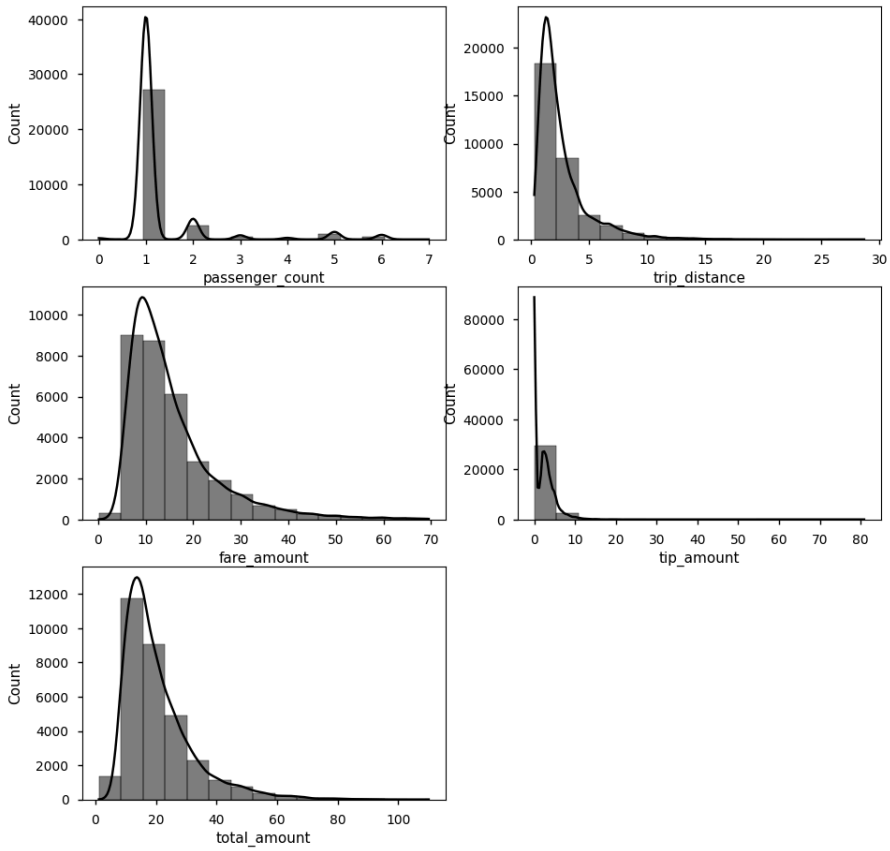
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')

plt.style.use(style='seaborn-v0_8-notebook')


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

file_path = '/content/drive/My Drive/Final_Data.csv'


data = pd.read_csv(file_path)
print(data)
```

```
     32040         2           134     Queens          Kew Gardens      Boro Zone
     32041         2           129     Queens      Jackson Heights      Boro Zone
     32042         2           129     Queens      Jackson Heights      Boro Zone
     32043         2            75  Manhattan    East Harlem South      Boro Zone

            DOLocationID  DOBorough                DOZone DOservice_zone  \
     0               143  Manhattan    Lincoln Square West    Yellow Zone
     1                43  Manhattan           Central Park    Yellow Zone
     2               238  Manhattan  Upper West Side North    Yellow Zone
     3                74  Manhattan      East Harlem North      Boro Zone
     4               262  Manhattan         Yorkville East    Yellow Zone
     ...             ...        ...                    ...           ...
     32039           129     Queens        Jackson Heights      Boro Zone
     32040           197     Queens          Richmond Hill      Boro Zone
     32041           260     Queens               Woodside      Boro Zone
     32042           260     Queens               Woodside      Boro Zone
     32043            51      Bronx            Co-Op City      Boro Zone

            passenger_count  ...  trip_type  congestion_surcharge  Dropoff_year  \
     0                    1  ...          1                  2.75          2023
     1                    1  ...          1                  0.00          2023
     2                    1  ...          1                  0.00          2023
     3                    1  ...          1                  0.00          2023
     4                    1  ...          1                  2.75          2023
     ...                ...  ...        ...                   ...           ...
     32039                1  ...          1                  0.00          2023
     32040                1  ...          1                  0.00          2023
     32041                2  ...          1                  0.00          2023
     32042                2  ...          1                  0.00          2023
     32043                5  ...          1                  0.00          2023

            Dropoff_day  Dropoff_time  Pickup_year  Pickup_day  Pickup_time  \
     0                1       0:37:00         2023           1      0:26:00
     1                1       0:57:00         2023           1      0:51:00
     2                1       0:19:00         2023           1      0:13:00
     3                1       0:39:00         2023           1      0:33:00
     4                1       1:11:00         2023           1      0:53:00
     ...            ...           ...          ...         ...          ...
     32039           31      22:58:00         2023          31     22:51:00
     32040           31      23:52:00         2023          31     23:43:00
     32041           31      23:18:00         2023          31     23:05:00
     32042           31      23:40:00         2023          31     23:37:00
     32043           31      23:53:00         2023          31     23:34:00

            Day_Of_Ride_char  Trip_Duration_minutes
     0                Monday                     11
     1                Monday                     71
     2                Monday                    131
     3                Monday                    191
     4                Monday                    251
     ...                 ...                    ...
     32039         Wednesday                    491
     32040         Wednesday                    551
     32041         Wednesday                    611
     32042         Wednesday                    671
     32043         Wednesday                    731

     [32044 rows x 29 columns]
```

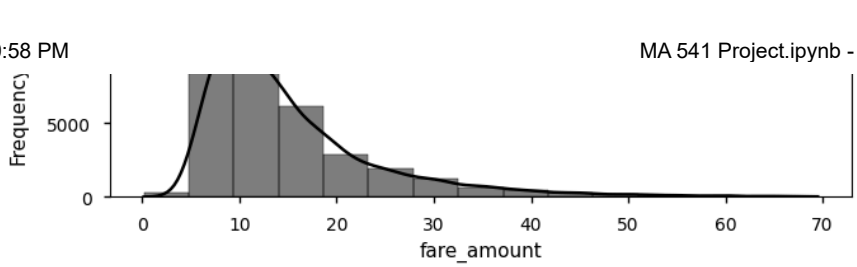
```
data.info()
```

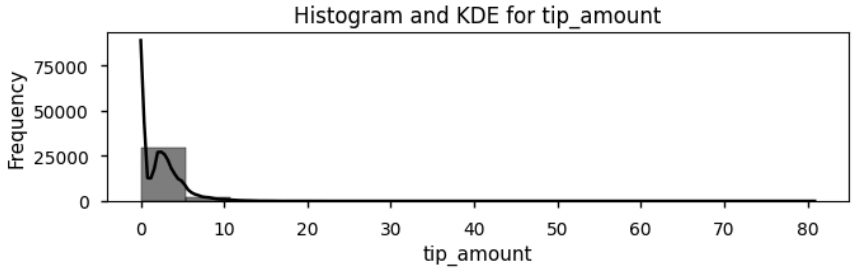
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32044 entries, 0 to 32043
Data columns (total 29 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   VendorID               32044 non-null  int64  
 1   PULocationID           32044 non-null  int64  
 2   PUBorough              32044 non-null  object 
 3   PUZone                 32044 non-null  object 
 4   PUservice_zone         32044 non-null  object 
 5   DOLocationID           32044 non-null  int64  
 6   DOBorough              32044 non-null  object 
 7   DOZone                 32044 non-null  object 
 8   DOservice_zone         32044 non-null  object 
 9   passenger_count        32044 non-null  int64  
 10  trip_distance          32044 non-null  float64
 11  fare_amount            32044 non-null  float64
 12  extra                  32044 non-null  float64
 13  mta_tax                32044 non-null  float64
 14  tip_amount             32044 non-null  float64
 15  tolls_amount           32044 non-null  float64
 16  improvement_surcharge  32044 non-null  float64
 17  total_amount           32044 non-null  float64
 18  payment_type           32044 non-null  int64  
 19  trip_type              32044 non-null  int64  
 20  congestion_surcharge   32044 non-null  float64
 21  Dropoff_year           32044 non-null  int64  
 22  Dropoff_day            32044 non-null  int64  
 23  Dropoff_time           32044 non-null  object 
 24  Pickup_year            32044 non-null  int64  
 25  Pickup_day             32044 non-null  int64  
 26  Pickup_time            32044 non-null  object 
 27  Day_Of_Ride_char       32044 non-null  object 
 28  Trip_Duration_minutes  32044 non-null  int64  
dtypes: float64(9), int64(11), object(9)
memory usage: 7.1+ MB
```

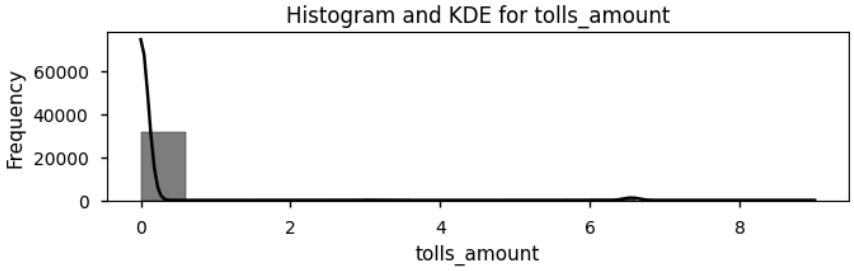
```
columns = ['passenger_count', 'trip_distance', 'fare_amount', 'tip_amount', 'total_amount']
plt.figure(figsize=(11, 11))
for i,col in enumerate(columns):
    plt.subplot(3, 2, i+1)
    sns.histplot(data = data, x = col, kde = True, bins = 15, color = 'black')
plt.show()
```

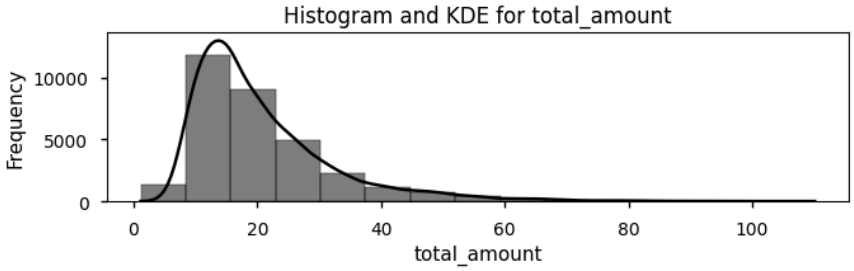

Count
passenger_count
trip_distance
fare_amount
tip_amount
total_amount

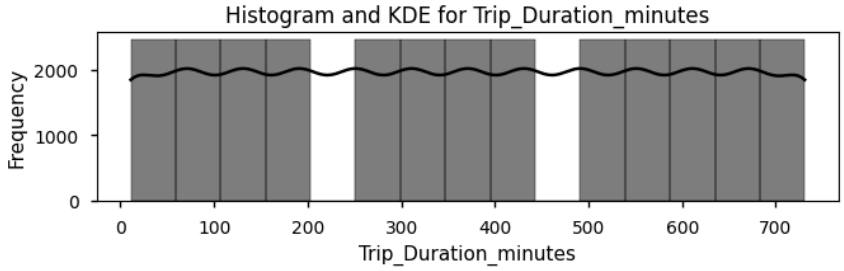
```
# Define the columns you want to analyze
columns = ['PUBorough','DOBorough','passenger_count', 'trip_distance', 'fare_amount', 'tip_amount', 'tolls_amount', 'total_amount','Trip_Dur

# Create a figure for the subplots
plt.figure(figsize=(7, 20))  # Adjust the figure size as needed

# Iterate through each column
for i, col in enumerate(columns):
    # Create a subplot for each column
    plt.subplot(len(columns), 1, i + 1)

    # Plot a histogram with a KDE plot overlay for each column
    sns.histplot(data[col], kde=True, bins=15, color='black')

    # Add a title to each subplot
    plt.title(f'Histogram and KDE for {col}')

    # Add labels for x and y axes
    plt.xlabel(col)
    plt.ylabel('Frequency')

# Adjust the layout for better spacing
plt.tight_layout()

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


Frequency
5000
0
0
10
20
30
40
50
60
70
fare_amount

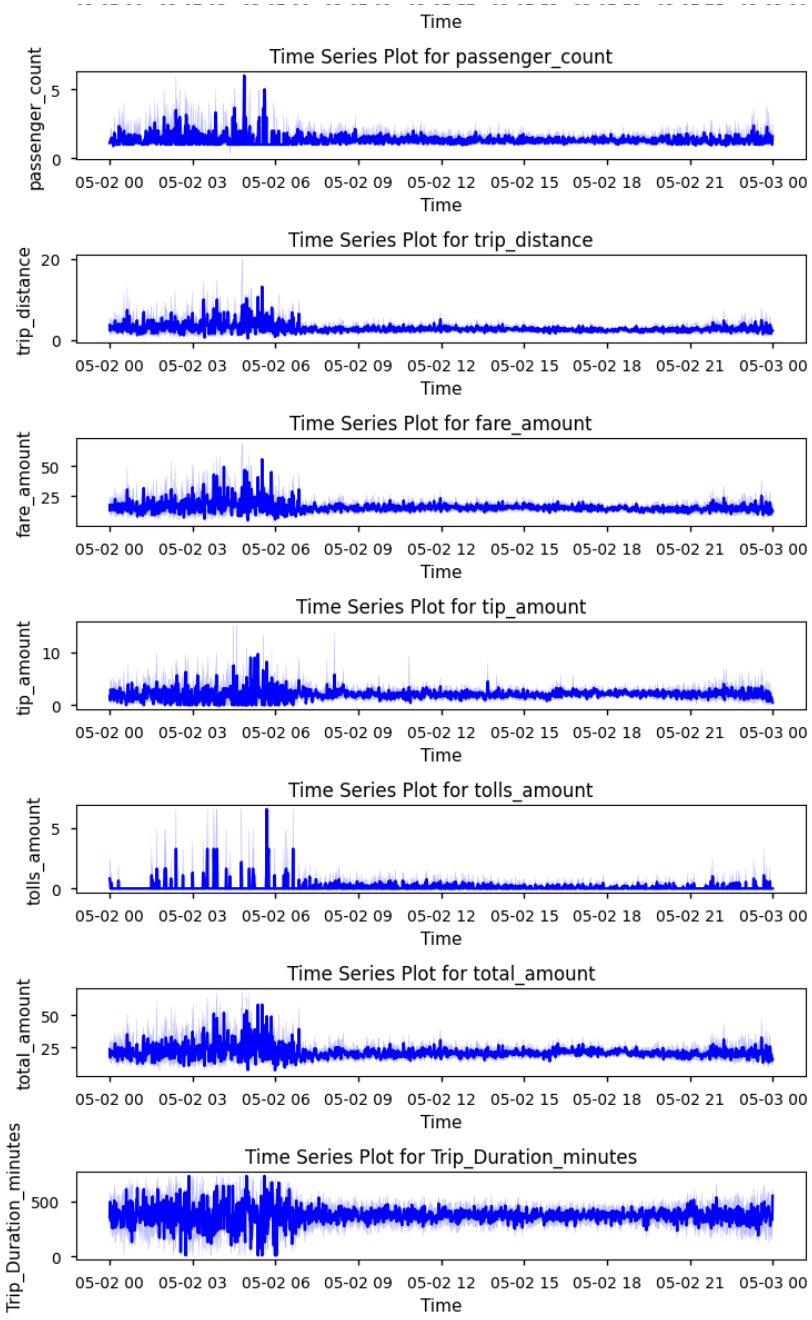

Histogram and KDE for tip_amount
Frequency
75000
50000
25000
0
0
10
20
30
40
50
60
70
80
tip_amount


Histogram and KDE for tolls_amount
Frequency
60000
40000
20000
0
0
2
4
6
8
tolls_amount

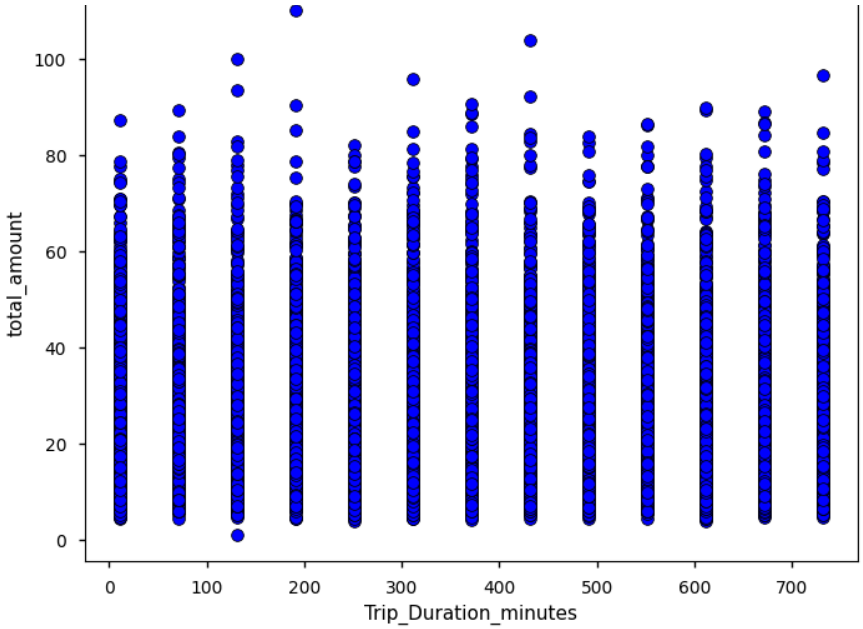

Histogram and KDE for total_amount
Frequency
10000
5000
0
0
20
40
60
80
100
total_amount

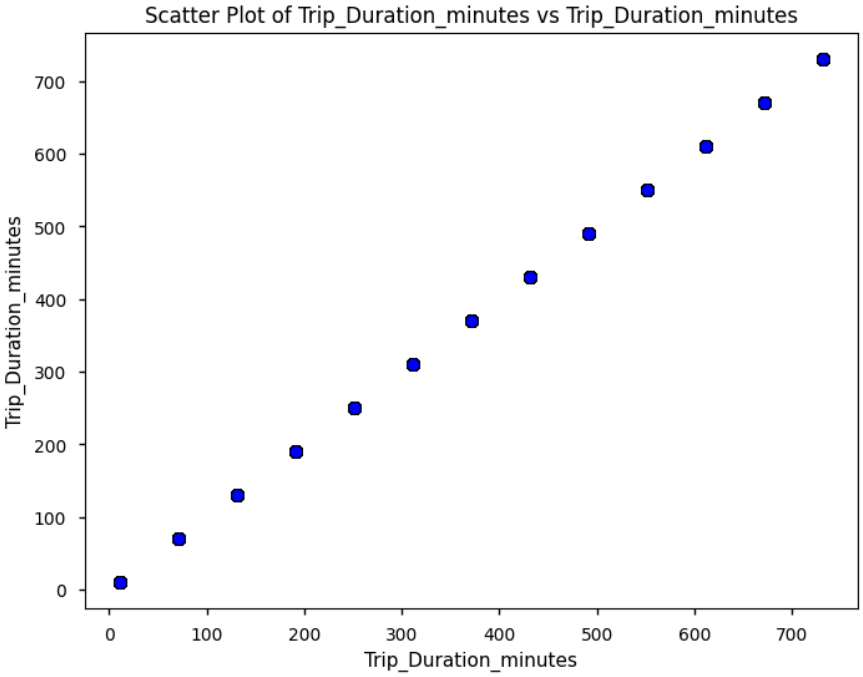

Histogram and KDE for Trip_Duration_minutes
Frequency
2000
1000
0
0
100
200
300
400
500
600
700
Trip_Duration_minutes

