
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-5.par
quet
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-6.par
quet
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-7.par
quet
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-8.par
quet
Processing: C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records\2023-9.par
quet
   VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
0         2  2023-01-01 00:07:18   2023-01-01 00:23:15              1.0
1         2  2023-01-01 00:16:41   2023-01-01 00:21:46              2.0
2         2  2023-01-01 00:14:03   2023-01-01 00:24:36              3.0
3         2  2023-01-01 00:24:30   2023-01-01 00:29:55              1.0
4         2  2023-01-01 00:43:00   2023-01-01 01:01:00              NaN

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID  DOLocationID  \
0           7.74         1.0                  N           138           256
1           1.24         1.0                  N           161           237
2           1.44         1.0                  N           237           141
3           0.54         1.0                  N           143           142
4          19.24         NaN               None            66           107

   payment_type  ...  mta_tax  tip_amount  tolls_amount  \
0             2  ...      0.5        0.00           0.0
1             1  ...      0.5        2.58           0.0
2             2  ...      0.5        0.00           0.0
3             2  ...      0.5        0.00           0.0
4             0  ...      0.5        5.93           0.0

   improvement_surcharge  total_amount  congestion_surcharge  airport_fee  \
0                    1.0         41.15                   0.0         1.25
1                    1.0         15.48                   2.5         0.00
2                    1.0         16.40                   2.5         0.00
3                    1.0         11.50                   2.5         0.00
4                    1.0         35.57                   NaN          NaN

   pickup_date  pickup_hour Airport_fee
0   2023-01-01            0         NaN
1   2023-01-01            0         NaN
2   2023-01-01            0         NaN
```

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

[5 rows x 22 columns]
```

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

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

In [299…
```
# Store the df in csv/parquet
# df.to_parquet('')
df.to_parquet(r'C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records.parqu
```

2 Data Cleaning

[30 marks]

Now we can load the new data directly.

In [301…
```
# Load the new data file
df= pd.read_parquet(r'C:\Users\aryan\Downloads\Datasets and Dictionary\trip_records
```

In [303…
```
# df.head()
df.head()
```

Out[303…

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
0	2	2023-01-01 00:07:18	2023-01-01 00:23:15	1.0	7.74
1	2	2023-01-01 00:16:41	2023-01-01 00:21:46	2.0	1.24
2	2	2023-01-01 00:14:03	2023-01-01 00:24:36	3.0	1.44
3	2	2023-01-01 00:24:30	2023-01-01 00:29:55	1.0	0.54
4	2	2023-01-01 00:43:00	2023-01-01 01:01:00	NaN	19.24

5 rows × 22 columns

In [305…
```
# df.info()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1896400 entries, 0 to 1896399
Data columns (total 22 columns):
 #   Column                 Dtype
---  ------                 -----
 0   VendorID               int64
 1   tpep_pickup_datetime   datetime64[us]
 2   tpep_dropoff_datetime  datetime64[us]
 3   passenger_count        float64
 4   trip_distance          float64
 5   RatecodeID             float64
 6   store_and_fwd_flag     object
 7   PULocationID           int64
 8   DOLocationID           int64
 9   payment_type           int64
 10  fare_amount            float64
 11  extra                  float64
 12  mta_tax                float64
 13  tip_amount             float64
 14  tolls_amount           float64
 15  improvement_surcharge  float64
 16  total_amount           float64
 17  congestion_surcharge   float64
 18  airport_fee            float64
 19  pickup_date            object
 20  pickup_hour            int32
 21  Airport_fee            float64
dtypes: datetime64[us](2), float64(13), int32(1), int64(4), object(2)
memory usage: 311.1+ 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

In [312…

```
# Fix the index and drop any columns that are not needed
df = df.reset_index(drop=True)
df = df.drop(columns=['store_and_fwd_flag'], errors='ignore')
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1896400 entries, 0 to 1896399
Data columns (total 21 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   PULocationID           int64
 7   DOLocationID           int64
 8   payment_type           int64
 9   fare_amount            float64
 10  extra                  float64
 11  mta_tax                float64
 12  tip_amount             float64
 13  tolls_amount           float64
 14  improvement_surcharge  float64
 15  total_amount           float64
 16  congestion_surcharge   float64
 17  airport_fee            float64
 18  pickup_date            object
 19  pickup_hour            int32
 20  Airport_fee            float64
dtypes: datetime64[us](2), float64(13), int32(1), int64(4), object(1)
memory usage: 296.6+ MB
```

Out[312...

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_
count	1.896400e+06	1896400	1896400	1.831526e+06	1.896
mean	1.733026e+00	2023-07-02 19:59:52.930795	2023-07-02 20:17:18.919563	1.369215e+00	3.858
min	1.000000e+00	2022-12-31 23:51:30	2022-12-31 23:56:06	0.000000e+00	0.000
25%	1.000000e+00	2023-04-02 16:10:08.750000	2023-04-02 16:27:43.500000	1.000000e+00	1.050
50%	2.000000e+00	2023-06-27 15:44:22.500000	2023-06-27 16:01:15	1.000000e+00	1.790
75%	2.000000e+00	2023-10-06 19:37:45	2023-10-06 19:53:39	1.000000e+00	3.400
max	6.000000e+00	2023-12-31 23:57:51	2024-01-01 20:50:55	9.000000e+00	1.263
std	4.476401e-01	NaN	NaN	8.927560e-01	1.294

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 [316…
```
# Combine the two airport fee columns
cols = ['airport_fee', 'Airport_fee']
df['Airport_fee'] = df[cols].fillna(0).sum(axis=1) if all(c in df.columns for c in
df.drop(columns=[c for c in cols if c in df.columns and c != 'Airport_fee'], inplac
df.describe()
```

Out[316…

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_
count	1.896400e+06	1896400	1896400	1.831526e+06	1.896
mean	1.733026e+00	2023-07-02 19:59:52.930795	2023-07-02 20:17:18.919563	1.369215e+00	3.858
min	1.000000e+00	2022-12-31 23:51:30	2022-12-31 23:56:06	0.000000e+00	0.000
25%	1.000000e+00	2023-04-02 16:10:08.750000	2023-04-02 16:27:43.500000	1.000000e+00	1.050
50%	2.000000e+00	2023-06-27 15:44:22.500000	2023-06-27 16:01:15	1.000000e+00	1.790
75%	2.000000e+00	2023-10-06 19:37:45	2023-10-06 19:53:39	1.000000e+00	3.400
max	6.000000e+00	2023-12-31 23:57:51	2024-01-01 20:50:55	9.000000e+00	1.263
std	4.476401e-01	NaN	NaN	8.927560e-01	1.294

2.1.3 [5 marks]

Fix columns with negative (monetary) values

In [322…
```
# check where values of fare amount are negative
df[df[ "fare_amount" ] < 0.0]
```

Out[322…

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance

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

In [324…
```
# Analyse RatecodeID for the negative fare amounts
df[df["fare_amount"] < 0.0]["RatecodeID"].value_counts()
```

Out[324…
```
Series([], Name: count, dtype: int64)
```

In [328…
```
# Find which columns have negative values
print("Columns with negative values:", [col for col in df.select_dtypes(include='nu
```

```
Columns with negative values: ['extra', 'mta_tax', 'improvement_surcharge', 'total_a
mount', 'congestion_surcharge', 'Airport_fee']
```

In [332…
```
# fix these negative values
# Negative values to abs() or positive values.
```

```
df[df.select_dtypes(include='number').columns] = df.select_dtypes(include='number')
df.describe()
```

Out[332…

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_
count	1.896400e+06	1896400	1896400	1.831526e+06	1.896
mean	1.733026e+00	2023-07-02 19:59:52.930795	2023-07-02 20:17:18.919563	1.369215e+00	3.858
min	1.000000e+00	2022-12-31 23:51:30	2022-12-31 23:56:06	0.000000e+00	0.000
25%	1.000000e+00	2023-04-02 16:10:08.750000	2023-04-02 16:27:43.500000	1.000000e+00	1.050
50%	2.000000e+00	2023-06-27 15:44:22.500000	2023-06-27 16:01:15	1.000000e+00	1.790
75%	2.000000e+00	2023-10-06 19:37:45	2023-10-06 19:53:39	1.000000e+00	3.400
max	6.000000e+00	2023-12-31 23:57:51	2024-01-01 20:50:55	9.000000e+00	1.263
std	4.476401e-01	NaN	NaN	8.927560e-01	1.294

2.2 Handling Missing Values

[10 marks]

2.2.1 [2 marks]

Find the proportion of missing values in each column

In [335…

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

Out[335…
```
VendorID                0.000000
tpep_pickup_datetime    0.000000
tpep_dropoff_datetime   0.000000
passenger_count         0.034209
trip_distance           0.000000
RatecodeID              0.034209
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    0.034209
pickup_date             0.000000
pickup_hour             0.000000
Airport_fee             0.000000
dtype: float64
```

2.2.2 [3 marks]

Handling missing values in `passenger_count`

In [339…
```
# Display the rows with null values
df[df.isnull().any(axis=1)]

# Impute NaN values in 'passenger_count'
# Replacing the  NaN in 'passenger_count' with its mean value
df['passenger_count'] = df['passenger_count'].fillna(df['passenger_count'].mean())
df. describe()
```

Out[339…

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_
count	1.896400e+06	1896400	1896400	1.896400e+06	1.896
mean	1.733026e+00	2023-07-02 19:59:52.930795	2023-07-02 20:17:18.919563	1.369215e+00	3.858
min	1.000000e+00	2022-12-31 23:51:30	2022-12-31 23:56:06	0.000000e+00	0.000
25%	1.000000e+00	2023-04-02 16:10:08.750000	2023-04-02 16:27:43.500000	1.000000e+00	1.050
50%	2.000000e+00	2023-06-27 15:44:22.500000	2023-06-27 16:01:15	1.000000e+00	1.790
75%	2.000000e+00	2023-10-06 19:37:45	2023-10-06 19:53:39	1.369215e+00	3.400
max	6.000000e+00	2023-12-31 23:57:51	2024-01-01 20:50:55	9.000000e+00	1.263
std	4.476401e-01	NaN	NaN	8.773530e-01	1.294

Did you find zeroes in passenger_count? Handle these.

2.2.3 [2 marks]

Handle missing values in `RatecodeID`

In [341…
```
# Zeros in passenger count?
# Replacing the zeros in passenger_count with the mean of the column.
df['passenger_count'] = df['passenger_count'].replace(0, df['passenger_count'].mean
df.describe()
```

Out[341…

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_
count	1.896400e+06	1896400	1896400	1.896400e+06	1.896
mean	1.733026e+00	2023-07-02 19:59:52.930795	2023-07-02 20:17:18.919563	1.390645e+00	3.858
min	1.000000e+00	2022-12-31 23:51:30	2022-12-31 23:56:06	1.000000e+00	0.000
25%	1.000000e+00	2023-04-02 16:10:08.750000	2023-04-02 16:27:43.500000	1.000000e+00	1.050
50%	2.000000e+00	2023-06-27 15:44:22.500000	2023-06-27 16:01:15	1.000000e+00	1.790
75%	2.000000e+00	2023-10-06 19:37:45	2023-10-06 19:53:39	1.369215e+00	3.400
max	6.000000e+00	2023-12-31 23:57:51	2024-01-01 20:50:55	9.000000e+00	1.263
std	4.476401e-01	NaN	NaN	8.602016e-01	1.294

2.2.4 [3 marks]

Impute NaN in `congestion_surcharge`

In [359…
```
# handle null values in congestion_surcharge
print("\nMean of congestion_surcharge:", df['congestion_surcharge'].mean())
```

```
Mean of congestion_surcharge: 2.3076770408937683
```

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

In [365…
```
# Handle any remaining missing values
df.isnull().sum()
```

Out[365…
```
VendorID                    0
tpep_pickup_datetime        0
tpep_dropoff_datetime       0
passenger_count             0
trip_distance               0
RatecodeID              64874
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    64874
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.