```
# Make sure you have a time column (e.g., 'timestamp') in your data
# Convert the time column to datetime format (if not already in datetime format)
data['Pickup_time'] = pd.to_datetime(data['Pickup_time'])

# Set the time column as the index
data.set_index('Pickup_time', inplace=True)

# Create a figure for the subplots
plt.figure(figsize=(8, 15))  # Adjust the figure size as needed

# Iterate through each column
for i, col in enumerate(columns):
    # Create a subplot for each column
    plt.subplot(len(columns), 1, i + 1)

    # Plot the time series data for each column
    sns.lineplot(data=data, x=data.index, y=col, color='blue')

    # Add a title to each subplot
    plt.title(f'Time Series Plot for {col}')

    # Add labels for x and y axes
    plt.xlabel('Time')
    plt.ylabel(col)

# Adjust the layout for better spacing
plt.tight_layout()

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


Time
Time Series Plot for passenger_count
passenger_count
0
5
05-02 00
05-02 03
05-02 06
05-02 09
05-02 12
05-02 15
05-02 18
05-02 21
05-03 00
Time
Time Series Plot for trip_distance
trip_distance
0
20
Time
Time Series Plot for fare_amount
fare_amount
25
50
Time
Time Series Plot for tip_amount
tip_amount
0
10
Time
Time Series Plot for tolls_amount
tolls_amount
0
5
Time
Time Series Plot for total_amount
total_amount
25
50
Time
Time Series Plot for Trip_Duration_minutes
Trip_Duration_minutes
0
500
Time

```
# Iterate through each pair of columns
for i, col_x in enumerate(columns):
    for j, col_y in enumerate(columns):
        # Create a subplot for each pair of columns
        plt.figure(figsize=(8, 6))

        # Plot a scatter plot for the pair of columns
        sns.scatterplot(data=data, x=col_x, y=col_y, color='b', edgecolors='black')

        # Add a title to the plot
        plt.title(f'Scatter Plot of {col_x} vs {col_y}')

        # Add labels for x and y axes
        plt.xlabel(col_x)
        plt.ylabel(col_y)

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


total_amount
Trip_Duration_minutes
0
20
40
60
80
100
0
100
200
300
400
500
600
700


Scatter Plot of Trip_Duration_minutes vs Trip_Duration_minutes
Trip_Duration_minutes
Trip_Duration_minutes
0
100
200
300
400
500
600
700

In [11]:
```
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm

data = pd.read_csv('Final_Data.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32044 entries, 0 to 32043
Data columns (total 29 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   VendorID                32044 non-null  int64  
 1   PULocationID            32044 non-null  int64  
 2   PUBorough               32044 non-null  object 
 3   PUZone                  32044 non-null  object 
 4   PUservice_zone          32044 non-null  object 
 5   DOLocationID            32044 non-null  int64  
 6   DOBorough               32044 non-null  object 
 7   DOZone                  32044 non-null  object 
 8   DOservice_zone          32044 non-null  object 
 9   passenger_count         32044 non-null  int64  
 10  trip_distance           32044 non-null  float64
 11  fare_amount             32044 non-null  float64
 12  extra                   32044 non-null  float64
 13  mta_tax                 32044 non-null  float64
 14  tip_amount              32044 non-null  float64
 15  tolls_amount            32044 non-null  float64
 16  improvement_surcharge   32044 non-null  float64
 17  total_amount            32044 non-null  float64
 18  payment_type            32044 non-null  int64  
 19  trip_type               32044 non-null  int64  
 20  congestion_surcharge    32044 non-null  float64
 21  Dropoff_year            32044 non-null  int64  
 22  Dropoff_day             32044 non-null  int64  
 23  Dropoff_time            32044 non-null  object 
 24  Pickup_year             32044 non-null  int64  
 25  Pickup_day              32044 non-null  int64  
 26  Pickup_time             32044 non-null  object 
 27  Day_Of_Ride_char        32044 non-null  object 
 28  Trip_Duration_minutes   32044 non-null  int64  
dtypes: float64(9), int64(11), object(9)
memory usage: 7.1+ MB
```

In [14]:
```
day_mapping = {
    1: 'Monday',
    2: 'Tuesday',
    3: 'Wednesday',
    4: 'Thursday',
    5: 'Friday',
    6: 'Saturday',
    7: 'Sunday',
    8: 'Monday',
    9: 'Tuesday',
    10: 'Wednesday',
    11: 'Thursday',
    12: 'Friday',
    13: 'Saturday',
    14: 'Sunday',
    15: 'Monday',
    16: 'Tuesday',
    17: 'Wednesday',
    18: 'Thursday',
    19: 'Friday',
    20: 'Saturday',
    21: 'Sunday',
    22: 'Monday',
    23: 'Tuesday',
    24: 'Wednesday',
    25: 'Thursday',
    26: 'Friday',
    27: 'Saturday',
    28: 'Sunday',
    29: 'Monday',
    30: 'Tuesday',
    31: 'Wednesday'
}

# Create a new column 'day_of_week' based on the mapping
data['day_of_week'] = data['Pickup_day'].map(day_mapping)
print(data.head())
```

```
   VendorID  PULocationID  PUBorough               PUZone PUservice_zone  \
0         2           166  Manhattan  Morningside Heights      Boro Zone   
1         2            24  Manhattan        Bloomingdale    Yellow Zone   
2         1            41  Manhattan       Central Harlem      Boro Zone   
3         1            41  Manhattan       Central Harlem      Boro Zone   
4         2            41  Manhattan       Central Harlem      Boro Zone   

   DOLocationID  DOBorough                DOZone DOservice_zone  \
0           143  Manhattan    Lincoln Square West    Yellow Zone   
1            43  Manhattan          Central Park    Yellow Zone   
2           238  Manhattan  Upper West Side North    Yellow Zone   
3            74  Manhattan      East Harlem North      Boro Zone   
4           262  Manhattan        Yorkville East    Yellow Zone   

   passenger_count  ...  congestion_surcharge  Dropoff_year  Dropoff_day  \
0                1  ...                  2.75          2023            1   
1                1  ...                  0.00          2023            1   
2                1  ...                  0.00          2023            1   
3                1  ...                  0.00          2023            1   
4                1  ...                  2.75          2023            1   

   Dropoff_time  Pickup_year  Pickup_day  Pickup_time  Day_Of_Ride_char  \
0      0:37:00         2023           1      0:26:00            Monday   
1      0:57:00         2023           1      0:51:00            Monday   
2      0:19:00         2023           1      0:13:00            Monday   
3      0:39:00         2023           1      0:33:00            Monday   
4      1:11:00         2023           1      0:53:00            Monday   

   Trip_Duration_minutes  day_of_week  
0                     11       Monday  
1                     71       Monday  
2                    131       Monday  
3                    191       Monday  
4                    251       Monday  

[5 rows x 30 columns]
```

In [15]:
```
data['day_of_week']
```

Out[15]:
```
0           Monday
1           Monday
2           Monday
3           Monday
4           Monday
            ...   
32039    Wednesday
32040    Wednesday
32041    Wednesday
32042    Wednesday
32043    Wednesday
Name: day_of_week, Length: 32044, dtype: object
```

In [16]:
```
selected_columns = ['trip_distance', 'total_amount','day_of_week']
trip_fare_data = data[selected_columns]
trip_fare_data
```

Out[16]:

	trip_distance	total_amount	day_of_week
0	2.58	24.18	Monday
1	1.81	15.84	Monday
2	1.30	10.20	Monday
3	1.10	8.00	Monday
4	2.78	22.95	Monday
...	...	...	...
32039	1.45	11.80	Wednesday
32040	1.71	15.84	Wednesday
32041	1.45	15.30	Wednesday
32042	0.91	9.00	Wednesday
32043	11.18	44.70	Wednesday

32044 rows × 3 columns

In [17]:
```
from scipy.stats import kruskal

# Extract data for each day of the week
days = trip_fare_data['day_of_week'].unique()
data_by_day = {day: trip_fare_data[trip_fare_data['day_of_week'] == day]['total_amount'] for day in days}

# Perform Kruskal-Wallis test for each day
test_results = {}
for day, amounts in data_by_day.items():
    statistic, p_value = kruskal(amounts, trip_fare_data['trip_distance'])
    test_results[day] = {'statistic': statistic, 'p_value': p_value}

# Print test results
for day, results in test_results.items():
    print(f"Day {day}:")
    print(f"Kruskal-Wallis Statistic: {results['statistic']}")
    print(f"p-value: {results['p_value']}")
    if results['p_value'] > 0.05:
        print("Fail to reject H0: Total amount does not depend significantly on trip distance for this day.")
    else:
        print("Reject H0: Total amount depends significantly on trip distance for this day.")
    print()
```

```
Day Monday:
Kruskal-Wallis Statistic: 10133.637621807226
p-value: 0.0
Reject H0: Total amount depends significantly on trip distance for this day.

Day Tuesday:
Kruskal-Wallis Statistic: 12053.7868967797
p-value: 0.0
Reject H0: Total amount depends significantly on trip distance for this day.

Day Wednesday:
Kruskal-Wallis Statistic: 13387.789836390742
p-value: 0.0
Reject H0: Total amount depends significantly on trip distance for this day.

Day Thursday:
Kruskal-Wallis Statistic: 11954.968293776332
p-value: 0.0
Reject H0: Total amount depends significantly on trip distance for this day.

Day Friday:
Kruskal-Wallis Statistic: 12251.915993879173
p-value: 0.0
Reject H0: Total amount depends significantly on trip distance for this day.

Day Saturday:
Kruskal-Wallis Statistic: 12012.869687779823
p-value: 0.0
Reject H0: Total amount depends significantly on trip distance for this day.

Day Sunday:
Kruskal-Wallis Statistic: 10409.878148190419
p-value: 0.0
Reject H0: Total amount depends significantly on trip distance for this day.
```

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

```
df=pd.read_csv('/content/cleaned_dataset.csv')
```

```
df.head()
```

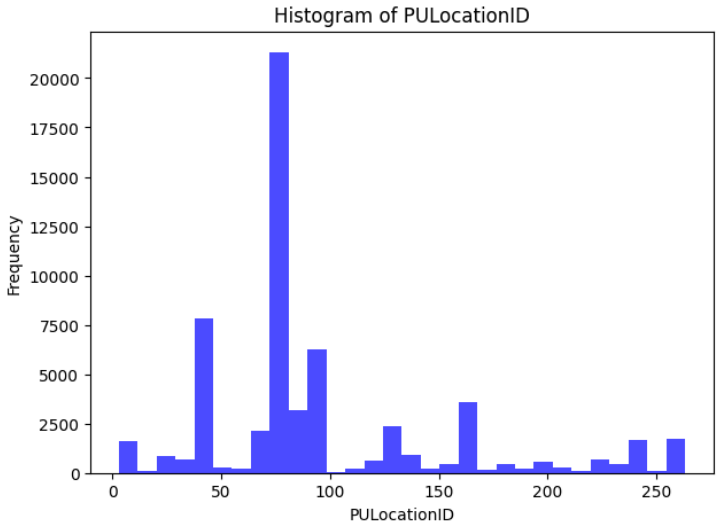
	VendorID	PULocationID	PUBorough	PUZone	PUservice_zone	DOLocationID	DOBorough	DOZone	DOservice_zone	passenger_count	...	congestion_surcharge
0	2	166	Manhattan	Morningside Heights	Boro Zone	143	Manhattan	Lincoln Square West	Yellow Zone	1	...	2.75
1	2	24	Manhattan	Bloomingdale	Yellow Zone	43	Manhattan	Central Park	Yellow Zone	1	...	0.00
2	1	41	Manhattan	Central Harlem	Boro Zone	238	Manhattan	Upper West Side North	Yellow Zone	1	...	0.00
3	1	41	Manhattan	Central Harlem	Boro Zone	74	Manhattan	East Harlem North	Boro Zone	1	...	0.00
4	2	41	Manhattan	Central Harlem	Boro Zone	262	Manhattan	Yorkville East	Yellow Zone	1	...	2.75

5 rows × 30 columns

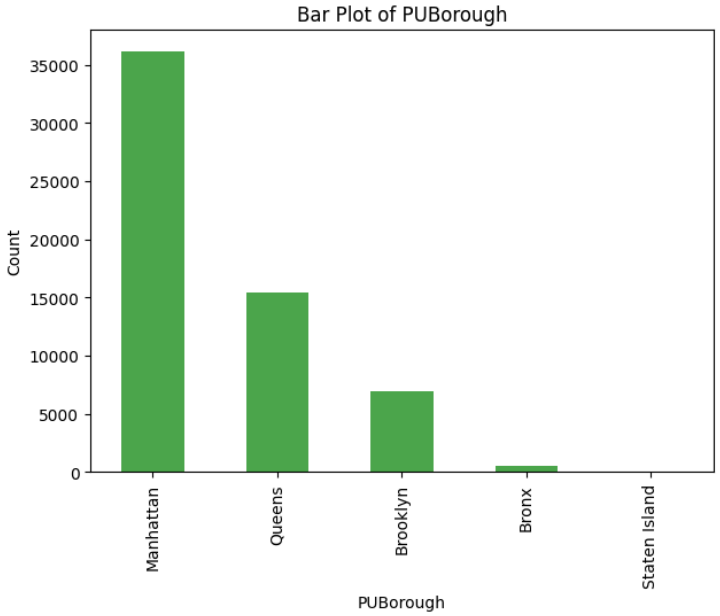
```
df.columns
```

```
Index(['VendorID', 'PULocationID', 'PUBorough', 'PUZone', 'PUservice_zone',
       'DOLocationID', 'DOBorough', 'DOZone', 'DOservice_zone',
       'passenger_count', 'trip_distance', 'fare_amount', 'extra', 'mta_tax',
       'tip_amount', 'tolls_amount', 'improvement_surcharge', 'total_amount',
       'payment_type', 'trip_type', 'congestion_surcharge', 'Dropoff_year',
       'Dropoff_day', 'Dropoff_time', 'Pickup_year', 'Pickup_day',
       'Pickup_time', 'Day_Of_Ride_char', 'Duration', 'Total_minutes'],
      dtype='object')
```

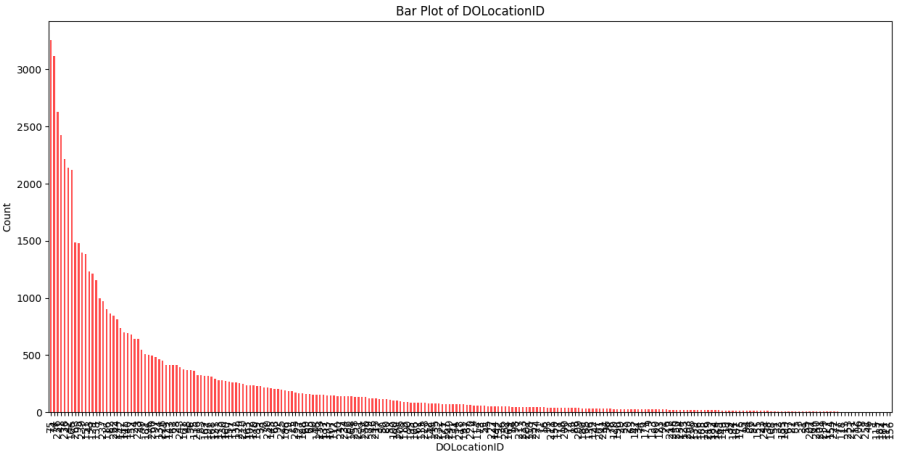
```
# Histogram for 'PULocationID'
plt.figure(figsize=(7, 5))
plt.hist(df['PULocationID'].dropna(), bins=30, color='blue', alpha=0.7)
plt.title('Histogram of PULocationID')
plt.xlabel('PULocationID')
plt.ylabel('Frequency')
plt.show()
```



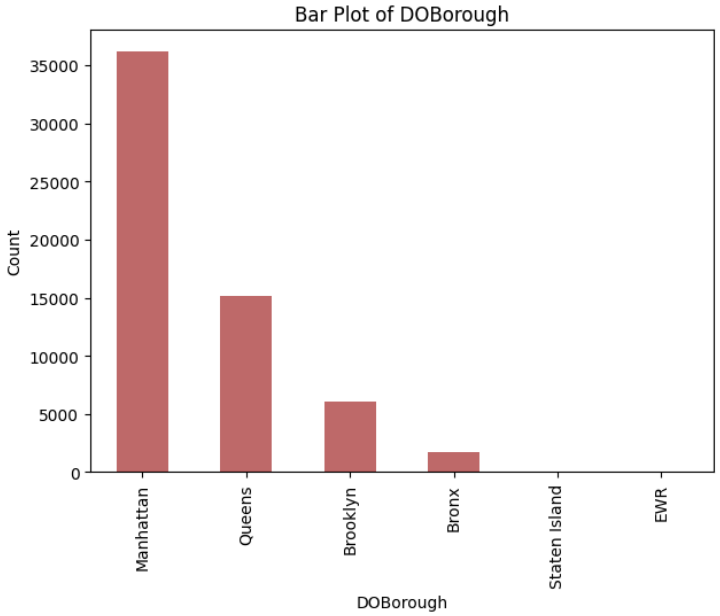

```
# Histogram for 'PUBorough'
plt.figure(figsize=(7, 5))
df['PUBorough'].value_counts().plot(kind='bar', color='green', alpha=0.7)
plt.title('Bar Plot of PUBorough')
plt.xlabel('PUBorough')
plt.ylabel('Count')
plt.show()
```



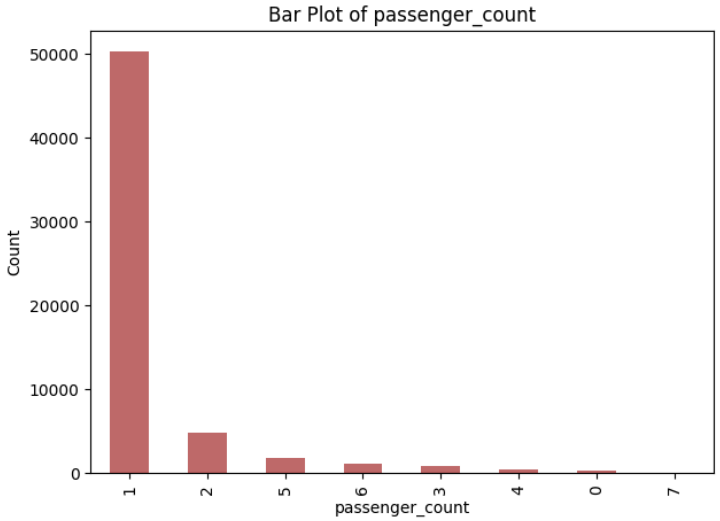

```
# Histogram for 'DOLocationID'
plt.figure(figsize=(15, 7))
df['DOLocationID'].value_counts().plot(kind='bar', color='red', alpha=0.7)
plt.title('Bar Plot of DOLocationID')
plt.xlabel('DOLocationID')
plt.ylabel('Count')
plt.show()
```




```
# Histogram for 'DOBorough'
plt.figure(figsize=(7, 5))
df['DOBorough'].value_counts().plot(kind='bar', color='brown', alpha=0.7)
plt.title('Bar Plot of DOBorough')
plt.xlabel('DOBorough')
plt.ylabel('Count')
plt.show()
```

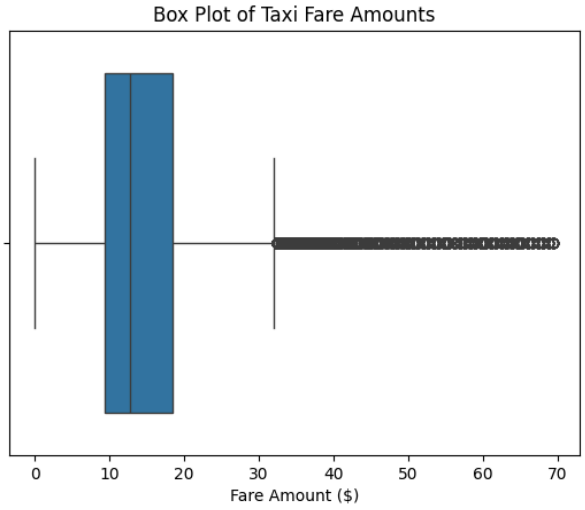



```
# Histogram for 'passenger_count'
plt.figure(figsize=(7, 5))
df['passenger_count'].value_counts().plot(kind='bar', color='brown', alpha=0.7)
plt.title('Bar Plot of passenger_count')
plt.xlabel('passenger_count')
plt.ylabel('Count')
plt.show()
```

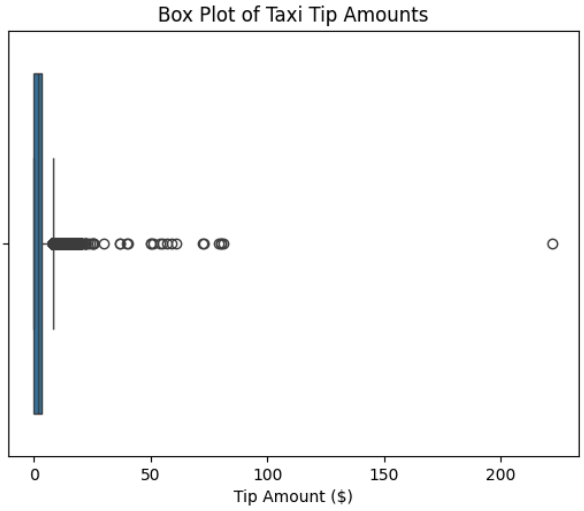



```
Start coding or generate with AI.
```

```
# Box plot for fare amount
sns.boxplot(x=df['fare_amount']).set(title='Box Plot of Taxi Fare Amounts')
plt.xlabel('Fare Amount ($)')
plt.show()
```



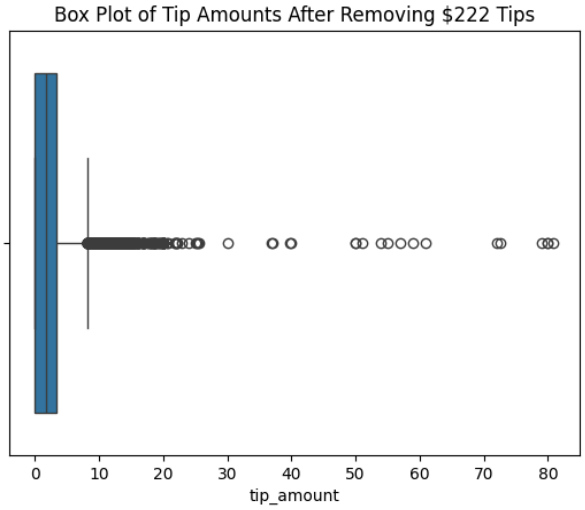

```
# Box plot for tip amount
sns.boxplot(x=df['tip_amount']).set(title='Box Plot of Taxi Tip Amounts')
plt.xlabel('Tip Amount ($)')
plt.show()
```



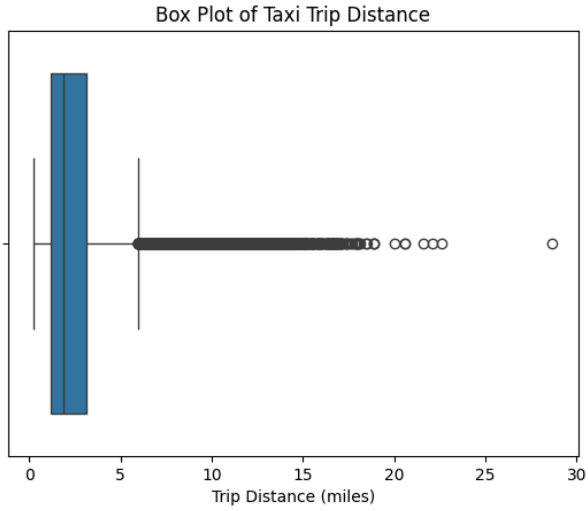

```
#Filter out rows where 'tip_amount' is greater than 100
df = df[df['tip_amount'] <= 100]

# Check the shape of the DataFrame after the operation
print("DataFrame shape after removing rows with tip_amount > 100:", df.shape)
```

```
DataFrame shape after removing rows with tip_amount > 100: (59158, 30)
```

```
sns.boxplot(x=df['tip_amount'])
plt.title('Box Plot of Tip Amounts After Removing $222 Tips')
plt.show()
```




```
# Box plot for trip distance
sns.boxplot(x=df['trip_distance']).set(title='Box Plot of Taxi Trip Distance')
plt.xlabel('Trip Distance (miles)')
plt.show()
```



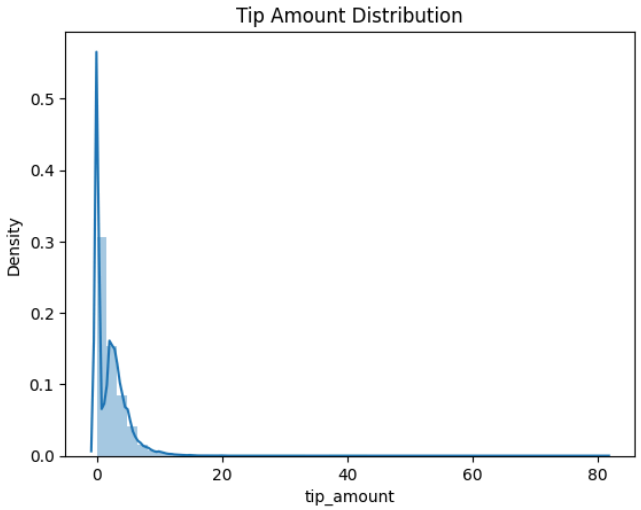

```
# Summary statistics
summary = df.describe()
print(summary)
```

```
           VendorID  PULocationID  DOLocationID  passenger_count  \
count  59158.000000  59158.000000  59158.000000     59158.000000
mean       1.872393     96.138764    138.026353         1.324098
std        0.333655     58.483043     76.371527         0.992160
min        1.000000      3.000000      1.000000         0.000000
25%        2.000000     74.000000     74.000000         1.000000
50%        2.000000     75.000000    138.000000         1.000000
75%        2.000000     97.000000    220.000000         1.000000
max        2.000000    263.000000    263.000000         7.000000

       trip_distance   fare_amount         extra       mta_tax    tip_amount  \
count   59158.000000  59158.000000  59158.000000  59158.000000  59158.000000
mean        2.596931     15.396912      0.921427      0.621091      2.076723
std         2.204801      9.353774      1.305405      0.338462      2.659227
min         0.270000      0.010000      0.000000      0.000000      0.000000
25%         1.220000      9.300000      0.000000      0.500000      0.000000
50%         1.890000     12.800000      0.000000      0.500000      1.740000
75%         3.130000     18.400000      2.500000      0.500000      3.280000
max        28.670000     69.500000      7.500000      1.500000     80.880000

       tolls_amount  improvement_surcharge  total_amount  payment_type  \
count  59158.000000           59158.000000  59158.000000  59158.000000
mean       0.105886               0.956006     20.617358      1.364346
std        0.821667               0.170719     11.446713      0.490022
min        0.000000               0.000000      1.010000      1.000000
25%        0.000000               1.000000     12.900000      1.000000
50%        0.000000               1.000000     17.500000      1.000000
75%        0.000000               1.000000     25.100000      2.000000
max       14.750000               1.000000    130.700000      4.000000

          trip_type  congestion_surcharge  Dropoff_year   Dropoff_day  \
```

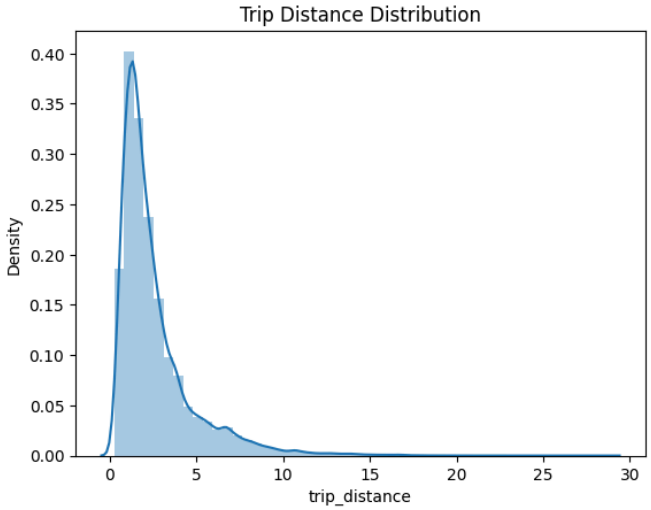
```
       count  59158.000000          59158.000000       59158.0  59158.000000
       mean       1.009804              0.773919        2023.0     16.358971
       std        0.098531              1.236602           0.0      8.767089
       min        1.000000              0.000000        2023.0      1.000000
       25%        1.000000              0.000000        2023.0      9.000000
       50%        1.000000              0.000000        2023.0     17.000000
       75%        1.000000              2.750000        2023.0     24.000000
       max        2.000000              2.750000        2023.0     31.000000