In [368…
```
# Describe the data and check if there are any potential outliers present
# Check for potential out of place values in various columns
df.describe()
```

Out[368…

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_
count	1.896400e+06	1896400	1896400	1.896400e+06	1.896
mean	1.733026e+00	2023-07-02 19:59:52.930795	2023-07-02 20:17:18.919563	1.390645e+00	3.858
min	1.000000e+00	2022-12-31 23:51:30	2022-12-31 23:56:06	1.000000e+00	0.000
25%	1.000000e+00	2023-04-02 16:10:08.750000	2023-04-02 16:27:43.500000	1.000000e+00	1.050
50%	2.000000e+00	2023-06-27 15:44:22.500000	2023-06-27 16:01:15	1.000000e+00	1.790
75%	2.000000e+00	2023-10-06 19:37:45	2023-10-06 19:53:39	1.369215e+00	3.400
max	6.000000e+00	2023-12-31 23:57:51	2024-01-01 20:50:55	9.000000e+00	1.263
std	4.476401e-01	NaN	NaN	8.602016e-01	1.294

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 [370…

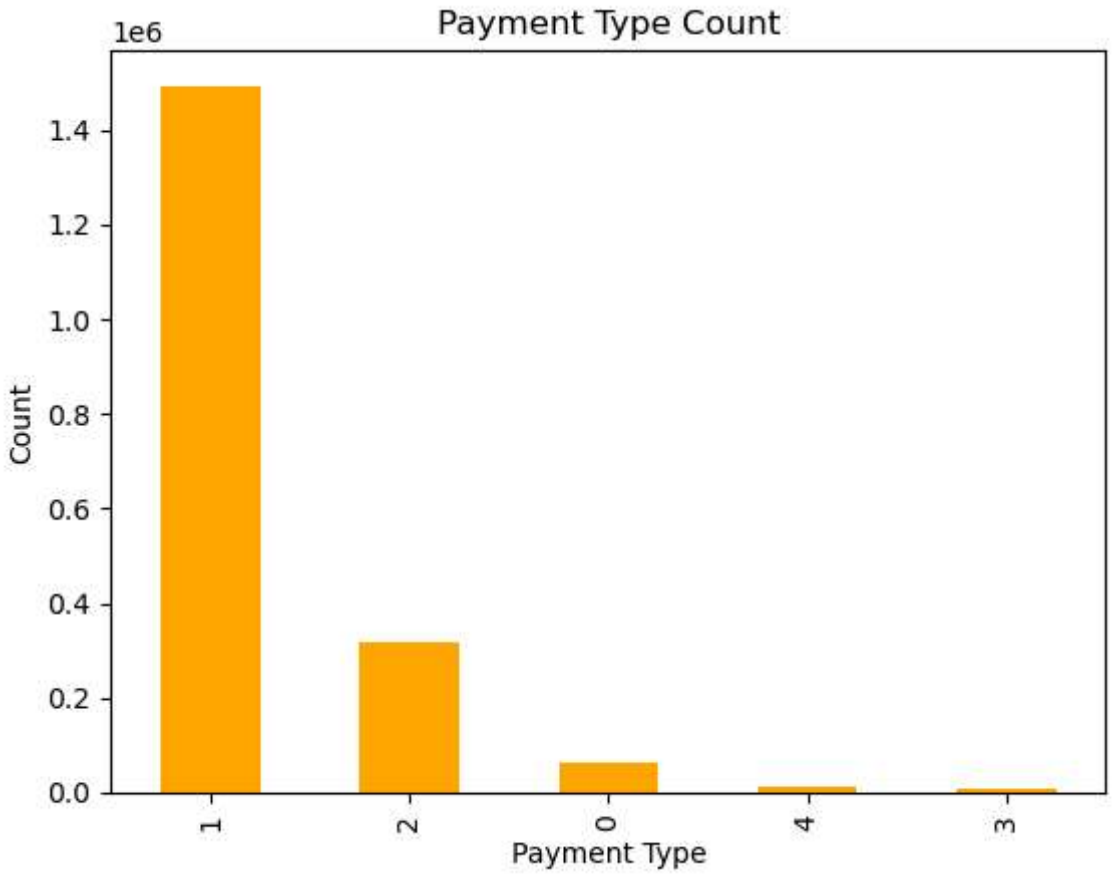
```
# remove passenger_count > 6
df [df["passenger_count"] > 6]
```

Out[370…

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_di
88797	2	2023-01-19 16:33:22	2023-01-19 17:57:32	8.0	
230693	2	2023-10-15 08:11:40	2023-10-15 08:39:01	7.0	
261244	2	2023-10-20 16:55:31	2023-10-20 16:56:27	7.0	
485300	2	2023-11-30 00:13:36	2023-11-30 00:13:39	8.0	
511725	2	2023-12-04 15:32:03	2023-12-04 15:32:08	9.0	
546188	2	2023-12-09 22:01:38	2023-12-09 22:01:40	8.0	
612517	2	2023-12-20 19:26:27	2023-12-20 19:33:17	9.0	
624897	2	2023-12-22 23:00:21	2023-12-22 23:00:24	8.0	
631419	2	2023-12-26 17:38:04	2023-12-26 17:38:06	8.0	
885785	2	2023-06-11 11:53:08	2023-06-11 11:53:29	9.0	
1059918	2	2023-08-16 06:10:57	2023-08-16 06:49:47	8.0	
1082350	2	2023-08-21 01:53:09	2023-08-21 01:53:11	8.0	
1171036	2	2023-02-08 23:26:39	2023-02-08 23:26:51	9.0	
1233147	2	2023-02-19 17:19:13	2023-02-19 17:57:24	9.0	
1333135	2	2023-04-09 09:22:54	2023-04-09 09:23:22	7.0	
1401814	2	2023-04-21 16:44:17	2023-04-21 16:44:19	8.0	
1421730	2	2023-04-28 02:24:47	2023-04-28 02:25:01	8.0	
1597591	2	2023-05-29 02:35:04	2023-05-29 02:35:16	7.0	
1679906	2	2023-07-16 16:33:55	2023-07-16 16:34:00	8.0	
1846867	2	2023-09-18 13:07:26	2023-09-18 14:05:27	8.0	
1848113	2	2023-09-18 17:26:13	2023-09-18 17:52:25	7.0	

In [380…

```
# Continue with outlier handling

# Example for outliers:-  short trips with unusually high tips
shortrideshigh_tip = df[(df["trip_distance"] < 0.5) & (df["tip_amount"] > 50)]

df['payment_type'].value_counts().plot.bar(color = 'Orange')
plt.title("Payment Type Count")
plt.xlabel("Payment Type")
plt.ylabel("Count")
plt.show()
```



In [394…

```
# Do any columns need standardising?
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Create a simple dataset
data = {
    'trip_distance': [18.30, 7.60, 16.17, 0.07],
    'fare_amount': [85, 70, 74, 92],
    'passenger_count': [8.0, 7.0, 7.0, 9.0]
}

df = pd.DataFrame(data)

# Standardizing the numeric columns
scaler = StandardScaler()
df.loc[:, df.select_dtypes("number").columns] = scaler.fit_transform(df.select_dtyp
print(df)
```

```
   trip_distance  fare_amount  passenger_count
0       1.071216     0.544191         0.301511
1      -0.404896    -1.174308        -0.904534
2       0.777373    -0.716041        -0.904534
3      -1.443692     1.346158         1.507557
```

```
C:\Users\aryan\AppData\Local\Temp\ipykernel_19784\1192289417.py:16: FutureWarning: S
etting an item of incompatible dtype is deprecated and will raise in a future error
of pandas. Value '[ 0.54419149 -1.17430796 -0.71604144  1.3461579 ]' has dtype incom
patible with int64, please explicitly cast to a compatible dtype first.
  df.loc[:, df.select_dtypes("number").columns] = scaler.fit_transform(df.select_dty
pes("number")).astype("float64")
```

3 Exploratory Data Analysis

[90 marks]

In [396…
```
df.columns.tolist()
```

Out[396…
```
['trip_distance', 'fare_amount', 'passenger_count']
```

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`

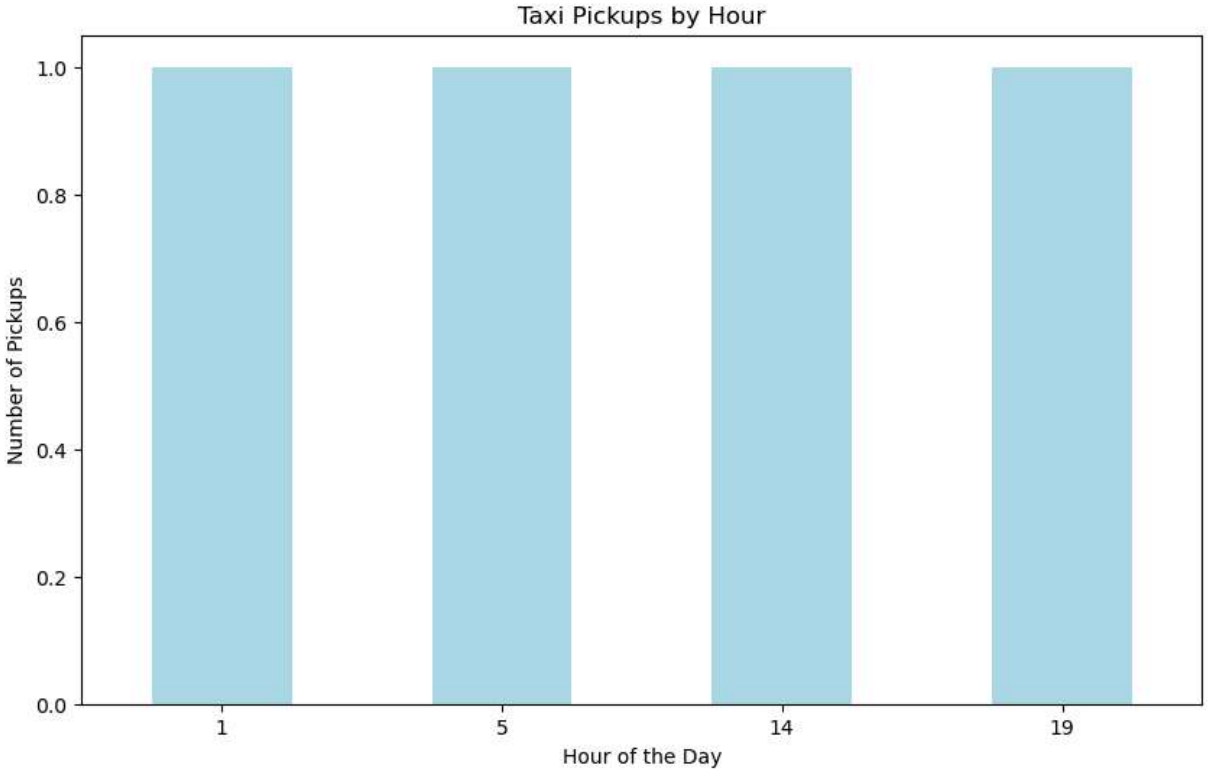
Temporal Analysis

3.1.2 [5 marks]

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

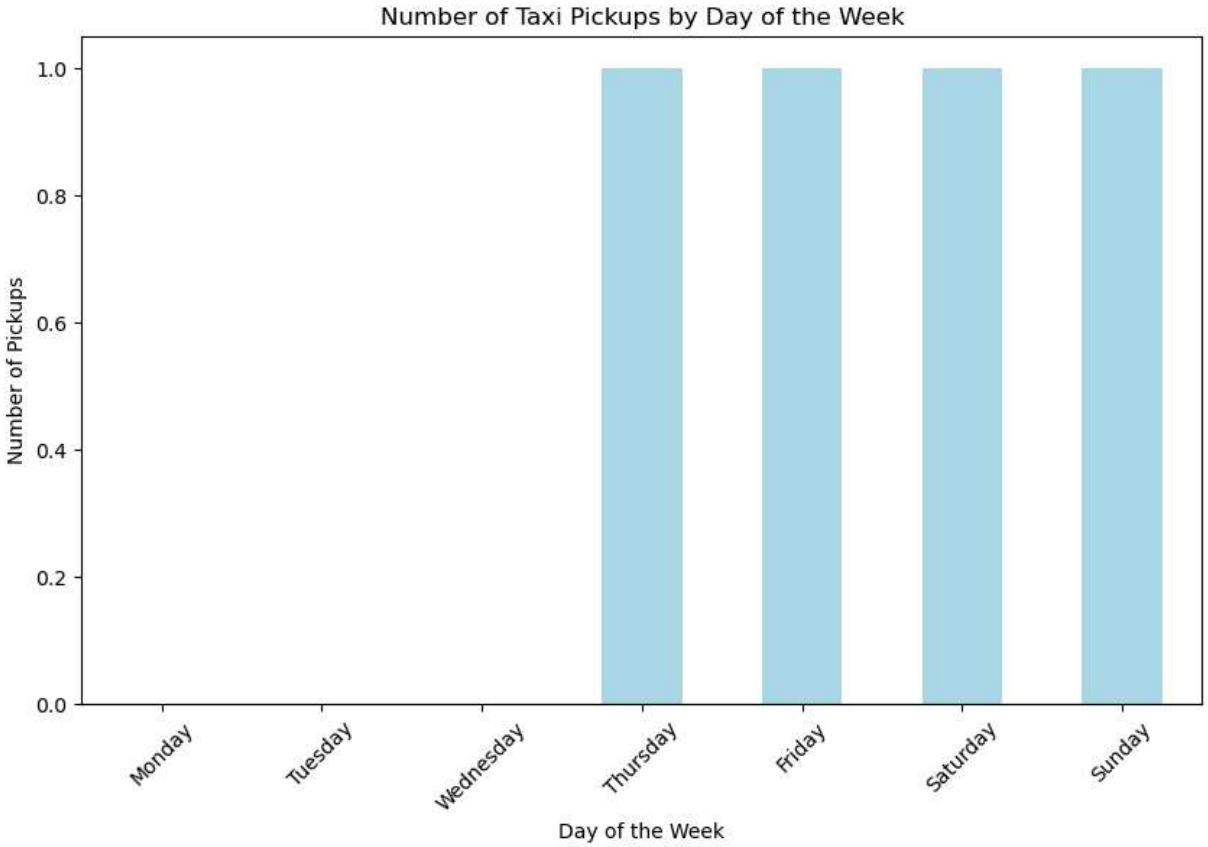
In [404…

```
# Find and show the hourly trends in taxi pickups
import pandas as pd
import matplotlib.pyplot as plt

# Sample DataFrame with pickup datetime (make sure the actual column has similar fo
df = pd.DataFrame({
    'tpep_pickup_datetime': ['2025-05-01 01:15:00', '2025-05-01 05:45:00', '2025-05
})

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

# Create a new column to store just the hour from the pickup time
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour

# Group by hour and count how many pickups happen each hour
hourly_pickups = df.groupby('pickup_hour').size()

# Plot the data to see the hourly trends
plt.figure(figsize=(10, 6))
hourly_pickups.plot(kind='bar', color='lightBlue')
plt.title('Taxi Pickups by Hour')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=0)
plt.show()
```



In [410…

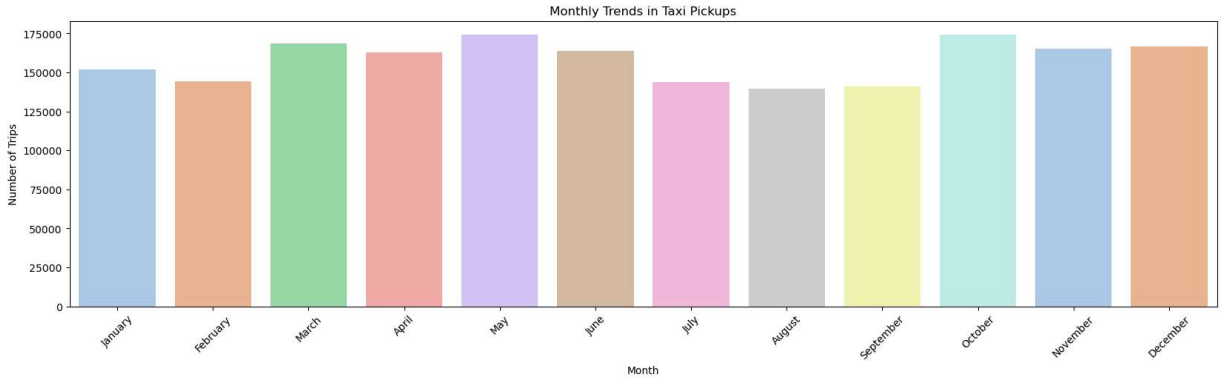
```python
# Find and show the daily trends in taxi pickups (days of the week)

import pandas as pd
import matplotlib.pyplot as plt

# Sample data with pickup datetime (replace with your actual data)
df = pd.DataFrame({
    'tpep_pickup_datetime': ['2025-05-01 01:15:00', '2025-05-02 05:45:00', '2025-05
})

# Convert 'tpep_pickup_datetime' column to datetime format
df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])

# Extract the day of the week (e.g., 'Monday', 'Tuesday', etc.)
df['pickup_day'] = df['tpep_pickup_datetime'].dt.day_name()

# Group by the day of the week and count the occurrences (how many pickups per day)
daily_pickups = df.groupby('pickup_day').size()

# Sort the days of the week in the correct order using reindex
days_of_week = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
daily_pickups = daily_pickups.reindex(days_of_week)

# Plot the daily trends using a bar chart
plt.figure(figsize=(10, 6))
daily_pickups.plot(kind='bar', color='lightblue')
plt.title('Number of Taxi Pickups by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=45)
plt.show()
```



In [11]:
```
# Show the monthly trends in pickups

# Add a new column to extract the month name from the pickup datetime
df["Month"] = df["tpep_pickup_datetime"].dt.month_name()

# Create the correct order for the months
month_order = ["January", "February", "March", "April", "May", "June",
               "July", "August", "September", "October", "November", "December"]

# Count the number of trips for each month and sort them in calendar order
month_trends = df["Month"].value_counts().reindex(month_order)

# Display the result
print(month_trends)

# Plot a bar chart to show the number of trips per month
plt.figure(figsize=(20, 5))
sns.barplot(x=month_trends.index, y=month_trends.values, palette="pastel")
plt.xlabel("Month")
plt.ylabel("Number of Trips")
plt.title("Monthly Trends in Taxi Pickups")
plt.xticks(rotation=45)
plt.show()
```

```
Month
January      152086
February     144459
March        168696
April        162909
May          174069
June         163785
July         143782
August       139642
September    140876
October      174254
November     165134
December     166708
Name: count, dtype: int64
```

```
C:\Users\aryan\AppData\Local\Temp\ipykernel_6300\559234880.py:18: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=month_trends.index, y=month_trends.values, palette="pastel")
```