              Pickup_year    Pickup_day  Total_minutes
       count      59158.0  59158.000000   59158.000000
       mean        2023.0     16.351702     700.998986
       std            0.0      8.766792     415.323959
       min         2023.0      1.000000      11.000000
       25%         2023.0      9.000000     371.000000
       50%         2023.0     17.000000     701.000000
       75%         2023.0     24.000000    1031.000000
       max         2023.0     31.000000    1391.000000
```

```
# Distribution plots
sns.distplot(df['tip_amount'])
plt.title('Tip Amount Distribution')
plt.show()

sns.distplot(df['trip_distance'])
plt.title('Trip Distance Distribution')
plt.show()
```

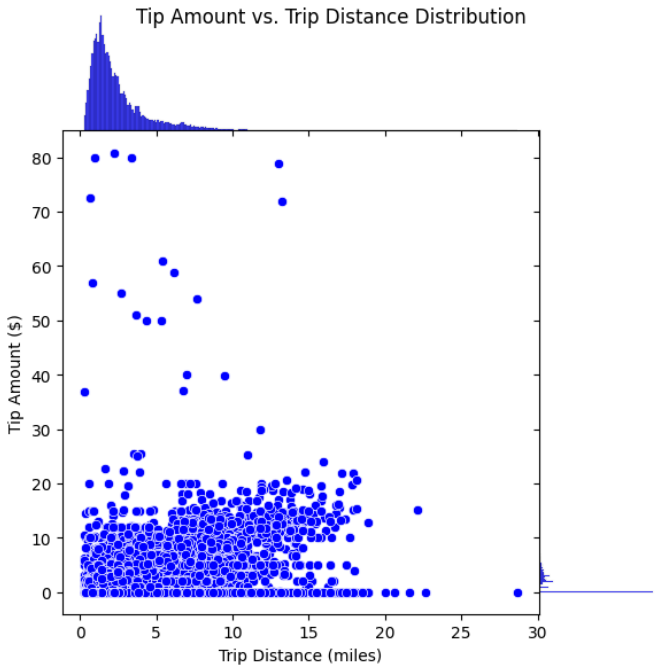
```
<ipython-input-103-4978ba3c7134>:2: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['tip_amount'])
```



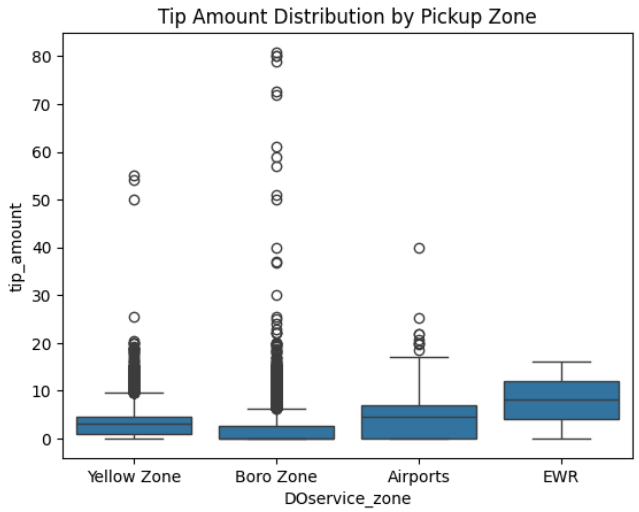

```
<ipython-input-103-4978ba3c7134>:6: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(df['trip_distance'])
```




```
# Create a joint plot of tip amount vs trip distance
sns.jointplot(x='trip_distance', y='tip_amount', data=df, kind='scatter', color='b', space=0, height=6, ratio=4)

# Add titles and labels
plt.suptitle('Tip Amount vs. Trip Distance Distribution', verticalalignment='top', fontsize=12)
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Tip Amount ($)')

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

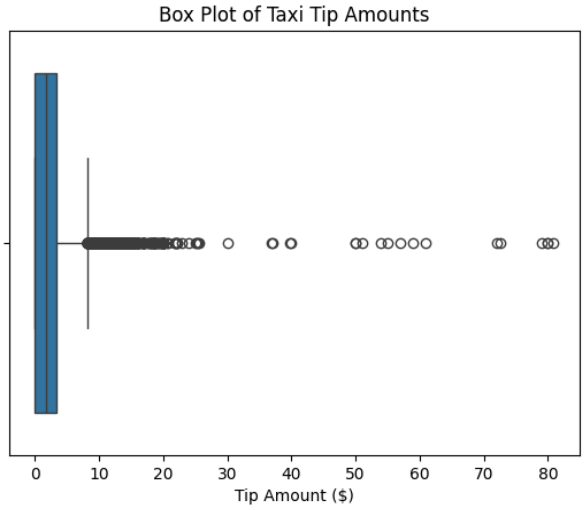



```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'PUZone' or 'DOZone' contains the zone information
# Here's how you can visualize the tip amount distributions across pickup zones
sns.boxplot(x='DOservice_zone', y='tip_amount', data=df)
plt.title('Tip Amount Distribution by Pickup Zone')
plt.show()
```




```
Start coding or generate with AI.
```

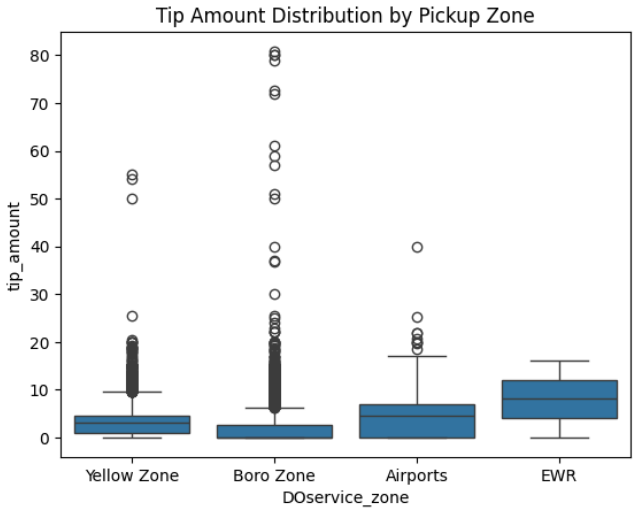
```
# Box plot for tip amount
sns.boxplot(x=df['tip_amount']).set(title='Box Plot of Taxi Tip Amounts')
plt.xlabel('Tip Amount ($)')
plt.show()
```




```
df.describe()
```

	VendorID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount	ex
count	59158.000000	59158.000000	59158.000000	59158.000000	59158.000000	59158.000000	59158.000(
mean	1.872393	96.138764	138.026353	1.324098	2.596931	15.396912	0.921
std	0.333655	58.483043	76.371527	0.992160	2.204801	9.353774	1.305
min	1.000000	3.000000	1.000000	0.000000	0.270000	0.010000	0.000(
25%	2.000000	74.000000	74.000000	1.000000	1.220000	9.300000	0.000(
50%	2.000000	75.000000	138.000000	1.000000	1.890000	12.800000	0.000(
75%	2.000000	97.000000	220.000000	1.000000	3.130000	18.400000	2.500(
max	2.000000	263.000000	263.000000	7.000000	28.670000	69.500000	7.500(

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

# Assuming 'PUZone' or 'DOZone' contains the zone information
# Here's how you can visualize the tip amount distributions across pickup zones
sns.boxplot(x='DOservice_zone', y='tip_amount', data=df)
plt.title('Tip Amount Distribution by Pickup Zone')
plt.show()
```



Hypothesis Statement

Null Hypothesis (H0)

The median tip amounts are the same across the different trip distance categories. Specifically, there is no significant difference in the median tips for trips that are:

- Less than 5 miles,
- Between 5 and 10 miles,
- Between 10 and 15 miles,
- Between 15 and 20 miles,
- Between 20 and 25 miles,

Alternative Hypothesis (Ha)

At least one of the specified trip distance categories has a median tip amount that is significantly different from the others.

Kruskal-Wallis test

```
from scipy.stats import kruskal
```

```
'''# Example of categorizing trip distances
bins = [0, 2, 6, 10, float('inf')]  # Defining bins as per the categories
labels = ['<2 miles', '3-6 miles', '7-10 miles', '>10 miles']
df['distance_category'] = pd.cut(df['trip_distance'], bins=bins, labels=labels)'''
```

```
'# Example of categorizing trip distances\nbins = [0, 2, 6, 10, float('inf')]  # Defining bins as per
the categories\nlabels = ['<2 miles', '3-6 miles', '7-10 miles', '>10 miles']\ndf['distance_category']
= pd.cut(df['trip_distance'], bins=bins, labels=labels)'
```

```
# Example of categorizing trip distances
bins = [0, 5, 10, 15, 20, 25, float('inf')]  # Extending to infinity to include all possible longer trips
labels = ['<5 miles', '5-10 miles', '10-15 miles', '15-20 miles', '20-25 miles', '>25 miles']
df['distance_category'] = pd.cut(df['trip_distance'], bins=bins, labels=labels)
```

```
# Extract tip amounts for each distance category
data_groups = [group['tip_amount'].values for name, group in df.groupby('distance_category')]

# Perform the Kruskal-Wallis test
stat, p_value = kruskal(*data_groups)

print('Kruskal-Wallis H statistic:', stat)
print('P-value:', p_value)
alpha = 0.05  # Commonly used significance level

if p_value < alpha:
    print("Reject the null hypothesis - There is a significant association between tip amount and trip distance.")
else:
    print("Fail to reject the null hypothesis - There is no significant association between tip amount and trip distance.")
```

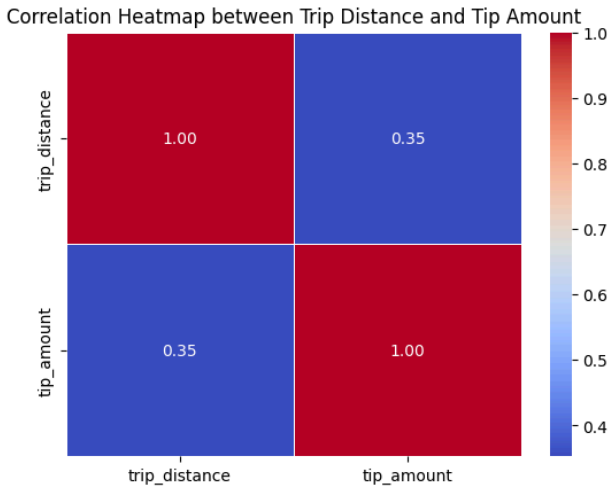
```
Kruskal-Wallis H statistic: 1098.429276209346
P-value: 2.926010222312085e-235
Reject the null hypothesis - There is a significant association between tip amount and trip distance.
```

```
Start coding or generate with AI.
```

```
# Calculate Pearson correlation
pearson_corr = df[['trip_distance', 'tip_amount']].corr(method='pearson')
print("Pearson Correlation:\n", pearson_corr)
```

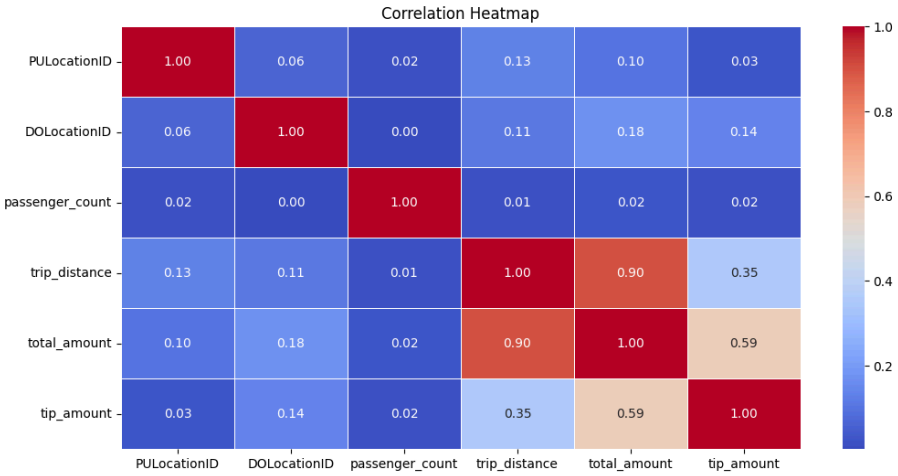
```
Pearson Correlation:
                trip_distance  tip_amount
trip_distance       1.000000    0.353048
tip_amount          0.353048    1.000000
```

```
# Heatmap for Pearson correlation
sns.heatmap(pearson_corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap between Trip Distance and Tip Amount')
plt.show()
```

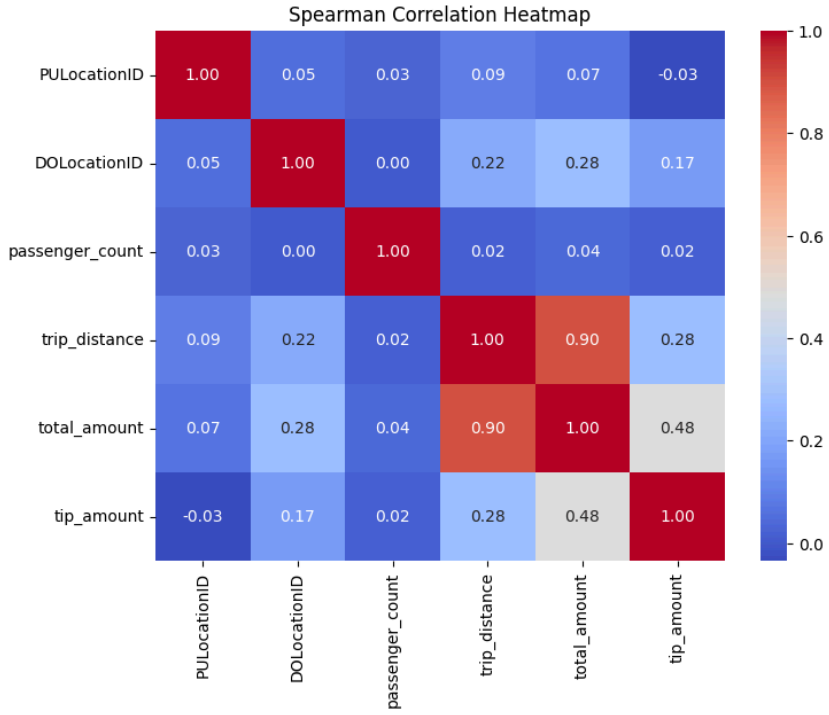


Correlation Analysis

```
selected_columns = ['PULocationID', 'DOLocationID', 'passenger_count','trip_distance', 'total_amount', 'tip_amount']
correlation_matrix = df[selected_columns].corr()
plt.figure(figsize=(12, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
```