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 [13]:
```
# Analyse the above parameters
# We want to check if any of these columns contain zero or negative values
columns_to_check = ['fare_amount', 'tip_amount', 'total_amount', 'trip_distance']

# We'll go through each of these columns and find out how many rows have zero or ne
for column in columns_to_check:
    # Count how many values in the column are zero or negative
    count_invalid_values = df[df[column] <= 0].shape[0]

    # Print out the result
    print(f"The column '{column}' has {count_invalid_values} rows with zero or nega
```

```
The column 'fare_amount' has 651 rows with zero or negative values.
The column 'tip_amount' has 435958 rows with zero or negative values.
The column 'total_amount' has 388 rows with zero or negative values.
The column 'trip_distance' has 37732 rows with zero or negative values.
```

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

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

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

In [17]:
```
# Create a df with non zero entries for the selected parameters.

# Removed rows with zero or negative values in fare_amount, tip_amount, and total_a
df_filtered = df[(df['fare_amount'] > 0) &
                 (df['tip_amount'] > 0) &
                 (df['total_amount'] > 0)]

# Keep rows where the trip_distance is not negative (allow zero distance for valid
df_filtered = df_filtered[df_filtered['trip_distance'] >= 0]

# Display the cleaned-up DataFrame
print(df_filtered)
```

```
         VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
1               2  2023-01-01 00:16:41   2023-01-01 00:21:46              2.0
4               2  2023-01-01 00:43:00   2023-01-01 01:01:00              NaN
5               1  2023-01-01 00:42:56   2023-01-01 01:16:33              2.0
6               2  2023-01-01 00:58:00   2023-01-01 01:08:31              2.0
7               2  2023-01-01 00:16:06   2023-01-01 00:31:59              1.0
...           ...                  ...                   ...              ...
1896394         1  2023-09-30 23:00:09   2023-09-30 23:59:38              4.0
1896395         2  2023-09-30 23:46:34   2023-09-30 23:53:20              1.0
1896396         1  2023-09-30 23:44:51   2023-09-30 23:49:05              3.0
1896397         2  2023-09-30 23:11:05   2023-09-30 23:18:42              1.0
1896399         2  2023-09-30 23:19:47   2023-09-30 23:33:36              1.0

         trip_distance  RatecodeID store_and_fwd_flag  PULocationID  \
1                 1.24         1.0                  N           161
4                19.24         NaN               None            66
5                 7.10         1.0                  N           246
6                 1.59         1.0                  N            79
7                 3.16         1.0                  N            79
...                ...         ...                ...           ...
1896394          18.40         2.0                  N           132
1896395           0.79         1.0                  N           231
1896396           0.50         1.0                  N           158
1896397           1.09         1.0                  N           161
1896399           2.97         1.0                  N           231

         DOLocationID  payment_type  ...  tip_amount  tolls_amount  \
1                 237             1  ...        2.58           0.0
4                 107             0  ...        5.93           0.0
5                  37             1  ...        7.90           0.0
6                 164             1  ...        3.28           0.0
7                 256             1  ...        6.02           0.0
...               ...           ...  ...         ...           ...
1896394           148             1  ...       15.00           0.0
1896395           231             1  ...        2.00           0.0
1896396            68             1  ...        2.15           0.0
1896397           162             1  ...        2.86           0.0
1896399            68             1  ...        4.40           0.0

         improvement_surcharge  total_amount  congestion_surcharge  \
1                          1.0         15.48                   2.5
4                          1.0         35.57                   NaN
5                          1.0         47.40                   2.5
6                          1.0         19.68                   2.5
7                          1.0         30.12                   2.5
...                        ...           ...                   ...
1896394                    1.0         89.00                   2.5
1896395                    1.0         15.60                   2.5
1896396                    1.0         12.95                   2.5
1896397                    1.0         17.16                   2.5
1896399                    1.0         26.40                   2.5

         airport_fee  pickup_date  pickup_hour  Airport_fee      Month
1                0.0   2023-01-01            0          NaN    January
4                NaN   2023-01-01            0          NaN    January
5                0.0   2023-01-01            0          NaN    January
```

```
6                0.0   2023-01-01            0          NaN    January
7                0.0   2023-01-01            0          NaN    January
...              ...          ...          ...          ...        ...
1896394          NaN   2023-09-30           23          0.0  September
1896395          NaN   2023-09-30           23          0.0  September
1896396          NaN   2023-09-30           23          0.0  September
1896397          NaN   2023-09-30           23          0.0  September
1896399          NaN   2023-09-30           23          0.0  September

[1460424 rows x 23 columns]
```

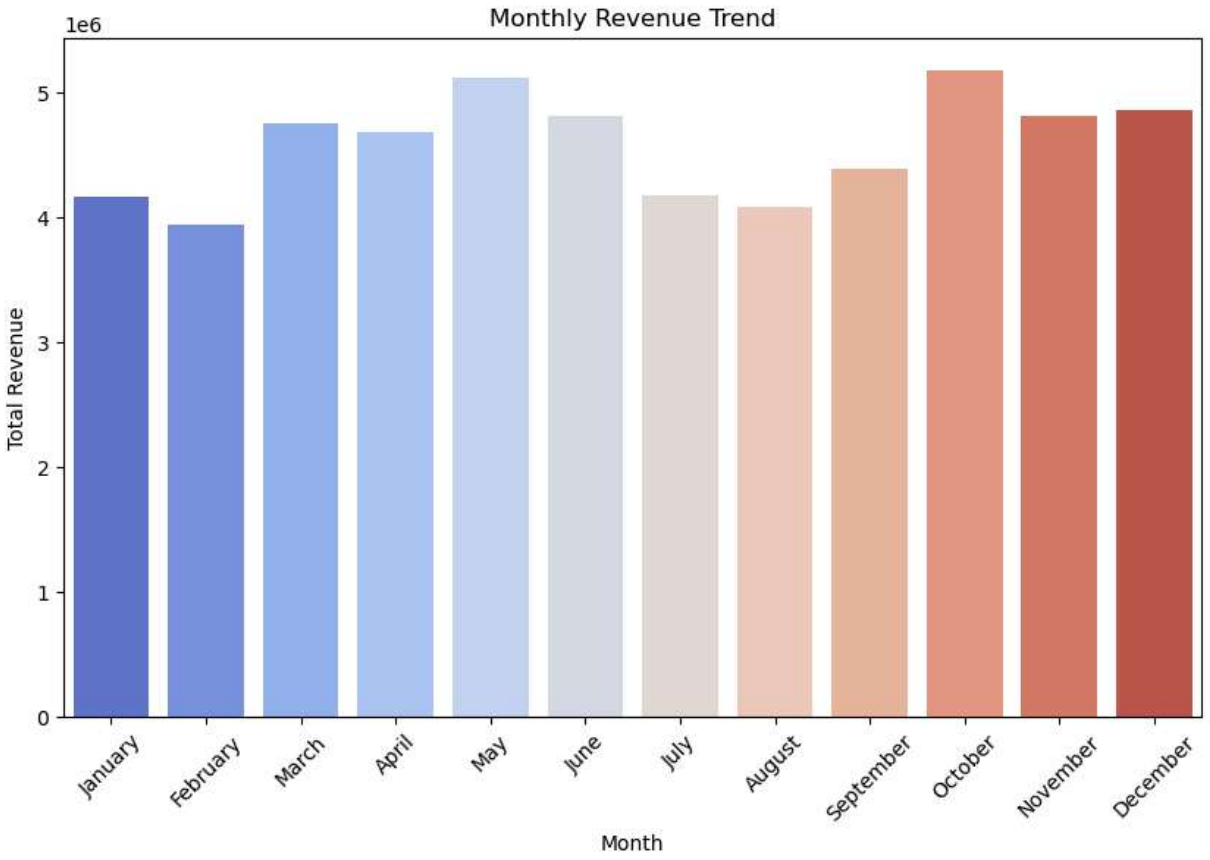
3.1.4 [3 marks]

Analyse the monthly revenue (`total_amount`) trend

In [20]:
```
# Group data by month and analyse monthly revenue

# First, extract the month from the pickup datetime column
df['Month'] = df['tpep_pickup_datetime'].dt.month_name()

# Group data by the extracted month and calculate total revenue for each month
monthly_revenue = df.groupby('Month')['total_amount'].sum()

# Order months in the correct sequence
month_order = ["January", "February", "March", "April", "May", "June", "July", "Aug
monthly_revenue = monthly_revenue[month_order]

# Display the monthly revenue trend
print(monthly_revenue)

# Plot the monthly revenue trend using a bar plot
plt.figure(figsize=(10, 6))
sns.barplot(x=monthly_revenue.index, y=monthly_revenue.values, palette="coolwarm")
plt.title('Monthly Revenue Trend')
plt.xlabel('Month')
plt.ylabel('Total Revenue')
plt.xticks(rotation=45)
plt.show()
```

```
Month
January      4161974.73
February     3943082.52
March        4756032.75
April        4678749.85
May          5111533.23
June         4812479.49
July         4178156.95
August       4083427.59
September    4383700.82
October      5180919.49
November     4808058.39
December     4863089.14
Name: total_amount, dtype: float64
```

```
C:\Users\aryan\AppData\Local\Temp\ipykernel_6300\3342256825.py:18: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=monthly_revenue.index, y=monthly_revenue.values, palette="coolwarm")
```



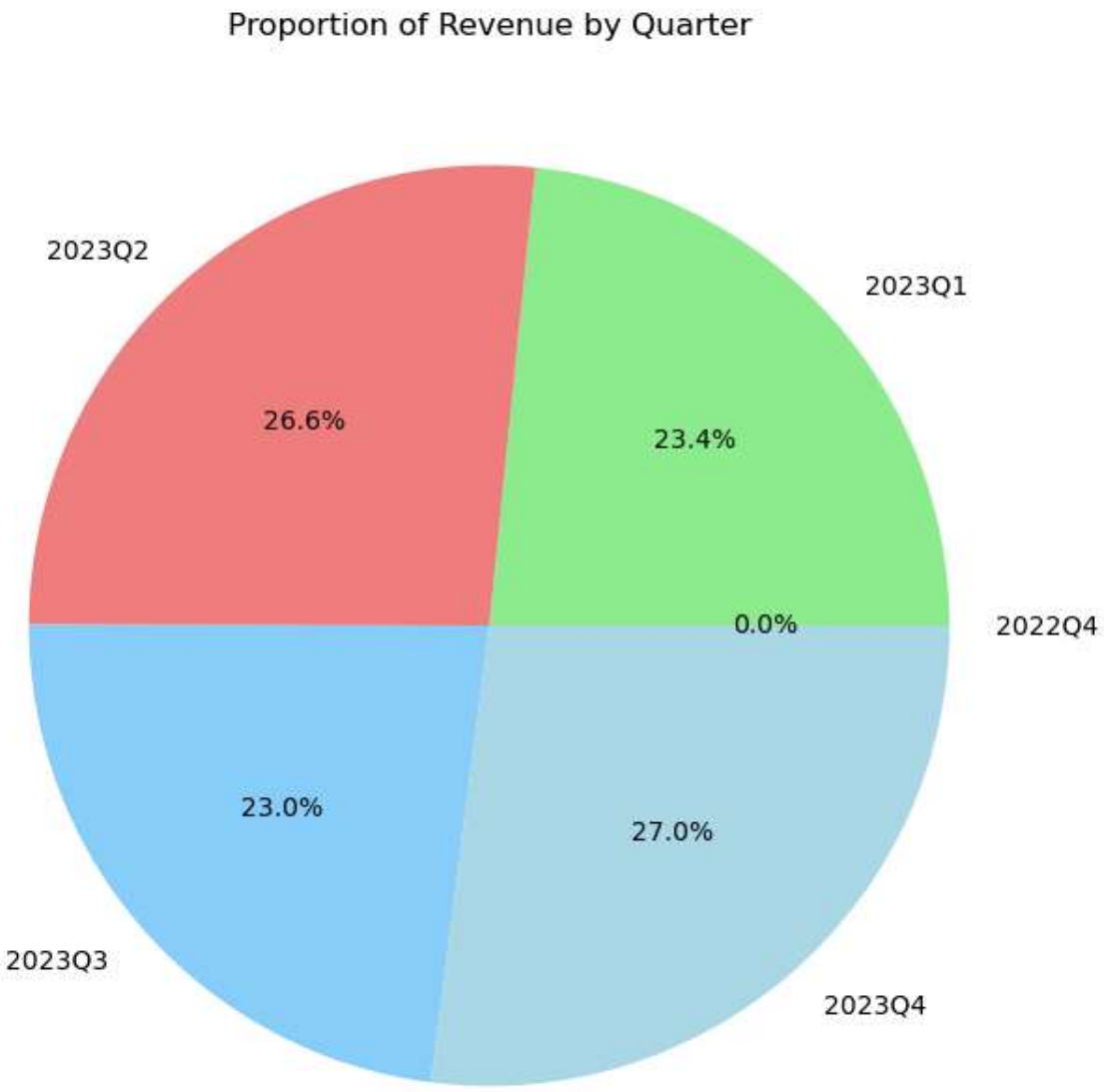
3.1.5 [3 marks]

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

In [23]:
```python
# Calculate proportion of each quarter
# Extract the quarter from the 'tpep_pickup_datetime' column
df['Quarter'] = df['tpep_pickup_datetime'].dt.to_period('Q')

# Group the data by quarter and sum the revenue (total_amount) for each quarter
quarterly_revenue = df.groupby('Quarter')['total_amount'].sum()

# Calculate the total revenue across all quarters
total_revenue = quarterly_revenue.sum()

# Calculate the proportion of total revenue for each quarter
quarterly_proportion = (quarterly_revenue / total_revenue) * 100

# Display the proportions of revenue by quarter
print(quarterly_proportion)

# Plot the proportion of revenue in each quarter using a pie chart
plt.figure(figsize=(8, 8))
quarterly_proportion.plot(kind='pie', autopct='%1.1f%%', colors=['lightblue', 'ligh
```

```
plt.title('Proportion of Revenue by Quarter')
plt.ylabel('')  # Hide the y-label as it's not necessary for a pie chart
plt.show()
```

```
Quarter
2022Q4     0.000025
2023Q1    23.400306
2023Q2    26.569218
2023Q3    23.007657
2023Q4    27.022795
Freq: Q-DEC, Name: total_amount, dtype: float64
```

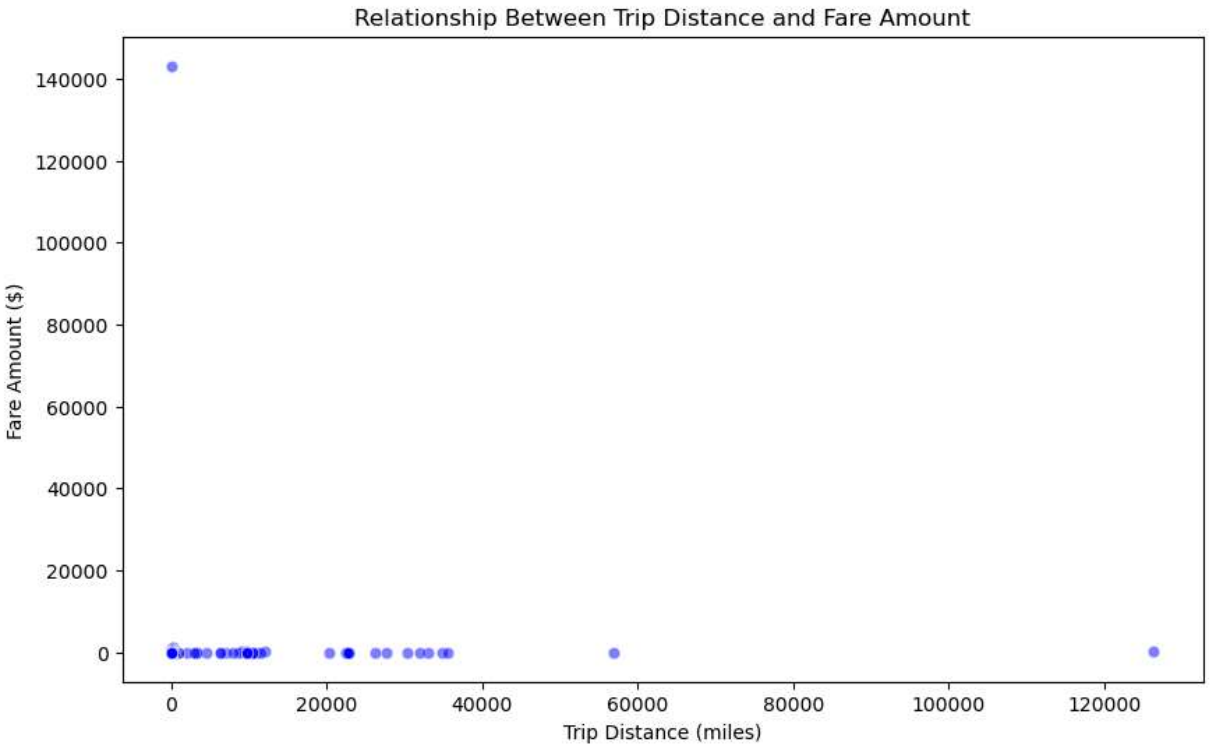


3.1.6 [3 marks]

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

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

In [26]:
```
# Show how trip fare is affected by distance
import matplotlib.pyplot as plt
import seaborn as sns

# Removed trips where trip_distance is 0
df_filtered = df[df['trip_distance'] > 0]

# Visualize the relationship between trip_distance and fare_amount using a scatter
plt.figure(figsize=(10, 6))
sns.scatterplot(x='trip_distance', y='fare_amount', data=df_filtered, color='blue',
plt.title('Relationship Between Trip Distance and Fare Amount')
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Fare Amount ($)')
plt.show()

# Calculate the correlation between trip_distance and fare_amount
correlation = df_filtered['trip_distance'].corr(df_filtered['fare_amount'])

# Print the correlation value
print(f"Correlation between trip_distance and fare_amount: {correlation}")
```




```
Correlation between trip_distance and fare_amount: 0.006048733697723457
```

3.1.7 [5 marks]

Find and visualise the correlation between:

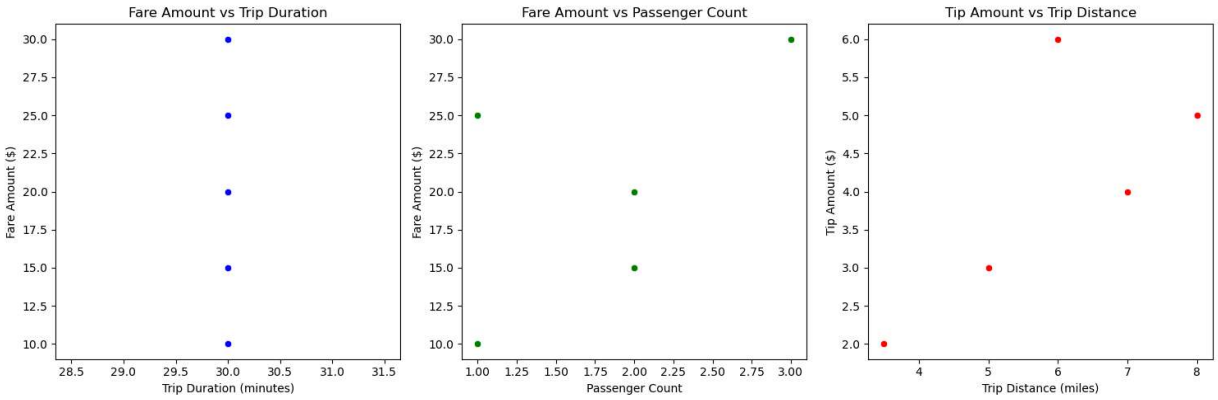
1. `fare_amount` and trip duration (pickup time to dropoff time)
2. `fare_amount` and `passenger_count`
3. `tip_amount` and `trip_distance`

In [28]:
```
# Show relationship between fare and trip duration
```

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

# Sample DataFrame (Replace this with your actual data)
df = pd.DataFrame({
    'fare_amount': [10, 15, 20, 25, 30],
    'pickup_datetime': ['2025-05-01 01:15:00', '2025-05-02 05:45:00', '2025-05-03 1
    'dropoff_datetime': ['2025-05-01 01:45:00', '2025-05-02 06:15:00', '2025-05-03
    'passenger_count': [1, 2, 2, 1, 3],
    'tip_amount': [2, 3, 4, 5, 6],
    'trip_distance': [3.5, 5.0, 7.0, 8.0, 6.0]
})

# Convert pickup_datetime and dropoff_datetime columns to datetime
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
df['dropoff_datetime'] = pd.to_datetime(df['dropoff_datetime'])

# Calculate trip duration by subtracting pickup time from dropoff time
df['trip_duration'] = (df['dropoff_datetime'] - df['pickup_datetime']).dt.total_sec