```
selected_columns = ['PULocationID', 'DOLocationID', 'passenger_count','trip_distance', 'total_amount', 'tip_amount']
spearman_corr = df[selected_columns].corr(method='spearman')
# Create a heatmap for Spearman's correlation
plt.figure(figsize=(8, 6))
sns.heatmap(spearman_corr, annot=True, fmt=".2f", cmap='coolwarm', cbar=True)
plt.title('Spearman Correlation Heatmap')
plt.show()
```



ANOVA

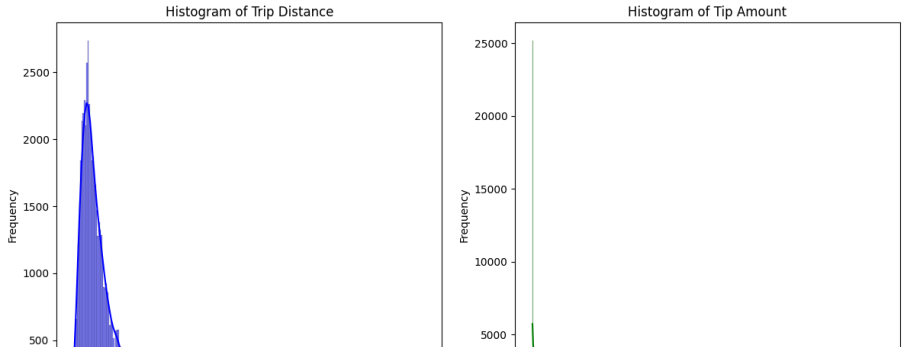
```
import statsmodels.api as sm
from statsmodels.formula.api import ols
```

```
model = ols('tip_amount ~ C(DOservice_zone)', data=df).fit()
```

```
anova_results = sm.stats.anova_lm(model, typ=2)  # Type 2 ANOVA DataFrame
print(anova_results)
```

```
                          sum_sq       df            F  PR(>F)
C(DOservice_zone)   39580.929923      3.0  2060.627785     0.0
Residual           378747.089504  59154.0          NaN     NaN
```

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

# Histogram for Trip Distance
plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
sns.histplot(df['trip_distance'], kde=True, color='blue')
plt.title('Histogram of Trip Distance')
plt.xlabel('Trip Distance')
plt.ylabel('Frequency')

# Histogram for Tip Amount
plt.subplot(1, 2, 2)  # 1 row, 2 columns, 2nd subplot
sns.histplot(df['tip_amount'], kde=True, color='green')
plt.title('Histogram of Tip Amount')
plt.xlabel('Tip Amount')
plt.ylabel('Frequency')

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



Estimate paramters


```
# For Trip Distance
```

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
pd.set_option('display.max_columns', 30)
```

```
C:\Users\AMIT\AppData\Roaming\Python\Python311\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas req
uires version '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

In [2]:
```
df = pd.read_csv(r"C:\Users\AMIT\Downloads\NYC TLC Trip Record_with location.csv")
df
```

```
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\3290471142.py:1: DtypeWarning: Columns (3) have mixed types. Specify
dtype option on import or set low_memory=False.
  df = pd.read_csv(r"C:\Users\AMIT\Downloads\NYC TLC Trip Record_with location.csv")
```

Out[2]:

	VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	store_and_fwd_flag	RatecodeID	PULocationID	PUBorough	PUZone	PUserv
0	2	1/1/2023 0:26	1/1/2023 0:37	N	1.0	166	Manhattan	Morningside Heights	E
1	2	1/1/2023 0:51	1/1/2023 0:57	N	1.0	24	Manhattan	Bloomingdale	Yel
2	2	1/1/2023 0:35	1/1/2023 0:41	N	1.0	223	Queens	Steinway	E
3	1	1/1/2023 0:13	1/1/2023 0:19	N	1.0	41	Manhattan	Central Harlem	E
4	1	1/1/2023 0:33	1/1/2023 0:39	N	1.0	41	Manhattan	Central Harlem	E
...	...	...	...	...	...	...	...	...	
68206	2	1/31/2023 22:29	1/31/2023 22:42	NaN	NaN	49	Brooklyn	Clinton Hill	E
68207	2	1/31/2023 22:40	1/31/2023 22:48	NaN	NaN	10	Queens	Baisley Park	E
68208	2	1/31/2023 23:46	2/1/2023 0:02	NaN	NaN	66	Brooklyn	DUMBO/Vinegar Hill	E
68209	2	1/31/2023 23:01	1/31/2023 23:19	NaN	NaN	225	Brooklyn	Stuyvesant Heights	E
68210	2	1/31/2023 23:51	2/1/2023 0:07	NaN	NaN	256	Brooklyn	Williamsburg (South Side)	E

68211 rows × 26 columns

In [3]:
```
columns_to_remove = ['RatecodeID','ehail_fee','store_and_fwd_flag']
data = df.drop(columns=columns_to_remove)
```

In [4]:
```
data
```

Out[4]:

	VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	PULocationID	PUBorough	PUZone	PUservice_zone	DOLocationID	DOBorou
0	2	1/1/2023 0:26	1/1/2023 0:37	166	Manhattan	Morningside Heights	Boro Zone	143	Manha
1	2	1/1/2023 0:51	1/1/2023 0:57	24	Manhattan	Bloomingdale	Yellow Zone	43	Manha
2	2	1/1/2023 0:35	1/1/2023 0:41	223	Queens	Steinway	Boro Zone	179	Que
3	1	1/1/2023 0:13	1/1/2023 0:19	41	Manhattan	Central Harlem	Boro Zone	238	Manha
4	1	1/1/2023 0:33	1/1/2023 0:39	41	Manhattan	Central Harlem	Boro Zone	74	Manha
...	...	...	...	...	...	...	...	...	
68206	2	1/31/2023 22:29	1/31/2023 22:42	49	Brooklyn	Clinton Hill	Boro Zone	62	Brool
68207	2	1/31/2023 22:40	1/31/2023 22:48	10	Queens	Baisley Park	Boro Zone	205	Que
68208	2	1/31/2023 23:46	2/1/2023 0:02	66	Brooklyn	DUMBO/Vinegar Hill	Boro Zone	37	Brool
68209	2	1/31/2023 23:01	1/31/2023 23:19	225	Brooklyn	Stuyvesant Heights	Boro Zone	189	Brool
68210	2	1/31/2023 23:51	2/1/2023 0:07	256	Brooklyn	Williamsburg (South Side)	Boro Zone	140	Manha

68211 rows × 23 columns

In [5]:
```
data.describe()
```

Out[5]:

	VendorID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount	extra	mta_tax	tip_amount	tolls_
count	68211.000000	68211.000000	68211.000000	63887.000000	68211.000000	68211.000000	68211.000000	68211.000000	68211.000000	68211
mean	1.863028	98.549735	138.429901	1.315870	8.114852	16.603545	0.825431	0.588340	2.139012	0
std	0.343820	61.244314	76.761311	0.979054	585.105955	13.470121	1.269904	0.385819	3.052710	1
min	1.000000	1.000000	1.000000	0.000000	0.000000	-70.000000	-2.500000	-0.500000	-10.500000	0
25%	2.000000	74.000000	74.000000	1.000000	1.110000	9.300000	0.000000	0.500000	0.000000	0
50%	2.000000	75.000000	138.000000	1.000000	1.850000	13.500000	0.000000	0.500000	1.600000	0
75%	2.000000	129.000000	219.000000	1.000000	3.210000	19.800000	1.000000	0.500000	3.330000	0
max	2.000000	265.000000	265.000000	9.000000	120098.840000	490.000000	12.500000	2.750000	222.220000	36

In [6]:
```
data = data[data['tip_amount'] != 222.22]
```

In [7]:
```
taxiDf = data
```

In [8]:
```
data.isnull().sum()
```

Out[8]:
```
VendorID                  0
lpep_pickup_datetime      0
lpep_dropoff_datetime     0
PULocationID              0
PUBorough                 0
PUZone                   66
PUservice_zone          201
DOLocationID              0
DOBorough                 0
DOZone                  242
DOservice_zone          668
passenger_count        4324
trip_distance             0
fare_amount               0
extra                     0
mta_tax                   0
tip_amount                0
tolls_amount              0
improvement_surcharge     0
total_amount              0
payment_type           4324
trip_type              4334
congestion_surcharge   4324
dtype: int64
```

In [9]:
```
data.dropna(subset=['PUZone','PUservice_zone','DOZone','DOservice_zone', 'passenger_count', 'payment_type', 'trip_type
```

```
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\4016327174.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data.dropna(subset=['PUZone','PUservice_zone','DOZone','DOservice_zone', 'passenger_count', 'payment_type', 'trip_type', 'congestion_surcharge'], inplace=True)
```

In [10]:
```
data.isnull().sum()
```

Out[10]:
```
VendorID                 0
lpep_pickup_datetime     0
lpep_dropoff_datetime    0
PULocationID             0
PUBorough                0
PUZone                   0
PUservice_zone           0
DOLocationID             0
DOBorough                0
DOZone                   0
DOservice_zone           0
passenger_count          0
trip_distance            0
fare_amount              0
extra                    0
mta_tax                  0
tip_amount               0
tolls_amount             0
improvement_surcharge    0
total_amount             0
payment_type             0
trip_type                0
congestion_surcharge     0
dtype: int64
```

In [11]:

```
data['lpep_dropoff_datetime'] = pd.to_datetime(data['lpep_dropoff_datetime'], format='%m/%d/%Y %H:%M')

data['Dropoff_year'] = data['lpep_dropoff_datetime'].dt.year
data['Dropoff_month'] = data['lpep_dropoff_datetime'].dt.month
data['Dropoff_day'] = data['lpep_dropoff_datetime'].dt.day

data['Dropoff_time'] = data['lpep_dropoff_datetime'].dt.time

# Now, data has four new columns: 'year', 'month', 'day', 'time'
```

```
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\1670506683.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['lpep_dropoff_datetime'] = pd.to_datetime(data['lpep_dropoff_datetime'], format='%m/%d/%Y %H:%M')
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\1670506683.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Dropoff_year'] = data['lpep_dropoff_datetime'].dt.year
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\1670506683.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Dropoff_month'] = data['lpep_dropoff_datetime'].dt.month
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\1670506683.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Dropoff_day'] = data['lpep_dropoff_datetime'].dt.day
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\1670506683.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Dropoff_time'] = data['lpep_dropoff_datetime'].dt.time
```

In [12]:

```
data['lpep_pickup_datetime'] = pd.to_datetime(data['lpep_pickup_datetime'], format='%m/%d/%Y %H:%M')

data['Pickup_year'] = data['lpep_pickup_datetime'].dt.year
data['Pickup_month'] = data['lpep_pickup_datetime'].dt.month
data['Pickup_day'] = data['lpep_pickup_datetime'].dt.day

data['Pickup_time'] = data['lpep_pickup_datetime'].dt.time

# Now, data has four new columns: 'year', 'month', 'day', 'time'
```

```
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\2755526917.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['lpep_pickup_datetime'] = pd.to_datetime(data['lpep_pickup_datetime'], format='%m/%d/%Y %H:%M')
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\2755526917.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Pickup_year'] = data['lpep_pickup_datetime'].dt.year
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\2755526917.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Pickup_month'] = data['lpep_pickup_datetime'].dt.month
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\2755526917.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Pickup_day'] = data['lpep_pickup_datetime'].dt.day
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\2755526917.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['Pickup_time'] = data['lpep_pickup_datetime'].dt.time
```

In [13]:
```
data
```

Out[13]:

	VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	PULocationID	PUBorough	PUZone	PUservice_zone	DOLocationID	DOBo
0	2	2023-01-01 00:26:00	2023-01-01 00:37:00	166	Manhattan	Morningside Heights	Boro Zone	143	Mar
1	2	2023-01-01 00:51:00	2023-01-01 00:57:00	24	Manhattan	Bloomingdale	Yellow Zone	43	Mar
2	2	2023-01-01 00:35:00	2023-01-01 00:41:00	223	Queens	Steinway	Boro Zone	179	C
3	1	2023-01-01 00:13:00	2023-01-01 00:19:00	41	Manhattan	Central Harlem	Boro Zone	238	Mar
4	1	2023-01-01 00:33:00	2023-01-01 00:39:00	41	Manhattan	Central Harlem	Boro Zone	74	Mar
...	...	...	...	...	...	...	...	...	
63882	2	2023-01-31 23:09:00	2023-01-31 23:17:00	130	Queens	Jamaica	Boro Zone	205	C
63883	2	2023-01-31 23:06:00	2023-01-31 23:17:00	65	Brooklyn	Downtown Brooklyn/MetroTech	Boro Zone	181	Br
63884	2	2023-01-31 23:17:00	2023-01-31 23:23:00	244	Manhattan	Washington Heights South	Boro Zone	116	Mar
63885	2	2023-01-31 23:29:00	2023-01-31 23:38:00	74	Manhattan	East Harlem North	Boro Zone	238	Mar
63886	2	2023-01-31 23:00:00	2023-01-31 23:10:00	95	Queens	Forest Hills	Boro Zone	95	C

63185 rows × 31 columns

In [14]:
```
data['Pickup_year'].value_counts()
```

Out[14]:
```
Pickup_year
2023    63182
2022        2
2009        1
Name: count, dtype: int64
```

In [15]:
```
data['Dropoff_month'].value_counts()
```

Out[15]:
```
Dropoff_month
1     63169
2        14
12        2
Name: count, dtype: int64
```

In [16]:
```
data['Pickup_month'].value_counts()
```

Out[16]:
```
Pickup_month
1     63182
12        2
2         1
Name: count, dtype: int64
```

In [17]:
```
data = data[data['Dropoff_month'] == 1]
```

In [18]:
```
data = data[data['Pickup_month'] == 1]
```

In [19]:
```
data = data[data['Pickup_year'] == 2023]
#drop all 2009,2022 values
```

In [20]:
```
data['Pickup_year'].value_counts()
#only 2023 jan now. Lets convert int day to name of the day
```

Out[20]:
```
Pickup_year
2023    63168
Name: count, dtype: int64
```

In [21]:
```
data['Day_Of_Ride_7'] = data.Pickup_day % 7
#1st jan 2023 is a sunday so = 1
```

In [22]:
```
day_mapping = {0: 'Sunday', 1: 'Monday', 2: 'Tuesday', 3: 'Wednesday', 4: 'Thursday', 5: 'Friday', 6: 'Saturday'}