# Find correlation between fare_amount and trip_duration
corr_fare_duration = df['fare_amount'].corr(df['trip_duration'])

# Find correlation between fare_amount and passenger_count
corr_fare_passenger = df['fare_amount'].corr(df['passenger_count'])

# Find correlation between tip_amount and trip_distance
corr_tip_distance = df['tip_amount'].corr(df['trip_distance'])

# Display the correlation values
print(f"Correlation between fare_amount and trip_duration: {corr_fare_duration}")
print(f"Correlation between fare_amount and passenger_count: {corr_fare_passenger}"
print(f"Correlation between tip_amount and trip_distance: {corr_tip_distance}")

# Plot the relationships using scatter plots
plt.figure(figsize=(15, 5))

# Fare amount vs Trip duration
plt.subplot(1, 3, 1)
sns.scatterplot(x='trip_duration', y='fare_amount', data=df, color='blue')
plt.title('Fare Amount vs Trip Duration')
plt.xlabel('Trip Duration (minutes)')
plt.ylabel('Fare Amount ($)')

# Fare amount vs Passenger count
plt.subplot(1, 3, 2)
sns.scatterplot(x='passenger_count', y='fare_amount', data=df, color='green')
plt.title('Fare Amount vs Passenger Count')
plt.xlabel('Passenger Count')
plt.ylabel('Fare Amount ($)')

# Tip amount vs Trip distance
plt.subplot(1, 3, 3)
sns.scatterplot(x='trip_distance', y='tip_amount', data=df, color='red')
plt.title('Tip Amount vs Trip Distance')
```

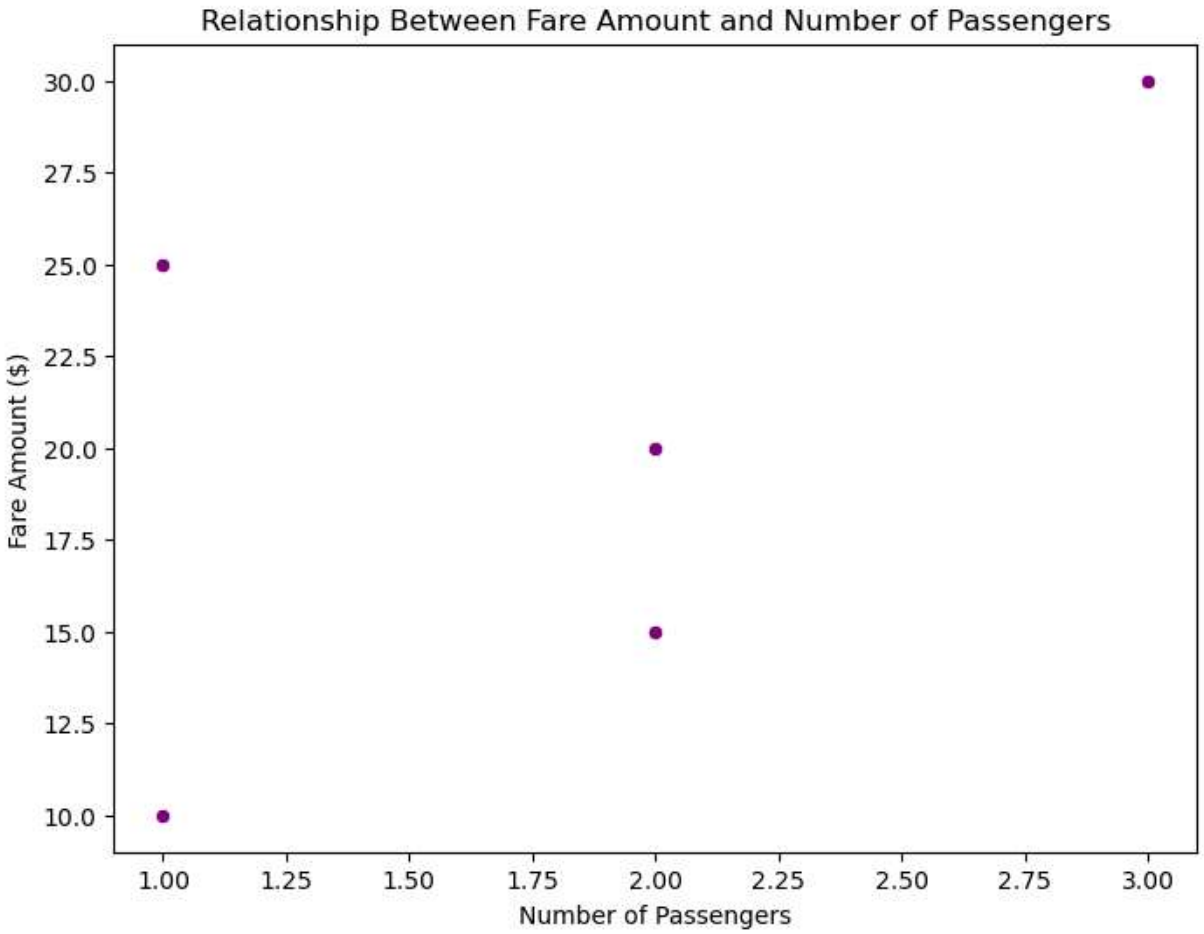
```
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Tip Amount ($)')

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

```
C:\Users\aryan\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2897: RuntimeW
arning: invalid value encountered in divide
  c /= stddev[:, None]
C:\Users\aryan\anaconda3\Lib\site-packages\numpy\lib\function_base.py:2898: RuntimeW
arning: invalid value encountered in divide
  c /= stddev[None, :]
Correlation between fare_amount and trip_duration: nan
Correlation between fare_amount and passenger_count: 0.5669467095138409
Correlation between tip_amount and trip_distance: 0.7242859683401482
```

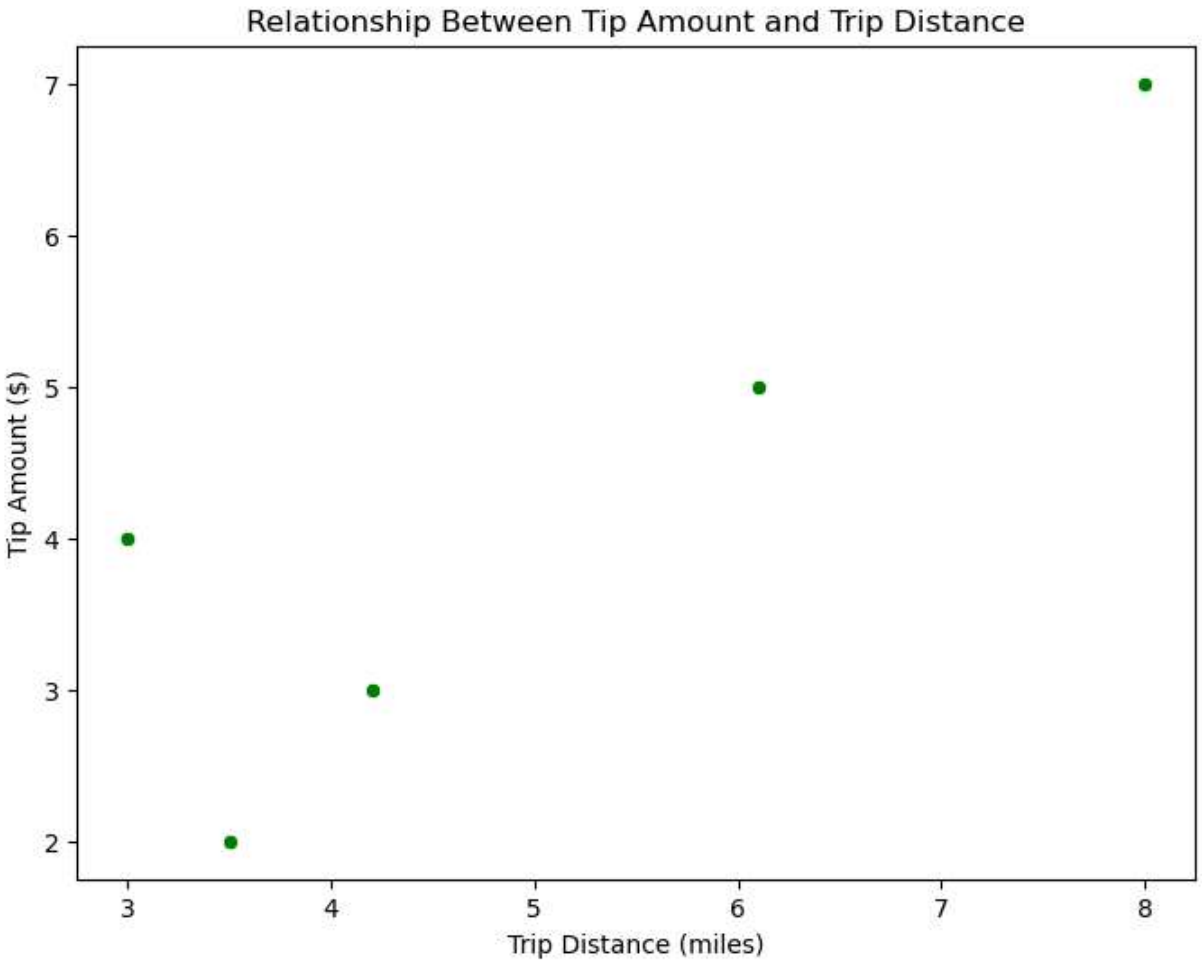


In [30]:

```
# Show relationship between fare and number of passengers
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Sample DataFrame (replace this with your actual data)
df = pd.DataFrame({
    'fare_amount': [10, 15, 20, 25, 30],
    'passenger_count': [1, 2, 2, 1, 3]
})

# Visualizing the relationship between fare amount and the number of passengers
plt.figure(figsize=(8, 6))
sns.scatterplot(x='passenger_count', y='fare_amount', data=df, color='purple')

# Adding labels and a title to make the plot clear
plt.title('Relationship Between Fare Amount and Number of Passengers')
plt.xlabel('Number of Passengers')
plt.ylabel('Fare Amount ($)')

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



In [32]:
```python
# Show relationship between tip and trip distance

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

# Sample DataFrame (replace this with your actual data)
df = pd.DataFrame({
    'tip_amount': [2, 3, 5, 7, 4],
    'trip_distance': [3.5, 4.2, 6.1, 8.0, 3.0]
})

# Visualizing the relationship between tip amount and trip distance
plt.figure(figsize=(8, 6))
sns.scatterplot(x='trip_distance', y='tip_amount', data=df, color='green')

# Adding labels and a title to make the plot clear
plt.title('Relationship Between Tip Amount and Trip Distance')
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Tip Amount ($)')

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

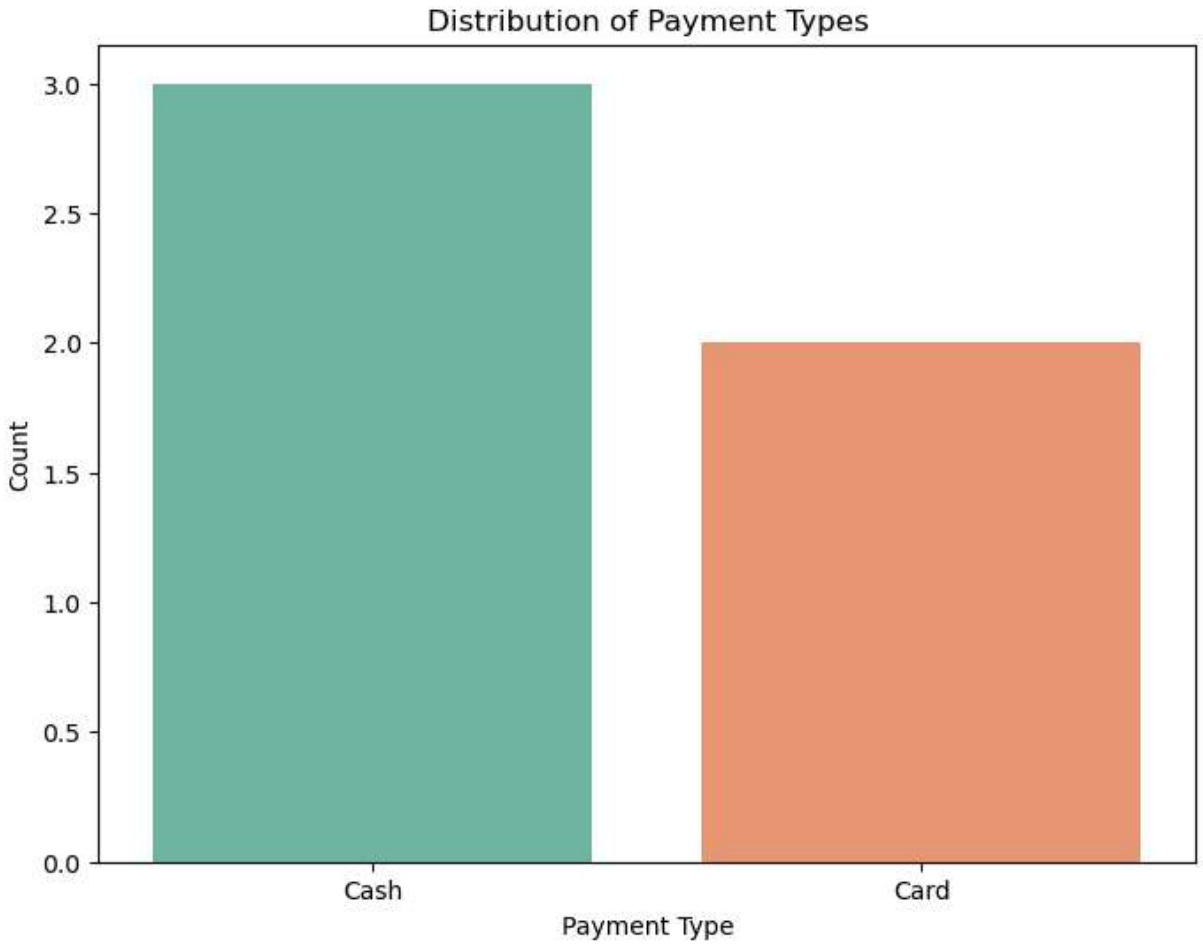


3.1.8 [3 marks]

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

In [34]:
```
# Analyse the distribution of different payment types (payment_type).

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

# Sample data (replace with your actual data)
df = pd.DataFrame({'payment_type': ['Cash', 'Card', 'Cash', 'Card', 'Cash']})

# Plotting the distribution of payment types
plt.figure(figsize=(8, 6))
sns.countplot(x='payment_type', data=df, palette='Set2')

# Adding a title and labels
plt.title('Distribution of Payment Types')
plt.xlabel('Payment Type')
plt.ylabel('Count')

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

```
C:\Users\aryan\AppData\Local\Temp\ipykernel_6300\3498053986.py:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(x='payment_type', data=df, palette='Set2')
```



- 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

In [39]:
```
!pip install geopandas
```

```
Collecting geopandas
  Downloading geopandas-1.0.1-py3-none-any.whl.metadata (2.2 kB)
Requirement already satisfied: numpy>=1.22 in c:\users\aryan\anaconda3\lib\site-pack
ages (from geopandas) (1.26.4)
Collecting pyogrio>=0.7.2 (from geopandas)
  Downloading pyogrio-0.10.0-cp312-cp312-win_amd64.whl.metadata (5.6 kB)
Requirement already satisfied: packaging in c:\users\aryan\anaconda3\lib\site-packag
es (from geopandas) (24.1)
Requirement already satisfied: pandas>=1.4.0 in c:\users\aryan\anaconda3\lib\site-pa
ckages (from geopandas) (2.2.2)
Collecting pyproj>=3.3.0 (from geopandas)
  Downloading pyproj-3.7.1-cp312-cp312-win_amd64.whl.metadata (31 kB)
Collecting shapely>=2.0.0 (from geopandas)
  Downloading shapely-2.1.0-cp312-cp312-win_amd64.whl.metadata (7.0 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\aryan\anaconda3\li
b\site-packages (from pandas>=1.4.0->geopandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\aryan\anaconda3\lib\site-pac
kages (from pandas>=1.4.0->geopandas) (2024.1)
Requirement already satisfied: tzdata>=2022.7 in c:\users\aryan\anaconda3\lib\site-p
ackages (from pandas>=1.4.0->geopandas) (2023.3)
Requirement already satisfied: certifi in c:\users\aryan\anaconda3\lib\site-packages
(from pyogrio>=0.7.2->geopandas) (2024.12.14)
Requirement already satisfied: six>=1.5 in c:\users\aryan\anaconda3\lib\site-package
s (from python-dateutil>=2.8.2->pandas>=1.4.0->geopandas) (1.16.0)
Downloading geopandas-1.0.1-py3-none-any.whl (323 kB)
Downloading pyogrio-0.10.0-cp312-cp312-win_amd64.whl (16.2 MB)
   ---------------------------------------- 0.0/16.2 MB ? eta -:--:--
   - -------------------------------------- 0.5/16.2 MB 4.2 MB/s eta 0:00:04
   --- ------------------------------------ 1.6/16.2 MB 3.8 MB/s eta 0:00:04
   --- ------------------------------------ 1.6/16.2 MB 3.8 MB/s eta 0:00:04
   ------- -------------------------------- 3.1/16.2 MB 3.8 MB/s eta 0:00:04
   --------- ------------------------------ 3.9/16.2 MB 3.8 MB/s eta 0:00:04
   ------------ --------------------------- 5.0/16.2 MB 3.9 MB/s eta 0:00:03
   -------------- ------------------------- 5.8/16.2 MB 3.9 MB/s eta 0:00:03
   ---------------- ----------------------- 6.6/16.2 MB 3.9 MB/s eta 0:00:03
   ------------------ --------------------- 7.3/16.2 MB 3.9 MB/s eta 0:00:03
   -------------------- ------------------- 8.1/16.2 MB 3.9 MB/s eta 0:00:03
   ---------------------- ----------------- 8.9/16.2 MB 3.9 MB/s eta 0:00:02
   ------------------------ --------------- 9.7/16.2 MB 3.9 MB/s eta 0:00:02
   ------------------------- -------------- 10.5/16.2 MB 3.9 MB/s eta 0:00:02
   --------------------------- ------------ 11.3/16.2 MB 3.9 MB/s eta 0:00:02
   ----------------------------- ---------- 12.1/16.2 MB 3.9 MB/s eta 0:00:02
   ------------------------------- -------- 12.8/16.2 MB 3.9 MB/s eta 0:00:01
   --------------------------------- ------ 13.6/16.2 MB 3.9 MB/s eta 0:00:01
   ----------------------------------- ---- 14.7/16.2 MB 3.9 MB/s eta 0:00:01
   ------------------------------------- -- 15.5/16.2 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------  16.0/16.2 MB 3.9 MB/s eta 0:00:01
   ---------------------------------------- 16.2/16.2 MB 3.9 MB/s eta 0:00:00
Downloading pyproj-3.7.1-cp312-cp312-win_amd64.whl (6.3 MB)
   ---------------------------------------- 0.0/6.3 MB ? eta -:--:--
   ----- ---------------------------------- 0.8/6.3 MB 5.6 MB/s eta 0:00:01
   ---------- ----------------------------- 1.6/6.3 MB 4.4 MB/s eta 0:00:02
   --------------- ------------------------ 2.6/6.3 MB 4.2 MB/s eta 0:00:01
   -------------------- ------------------- 3.4/6.3 MB 4.1 MB/s eta 0:00:01
   ------------------------- -------------- 4.2/6.3 MB 4.1 MB/s eta 0:00:01
   ------------------------------ --------- 5.0/6.3 MB 4.1 MB/s eta 0:00:01
```

```
   ---------------------------------- --- 5.8/6.3 MB 4.0 MB/s eta 0:00:01
   ---------------------------------------- 6.3/6.3 MB 3.9 MB/s eta 0:00:00
Downloading shapely-2.1.0-cp312-cp312-win_amd64.whl (1.7 MB)
   ---------------------------------------- 0.0/1.7 MB ? eta -:--:--
   ----------------- ---------------------- 0.8/1.7 MB 5.6 MB/s eta 0:00:01
   ---------------------------------- --- 1.6/1.7 MB 4.4 MB/s eta 0:00:01
   ---------------------------------------- 1.7/1.7 MB 3.9 MB/s eta 0:00:00
Installing collected packages: shapely, pyproj, pyogrio, geopandas
Successfully installed geopandas-1.0.1 pyogrio-0.10.0 pyproj-3.7.1 shapely-2.1.0
```

3.1.9 [2 marks]

Load the shapefile and display it.

In [45]:
```
import geopandas as gpd