data['Day_Of_Ride_char'] = data['Day_Of_Ride_7'].replace(day_mapping)
```

In [23]:
```
columns_toDrop = ['Day_Of_Ride_7','Dropoff_month','Pickup_month']
```

In [24]:
```
data = data.drop(columns=columns_toDrop)
data
```

Out[24]:

	VendorID	lpep_pickup_datetime	lpep_dropoff_datetime	PULocationID	PUBorough	PUZone	PUservice_zone	DOLocationID	DOBo
0	2	2023-01-01 00:26:00	2023-01-01 00:37:00	166	Manhattan	Morningside Heights	Boro Zone	143	Mar
1	2	2023-01-01 00:51:00	2023-01-01 00:57:00	24	Manhattan	Bloomingdale	Yellow Zone	43	Mar
2	2	2023-01-01 00:35:00	2023-01-01 00:41:00	223	Queens	Steinway	Boro Zone	179	C
3	1	2023-01-01 00:13:00	2023-01-01 00:19:00	41	Manhattan	Central Harlem	Boro Zone	238	Mar
4	1	2023-01-01 00:33:00	2023-01-01 00:39:00	41	Manhattan	Central Harlem	Boro Zone	74	Mar
...	...	...	...	...	...	...	...	...	
63882	2	2023-01-31 23:09:00	2023-01-31 23:17:00	130	Queens	Jamaica	Boro Zone	205	C
63883	2	2023-01-31 23:06:00	2023-01-31 23:17:00	65	Brooklyn	Downtown Brooklyn/MetroTech	Boro Zone	181	Br
63884	2	2023-01-31 23:17:00	2023-01-31 23:23:00	244	Manhattan	Washington Heights South	Boro Zone	116	Mar
63885	2	2023-01-31 23:29:00	2023-01-31 23:38:00	74	Manhattan	East Harlem North	Boro Zone	238	Mar
63886	2	2023-01-31 23:00:00	2023-01-31 23:10:00	95	Queens	Forest Hills	Boro Zone	95	C

63168 rows × 30 columns

In [25]:
```
col = ['lpep_pickup_datetime','lpep_dropoff_datetime']
data = data.drop(columns = col)
```

In [26]:
```
#no more null values , lets check for outliers using IQR
```

In [27]:
```
temp_data = data
```

In [28]:
```
temp_data = temp_data[temp_data['trip_distance'] < 0.26]
```

In [29]:
```
temp_data
```

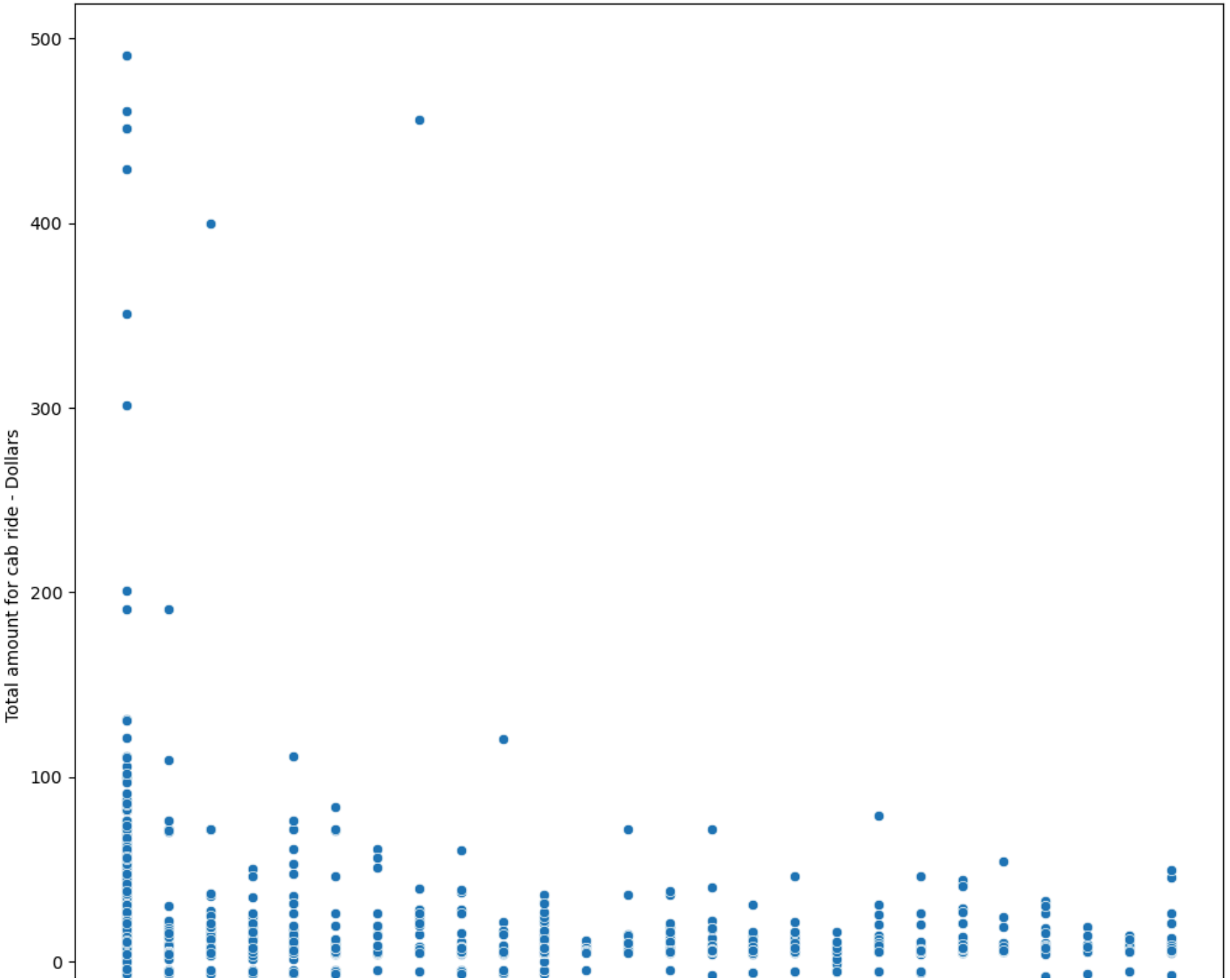
Out[29]:

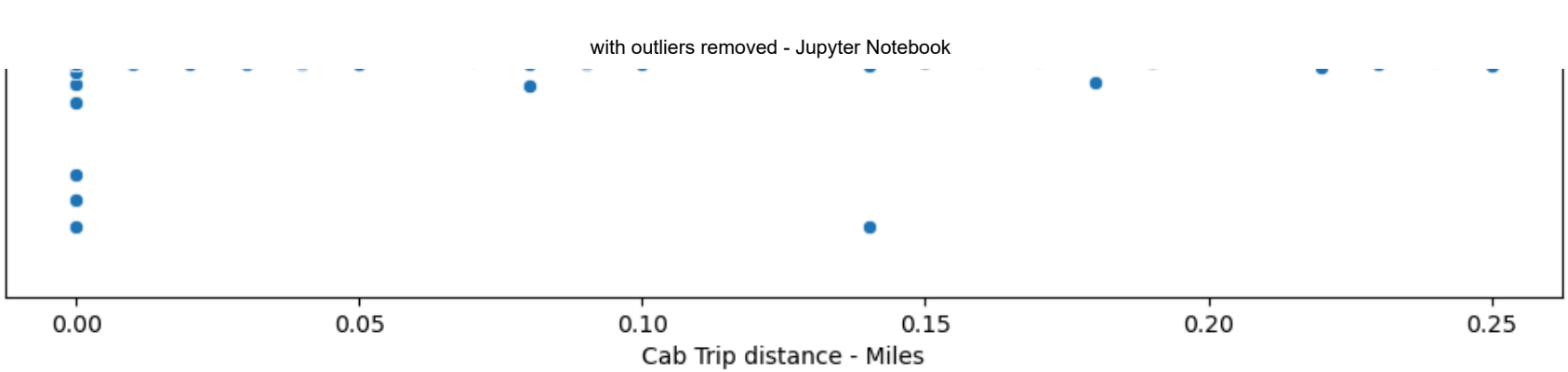
	VendorID	PULocationID	PUBorough	PUZone	PUservice_zone	DOLocationID	DOBorough	DOZone	DOservice_zone	passenger_co
2	2	223	Queens	Steinway	Boro Zone	179	Queens	Old Astoria	Boro Zone	
58	2	75	Manhattan	East Harlem South	Boro Zone	75	Manhattan	East Harlem South	Boro Zone	
99	2	168	Bronx	Mott Haven/Port Morris	Boro Zone	168	Bronx	Mott Haven/Port Morris	Boro Zone	
104	2	179	Queens	Old Astoria	Boro Zone	179	Queens	Old Astoria	Boro Zone	
107	2	7	Queens	Astoria	Boro Zone	7	Queens	Astoria	Boro Zone	
...	...	...	...	...	...	...	...	...	...	
63807	2	256	Brooklyn	Williamsburg (South Side)	Boro Zone	256	Brooklyn	Williamsburg (South Side)	Boro Zone	
63808	2	256	Brooklyn	Williamsburg (South Side)	Boro Zone	256	Brooklyn	Williamsburg (South Side)	Boro Zone	
63813	2	95	Queens	Forest Hills	Boro Zone	95	Queens	Forest Hills	Boro Zone	
63833	2	130	Queens	Jamaica	Boro Zone	130	Queens	Jamaica	Boro Zone	
63838	2	95	Queens	Forest Hills	Boro Zone	95	Queens	Forest Hills	Boro Zone	

3585 rows × 28 columns

In [30]:
```
plt.figure(figsize=(12, 12))
sns.scatterplot(data = temp_data, x = "trip_distance", y = "total_amount")
plt.xlabel("Cab Trip distance - Miles")
plt.ylabel("Total amount for cab ride - Dollars")
```

Out[30]:
```
Text(0, 0.5, 'Total amount for cab ride - Dollars')
```

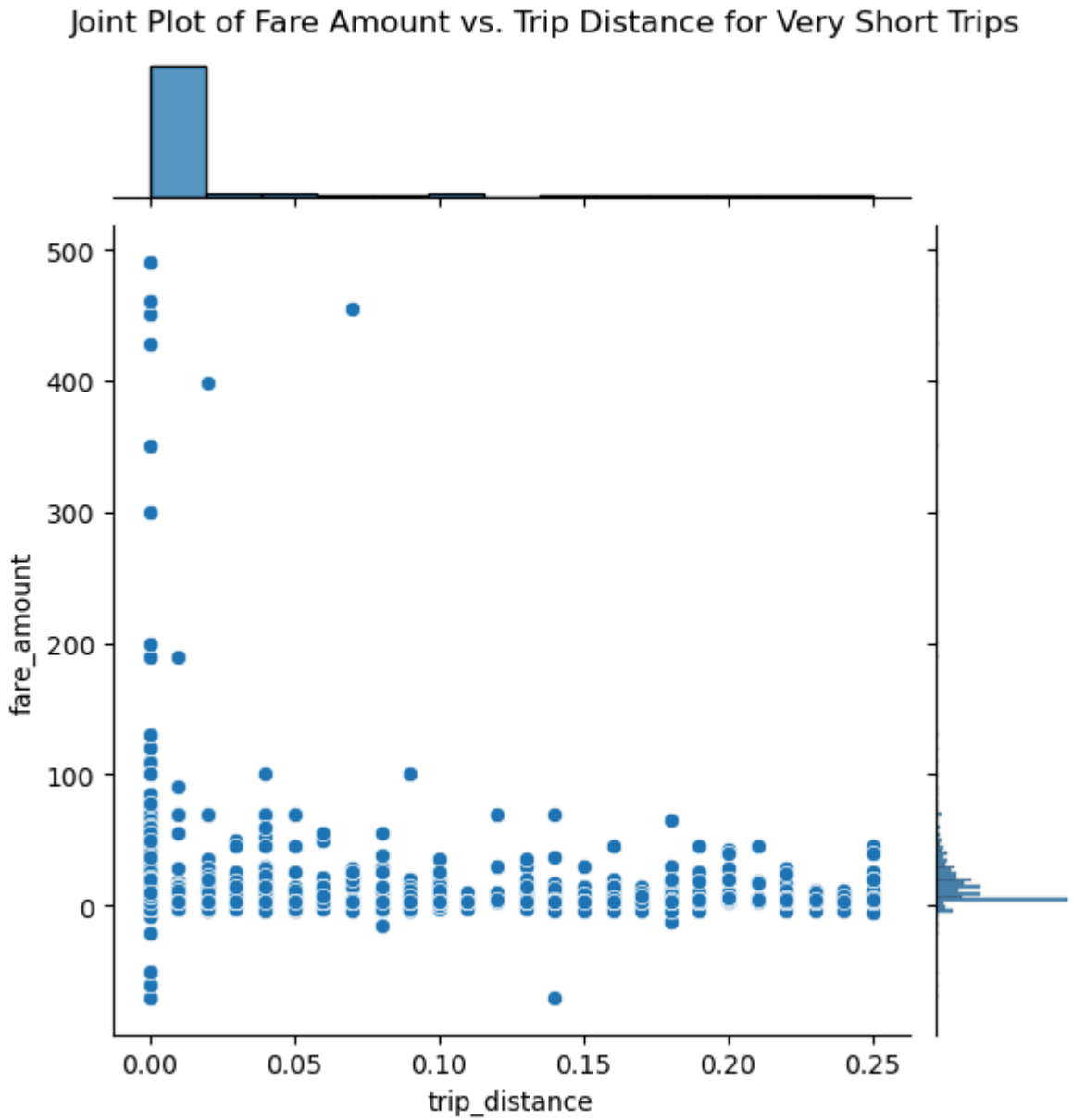

500
400
300
200
100
0
Total amount for cab ride - Dollars


0.00
0.05
0.10
0.15
0.20
0.25
Cab Trip distance - Miles

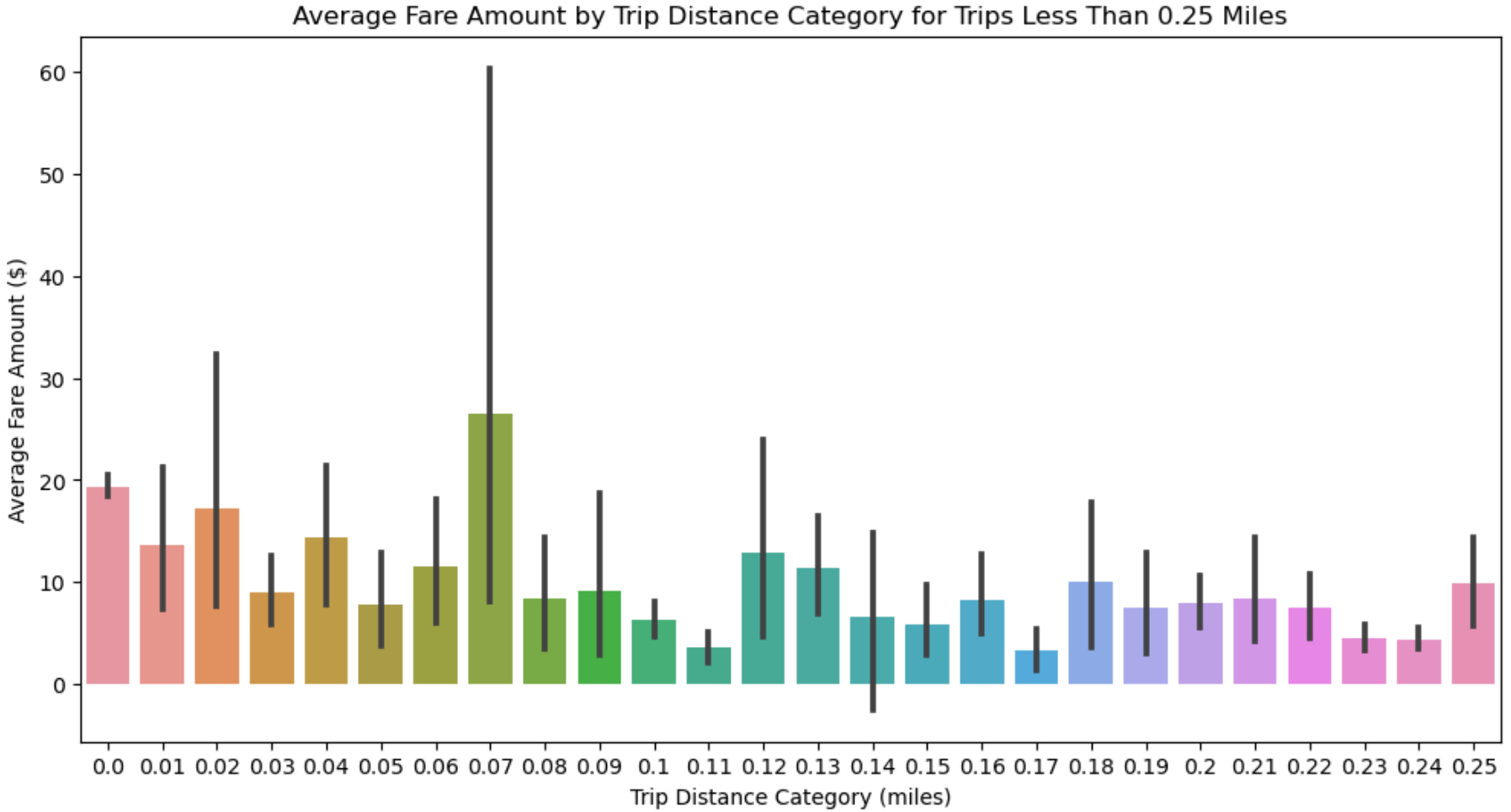
In [31]:

```
sns.jointplot(data=temp_data, x='trip_distance', y='fare_amount', kind='scatter')
plt.suptitle('Joint Plot of Fare Amount vs. Trip Distance for Very Short Trips', y=1.02)
plt.show()
```

```
C:\Users\AMIT\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\AMIT\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```


Joint Plot of Fare Amount vs. Trip Distance for Very Short Trips
fare_amount
trip_distance
500
400
300
200
100
0
0.00
0.05
0.10
0.15
0.20
0.25

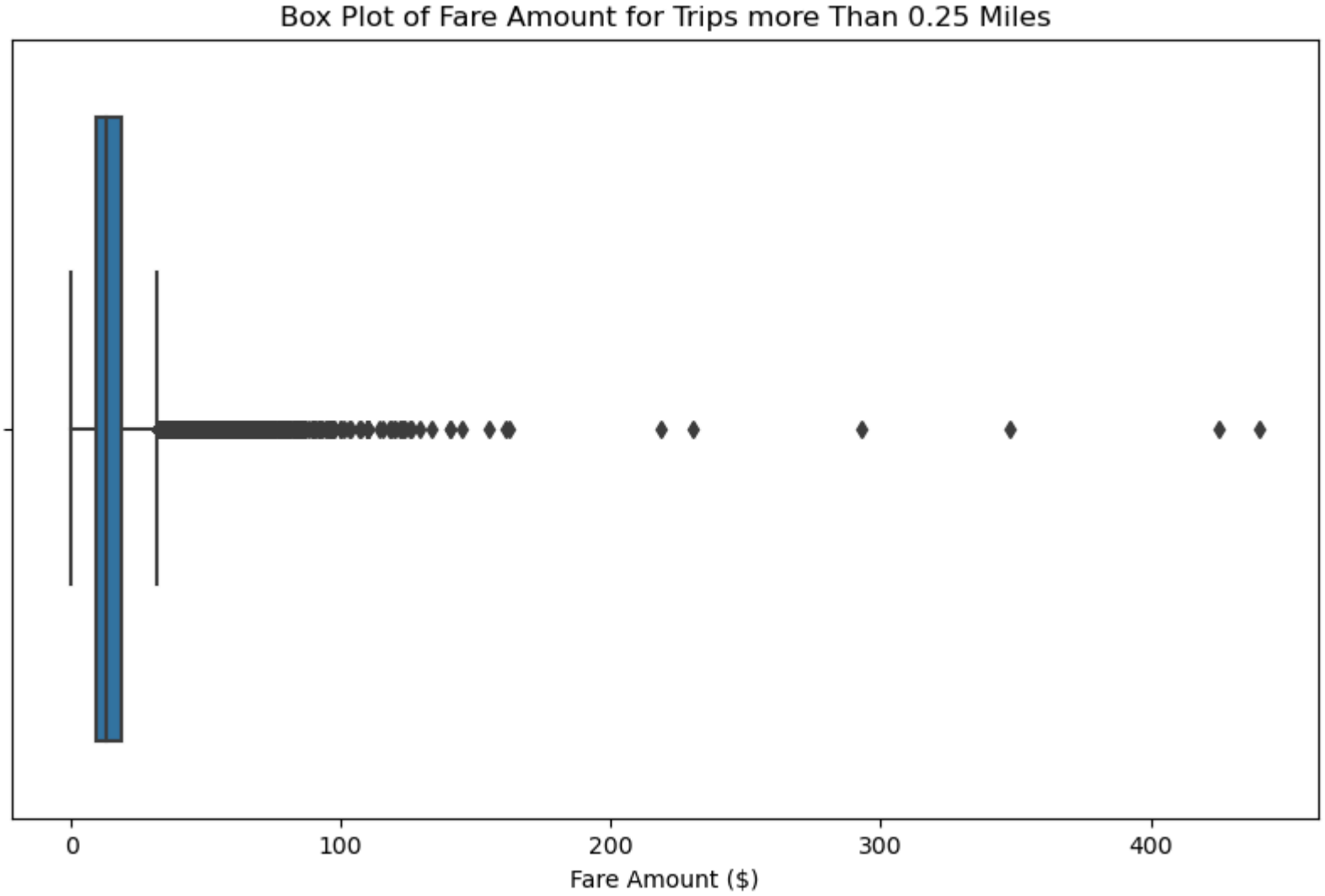
In [32]:
```
plt.figure(figsize=(12, 6))

sns.barplot(x='trip_distance', y='fare_amount', data=temp_data)
plt.title('Average Fare Amount by Trip Distance Category for Trips Less Than 0.25 Miles')
plt.xlabel('Trip Distance Category (miles)')
plt.ylabel('Average Fare Amount ($)')
plt.show()
```



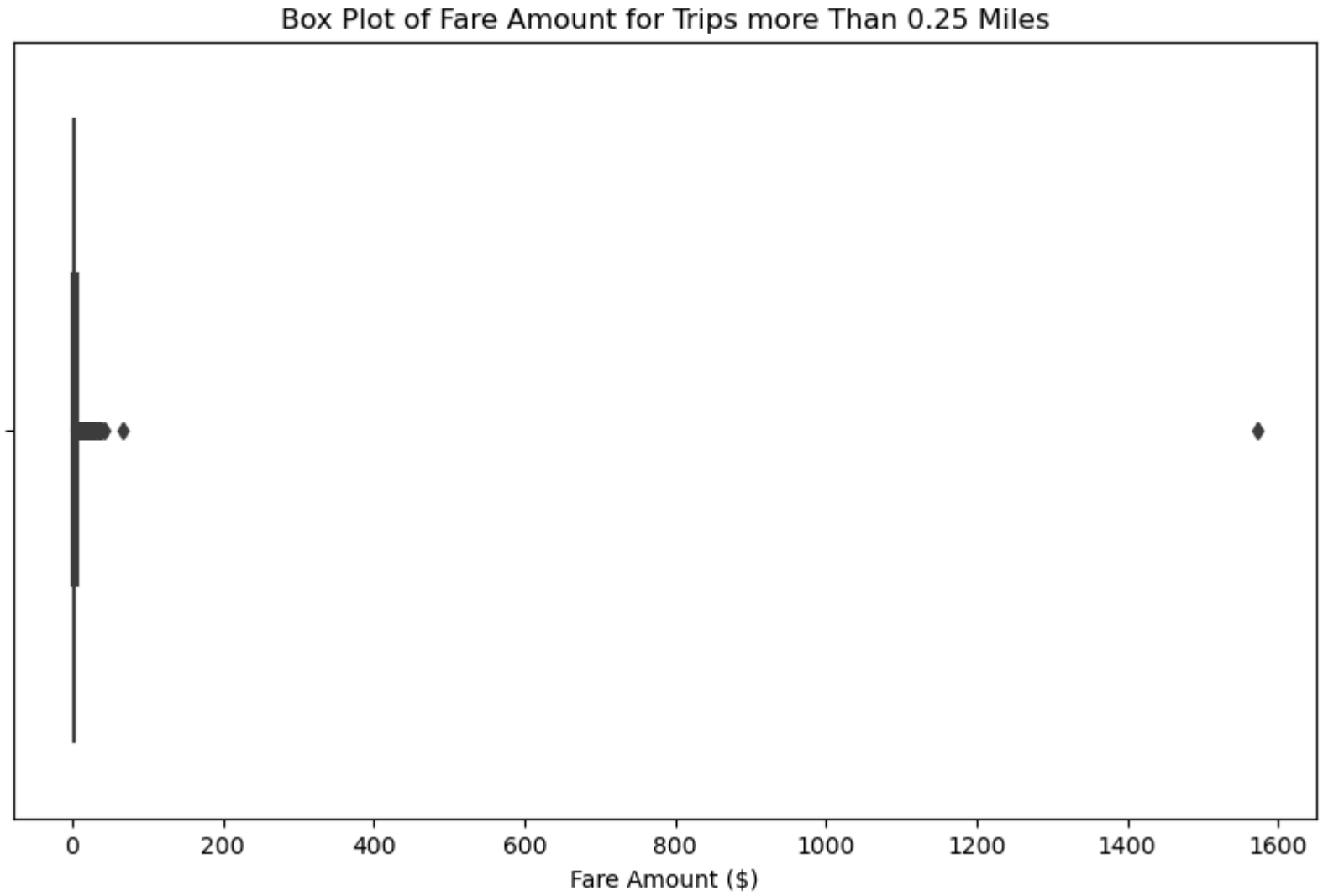
In [33]:
```
data = data[data['trip_distance'] > 0.26]
```

In [34]:
```
data = data[data['fare_amount'] > 0]
```

In [35]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(x=data['fare_amount'])
plt.title('Box Plot of Fare Amount for Trips more Than 0.25 Miles')
plt.xlabel('Fare Amount ($)')
plt.show()
```



In [36]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(x=data['trip_distance'])
plt.title('Box Plot of Fare Amount for Trips more Than 0.25 Miles')
plt.xlabel('Fare Amount ($)')
plt.show()
```



In [37]:
```
data.shape
```

Out[37]:
```
(59503, 28)
```

In [38]:
```
data = data[(data['fare_amount'] < 150)]
```

In [39]:
```
data = data[(data['fare_amount'] < 150)]
```

In [40]:
```
data = data[(data['fare_amount'] < 70)]
```

In [41]:
```
data = data[(data['trip_distance'] < 66.18)]
```

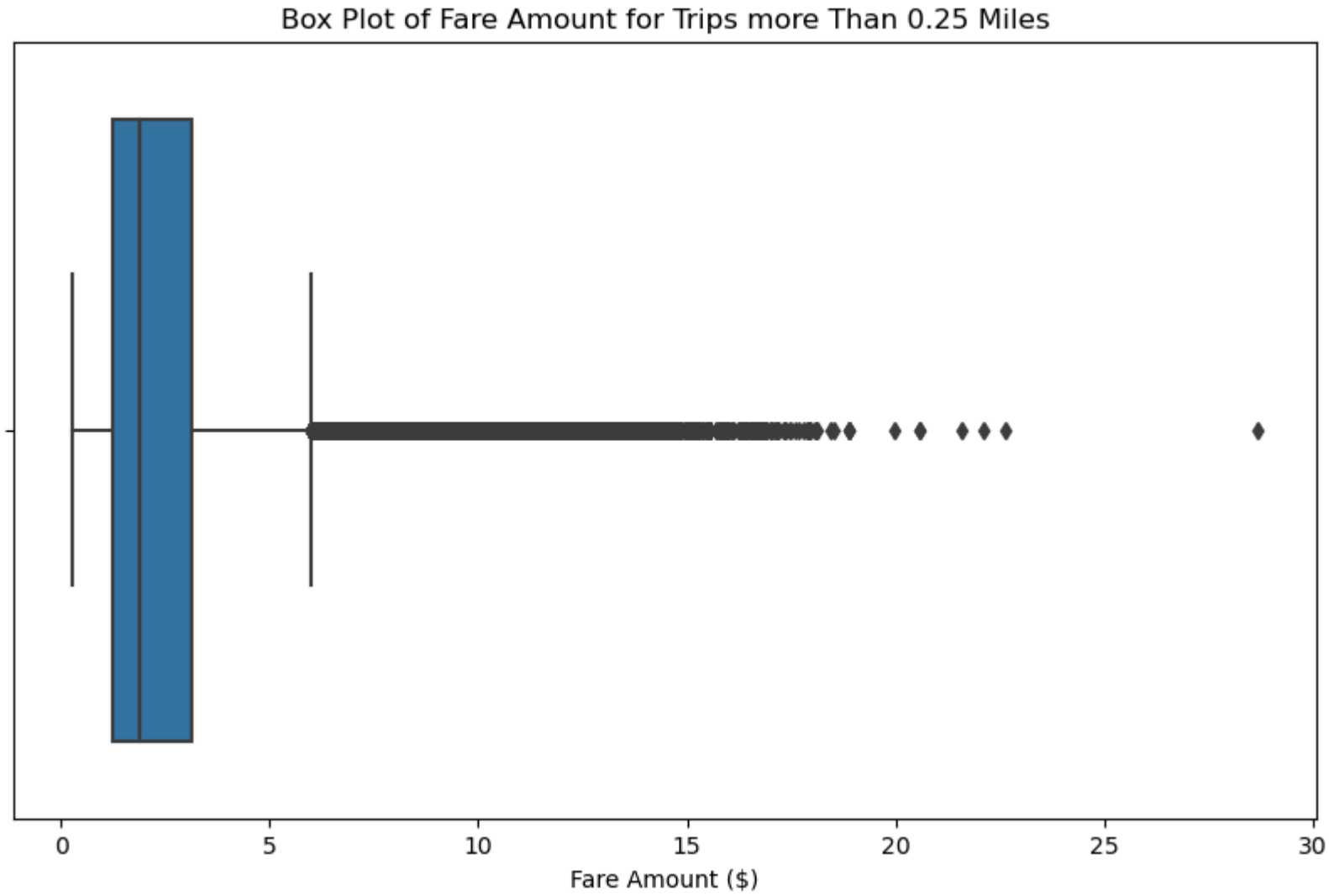
In [42]:
```
data
```

Out[42]:

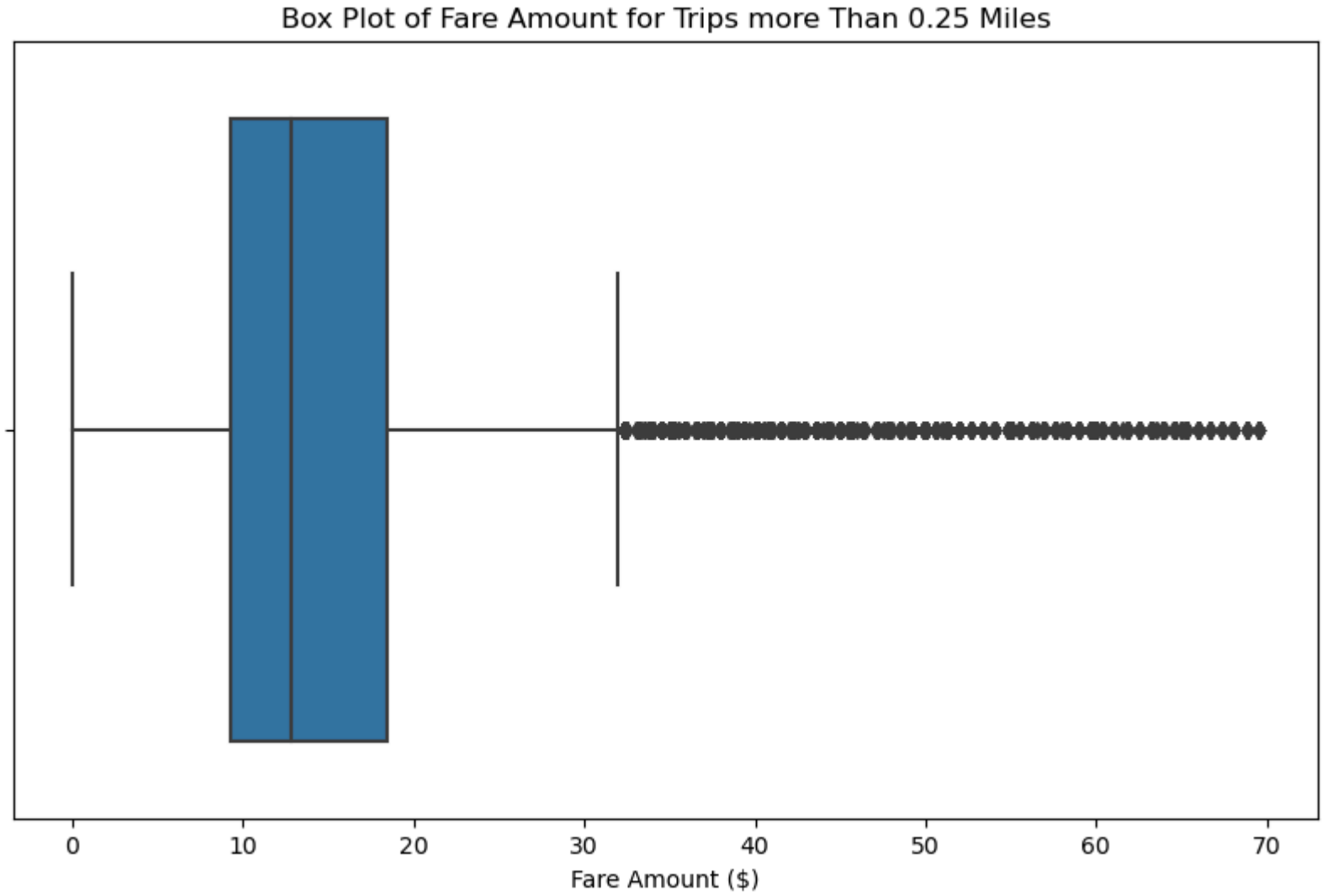
	VendorID	PULocationID	PUBorough	PUZone	PUservice_zone	DOLocationID	DOBorough	DOZone	DOservice_zone	passenger_
0	2	166	Manhattan	Morningside Heights	Boro Zone	143	Manhattan	Lincoln Square West	Yellow Zone	
1	2	24	Manhattan	Bloomingdale	Yellow Zone	43	Manhattan	Central Park	Yellow Zone	
3	1	41	Manhattan	Central Harlem	Boro Zone	238	Manhattan	Upper West Side North	Yellow Zone	
4	1	41	Manhattan	Central Harlem	Boro Zone	74	Manhattan	East Harlem North	Boro Zone	
5	2	41	Manhattan	Central Harlem	Boro Zone	262	Manhattan	Yorkville East	Yellow Zone	
...	...	...	...	...	...	...	...	...	...	...
63882	2	130	Queens	Jamaica	Boro Zone	205	Queens	Saint Albans	Boro Zone	
63883	2	65	Brooklyn	Downtown Brooklyn/MetroTech	Boro Zone	181	Brooklyn	Park Slope	Boro Zone	
63884	2	244	Manhattan	Washington Heights South	Boro Zone	116	Manhattan	Hamilton Heights	Boro Zone	
63885	2	74	Manhattan	East Harlem North	Boro Zone	238	Manhattan	Upper West Side North	Yellow Zone	
63886	2	95	Queens	Forest Hills	Boro Zone	95	Queens	Forest Hills	Boro Zone	