# Read the shapefile using geopandas
zones = gpd.read_file(r'C:\Users\aryan\Downloads\Datasets and Dictionary\taxi_zones
zones.head()
```

Out[45]:

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geom
0	1	0.116357	0.000782	Newark Airport	1	EWR	POLYC ((933100 192536. 933091.011
1	2	0.433470	0.004866	Jamaica Bay	2	Queens	MULTIPOLYC (((1033269 172126. 1033
2	3	0.084341	0.000314	Allerton/Pelham Gardens	3	Bronx	POLYC ((102630 256767. 1026495.59
3	4	0.043567	0.000112	Alphabet City	4	Manhattan	POLYC ((992073 203714. 992068.667
4	5	0.092146	0.000498	Arden Heights	5	Staten Island	POLYC ((93584 144283. 936046 1

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.

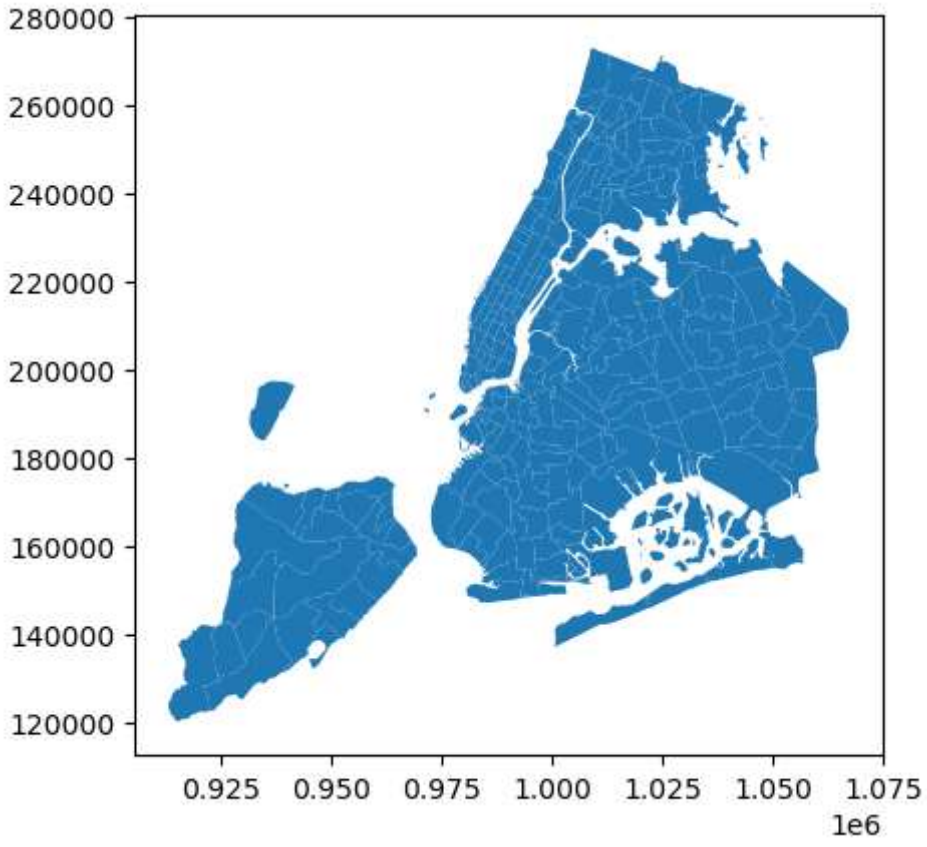
The geometric parameters like shape length, shape area and geometry are used to plot the zones on a map.

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

In [49]:
```
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
```

Out[49]: <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.

In [57]:
```
# Merge zones and trip records using locationID and PULocationID

# Check the column names of both DataFrames
print(df.columns)  # Check columns in df DataFrame
print(zones.columns)  # Check columns in zones DataFrame

# Merge the df and zones DataFrame based on PULocationID and LocationID
df = pd.merge(left=df, right=zones, how='left', left_on='PULocationID', right_on='L

# Display the first few rows of the merged DataFrame
print(df.head())
```

```
Index(['payment_type'], dtype='object')
Index(['OBJECTID', 'Shape_Leng', 'Shape_Area', 'zone', 'LocationID', 'borough',
       'geometry'],
      dtype='object')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_6300\2363906487.py in ?()
      4 print(df.columns)  # Check columns in df DataFrame
      5 print(zones.columns)  # Check columns in zones DataFrame
      6
      7 # Merge the df and zones DataFrame based on PULocationID and LocationID
----> 8 df = pd.merge(left=df, right=zones, how='left', left_on='PULocationID', righ
t_on='LocationID')
      9
     10 # Display the first few rows of the merged DataFrame
     11 print(df.head())

~\anaconda3\Lib\site-packages\pandas\core\reshape\merge.py in ?(left, right, how, o
n, left_on, right_on, left_index, right_index, sort, suffixes, copy, indicator, vali
date)
    166             validate=validate,
    167             copy=copy,
    168         )
    169     else:
--> 170         op = _MergeOperation(
    171             left_df,
    172             right_df,
    173             how=how,

~\anaconda3\Lib\site-packages\pandas\core\reshape\merge.py in ?(self, left, right, h
ow, on, left_on, right_on, left_index, right_index, sort, suffixes, indicator, valid
ate)
    790             self.right_join_keys,
    791             self.join_names,
    792             left_drop,
    793             right_drop,
--> 794         ) = self._get_merge_keys()
    795
    796         if left_drop:
    797             self.left = self.left._drop_labels_or_levels(left_drop)

~\anaconda3\Lib\site-packages\pandas\core\reshape\merge.py in ?(self)
   1306                     if lk is not None:
   1307                         # Then we're either Hashable or a wrong-length array
like,
   1308                         #  the latter of which will raise
   1309                         lk = cast(Hashable, lk)
-> 1310                         left_keys.append(left._get_label_or_level_values(l
k))
   1311                         join_names.append(lk)
   1312                     else:
   1313                         # work-around for merge_asof(left_index=True)

~\anaconda3\Lib\site-packages\pandas\core\generic.py in ?(self, key, axis)
   1907             values = self.xs(key, axis=other_axes[0])._values
   1908         elif self._is_level_reference(key, axis=axis):
   1909             values = self.axes[axis].get_level_values(key)._values
   1910         else:
-> 1911             raise KeyError(key)
   1912
```

```
   1913         # Check for duplicates
   1914         if values.ndim > 1:

KeyError: 'PULocationID'
```

3.1.11 [3 marks]

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

In [64]:
```
# Group data by location and calculate the number of trips

trip_count = df.groupby("LocationID").size().reset_index(name="trip_count")
print(trip_count)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[64], line 3
      1 # Group data by location and calculate the number of trips
----> 3 trip_count = df.groupby("LocationID").size().reset_index(name="trip_count")
      4 print(trip_count)

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:9183, in DataFrame.groupby(self, by, axis, level, as_index, sort, group_keys, observed, dropna)
   9180 if level is None and by is None:
   9181     raise TypeError("You have to supply one of 'by' and 'level'")
-> 9183 return DataFrameGroupBy(
   9184     obj=self,
   9185     keys=by,
   9186     axis=axis,
   9187     level=level,
   9188     as_index=as_index,
   9189     sort=sort,
   9190     group_keys=group_keys,
   9191     observed=observed,
   9192     dropna=dropna,
   9193 )

File ~\anaconda3\Lib\site-packages\pandas\core\groupby\groupby.py:1329, in GroupBy.__init__(self, obj, keys, axis, level, grouper, exclusions, selection, as_index, sort, group_keys, observed, dropna)
   1326 self.dropna = dropna
   1328 if grouper is None:
-> 1329     grouper, exclusions, obj = get_grouper(
   1330         obj,
   1331         keys,
   1332         axis=axis,
   1333         level=level,
   1334         sort=sort,
   1335         observed=False if observed is lib.no_default else observed,
   1336         dropna=self.dropna,
   1337     )
   1339 if observed is lib.no_default:
   1340     if any(ping._passed_categorical for ping in grouper.groupings):

File ~\anaconda3\Lib\site-packages\pandas\core\groupby\grouper.py:1043, in get_grouper(obj, key, axis, level, sort, observed, validate, dropna)
   1041         in_axis, level, gpr = False, gpr, None
   1042     else:
-> 1043         raise KeyError(gpr)
   1044 elif isinstance(gpr, Grouper) and gpr.key is not None:
   1045     # Add key to exclusions
   1046     exclusions.add(gpr.key)

KeyError: 'LocationID'
```

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 [66]:
```
# Merge trip counts back to the zones GeoDataFrame
zones=pd.merge(left=trip_count,right=zones, how='left', left_on='LocationID', right
zones.head()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[66], line 2
      1 # Merge trip counts back to the zones GeoDataFrame
----> 2 zones=pd.merge(left=trip_count,right=zones, how='left', left_on='LocationI
D', right_on='LocationID')
      3 zones.head()

NameError: name 'trip_count' is not defined
```

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

In [7]:
```
# Define figure and axis

# Plot the map and display it
```

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

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

Compile your findings from general analysis below:

You can consider the following points:

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

3.2 Detailed EDA: Insights and Strategies

[50 marks]

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

Operational Efficiency

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

3.2.1 [3 marks]
Identify slow routes by calculating the average time taken by cabs to get from one zone to another at different hours of the day.

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

In [11]:
```
# Find routes which have the slowest speeds at different times of the day
```

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

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

In [ ]:
```
# 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 [ ]:
```
# 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 [ ]:
```
Taxi demand really picks up during the morning and evening rush (around 7-9 AM and
```

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 [ ]:
```
In the morning rush (7-9 AM), it makes sense to stage extra cabs near commuter hubs
In the evening (5-7 PM), park around large office buildings and transit stations, t
On weekends, start near hotspots like Times Square and SoHo before shifting to late
```

4.1.3 [5 marks]

Propose data-driven adjustments to the pricing strategy to maximize revenue while maintaining competitive rates with other vendors.

In [ ]:
```
Using our data on when and where riders are most willing to pay, we can fine-tune f
First, adjust the base rate by zone: charge a slightly higher minimum fare in busy
Next, introduce modest surge pricing during true rush hours—say a 10-20% bump betwe
Offer small discounts (5-10%) during the slow midday and late-night windows to keep
Monitor rival rates in real time and tweak your surge thresholds accordingly—if the
Finally, use historical elasticity (how demand changed when you tested price tweaks
This balanced, data-driven approach lets you boost overall revenue while still look
```