59158 rows × 28 columns

In [43]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(x=data['trip_distance'])
plt.title('Box Plot of Fare Amount for Trips more Than 0.25 Miles')
plt.xlabel('Fare Amount ($)')
plt.show()
```



In [44]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(x=data['fare_amount'])
plt.title('Box Plot of Fare Amount for Trips more Than 0.25 Miles')
plt.xlabel('Fare Amount ($)')
plt.show()
```



In [45]:
```
data['passenger_count'].value_counts()
```

Out[45]:
```
passenger_count
1.0    50204
2.0     4730
5.0     1722
6.0     1011
3.0      868
4.0      345
0.0      277
7.0        1
Name: count, dtype: int64
```

In [46]:
```
data.to_csv('filename.csv', index=False)
```

In [47]:
```
data.shape
```

Out[47]:
```
(59158, 28)
```

In [48]:
```
data_vendor1 = data[data['VendorID'] == 1]
```

In [49]:
```
data_vendor2 = data[data['VendorID'] == 2]
```

In [50]:
```
data_vendor1
```

Out[50]:

	VendorID	PULocationID	PUBorough	PUZone	PUservice_zone	DOLocationID	DOBorough	DOZone	DOservice_zone	passenge
3	1	41	Manhattan	Central Harlem	Boro Zone	238	Manhattan	Upper West Side North	Yellow Zone	
4	1	41	Manhattan	Central Harlem	Boro Zone	74	Manhattan	East Harlem North	Boro Zone	
6	1	181	Brooklyn	Park Slope	Boro Zone	45	Manhattan	Chinatown	Yellow Zone	
10	1	255	Brooklyn	Williamsburg (North Side)	Boro Zone	234	Manhattan	Union Sq	Yellow Zone	
15	1	195	Brooklyn	Red Hook	Boro Zone	210	Brooklyn	Sheepshead Bay	Boro Zone	
...	...	...	...	...	...	...	...	...	...	...
63823	1	129	Queens	Jackson Heights	Boro Zone	83	Queens	Elmhurst/Maspeth	Boro Zone	
63840	1	25	Brooklyn	Boerum Hill	Boro Zone	49	Brooklyn	Clinton Hill	Boro Zone	
63843	1	74	Manhattan	East Harlem North	Boro Zone	24	Manhattan	Bloomingdale	Yellow Zone	
63844	1	166	Manhattan	Morningside Heights	Boro Zone	244	Manhattan	Washington Heights South	Boro Zone	
63846	1	181	Brooklyn	Park Slope	Boro Zone	181	Brooklyn	Park Slope	Boro Zone	

7549 rows × 28 columns

In [51]:
```
data_vendor2
```

Out[51]:

	VendorID	PULocationID	PUBorough	PUZone	PUservice_zone	DOLocationID	DOBorough	DOZone	DOservice_zone	passeng
0	2	166	Manhattan	Morningside Heights	Boro Zone	143	Manhattan	Lincoln Square West	Yellow Zone	
1	2	24	Manhattan	Bloomingdale	Yellow Zone	43	Manhattan	Central Park	Yellow Zone	
5	2	41	Manhattan	Central Harlem	Boro Zone	262	Manhattan	Yorkville East	Yellow Zone	
7	2	24	Manhattan	Bloomingdale	Yellow Zone	75	Manhattan	East Harlem South	Boro Zone	
8	2	41	Manhattan	Central Harlem	Boro Zone	166	Manhattan	Morningside Heights	Boro Zone	
...	...	...	...	...	...	...	...	...	...	...
63882	2	130	Queens	Jamaica	Boro Zone	205	Queens	Saint Albans	Boro Zone	
63883	2	65	Brooklyn	Downtown Brooklyn/MetroTech	Boro Zone	181	Brooklyn	Park Slope	Boro Zone	
63884	2	244	Manhattan	Washington Heights South	Boro Zone	116	Manhattan	Hamilton Heights	Boro Zone	
63885	2	74	Manhattan	East Harlem North	Boro Zone	238	Manhattan	Upper West Side North	Yellow Zone	
63886	2	95	Queens	Forest Hills	Boro Zone	95	Queens	Forest Hills	Boro Zone	

51609 rows × 28 columns

In [52]:
```
data_vendor1.describe()
```

Out[52]:

	VendorID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount	extra	mta_tax	tip_amount	tolls_amount
count	7549.0	7549.000000	7549.000000	7549.000000	7549.000000	7549.000000	7549.000000	7549.000000	7549.000000	7549.000000
mean	1.0	95.064114	139.308783	1.131938	2.585349	14.305999	1.517254	1.490131	1.964285	0.086111
std	0.0	58.193190	78.759343	0.530294	2.188011	8.775822	1.653469	0.107985	2.450913	0.756506
min	1.0	7.000000	1.000000	0.000000	0.300000	0.010000	0.000000	0.000000	0.000000	0.000000
25%	1.0	74.000000	74.000000	1.000000	1.300000	8.600000	0.000000	1.500000	0.000000	0.000000
50%	1.0	74.000000	141.000000	1.000000	1.900000	12.100000	1.000000	1.500000	1.640000	0.000000
75%	1.0	97.000000	229.000000	1.000000	3.100000	17.000000	2.750000	1.500000	3.200000	0.000000
max	1.0	263.000000	263.000000	6.000000	22.100000	69.500000	7.500000	1.500000	57.000000	14.750000

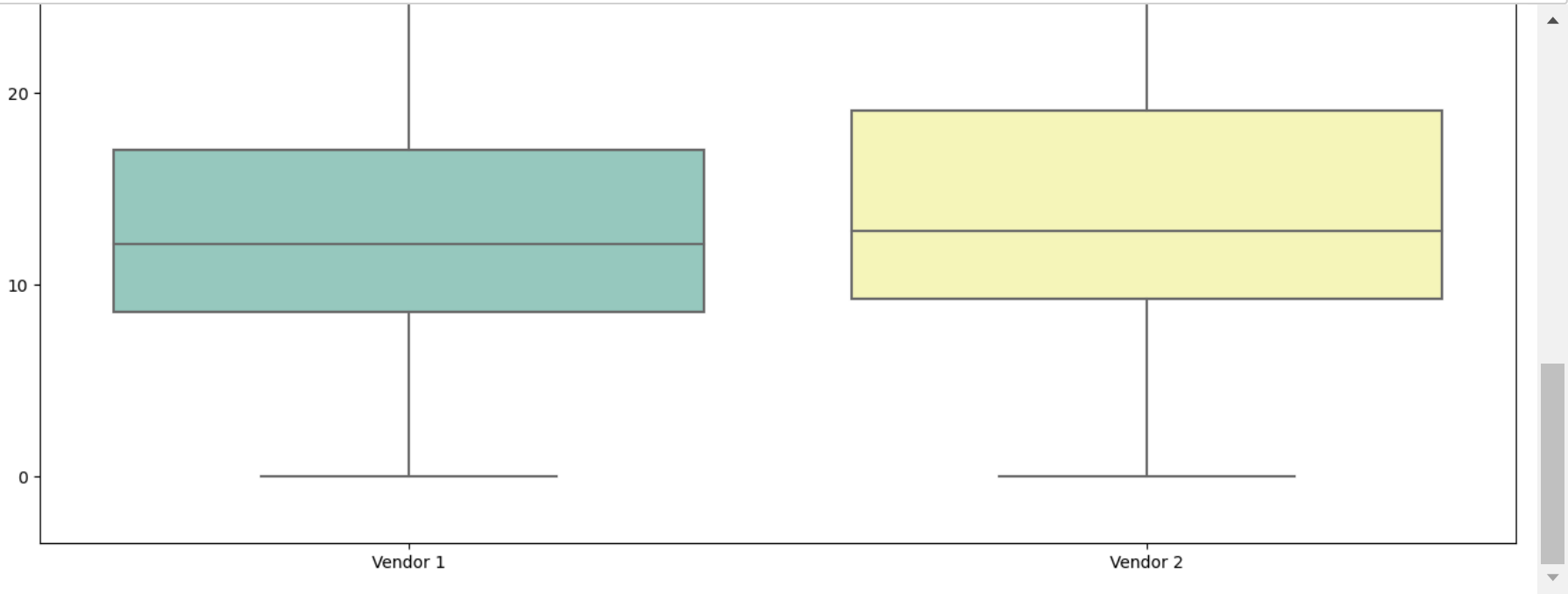
In [53]:
```
data_vendor2.describe()
```

Out[53]:

	VendorID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount	extra	mta_tax	tip_amount	tolls_amo
count	51609.0	51609.000000	51609.000000	51609.000000	51609.000000	51609.000000	51609.000000	51609.000000	51609.000000	51609.000
mean	2.0	96.295956	137.838768	1.352206	2.598625	15.556483	0.834273	0.493974	2.093169	0.108
std	0.0	58.524227	76.014936	1.039737	2.207262	9.424846	1.222279	0.054560	2.687973	0.830
min	2.0	3.000000	1.000000	0.000000	0.270000	0.050000	0.000000	0.000000	0.000000	0.000
25%	2.0	74.000000	74.000000	1.000000	1.220000	9.300000	0.000000	0.500000	0.000000	0.000
50%	2.0	75.000000	138.000000	1.000000	1.890000	12.800000	0.000000	0.500000	1.740000	0.000
75%	2.0	112.000000	216.000000	1.000000	3.140000	19.100000	1.000000	0.500000	3.330000	0.000
max	2.0	263.000000	263.000000	7.000000	28.670000	69.500000	7.500000	0.500000	80.880000	13.100

In [ ]:
```

```

In [54]:
```
plt.figure(figsize=(16, 16))
sns.boxplot(data=[data_vendor1['fare_amount'], data_vendor2['fare_amount']], palette='Set3')
plt.xticks([0, 1], ['Vendor 1', 'Vendor 2'])
plt.title('Comparison of Fare Amounts Between Vendors')
plt.ylabel('Fare Amount ($)')
plt.show()
```



The insights are in PPT

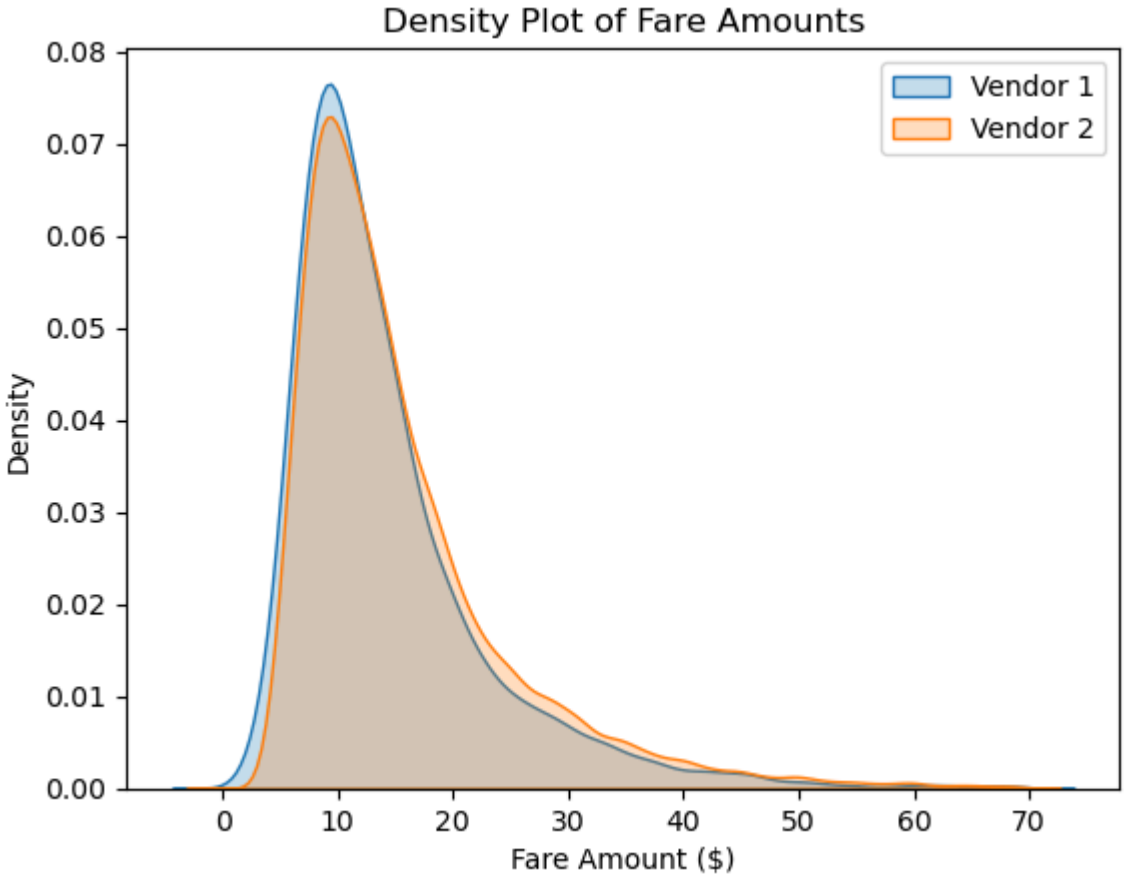
Vendor 1 has a lower median fare amount compared to Vendor 2. This is evident from the position of the median line within each box

The interquartile range (IQR), which is the range between the 25th and 75th percentiles and represented by the height of each box, is narrower

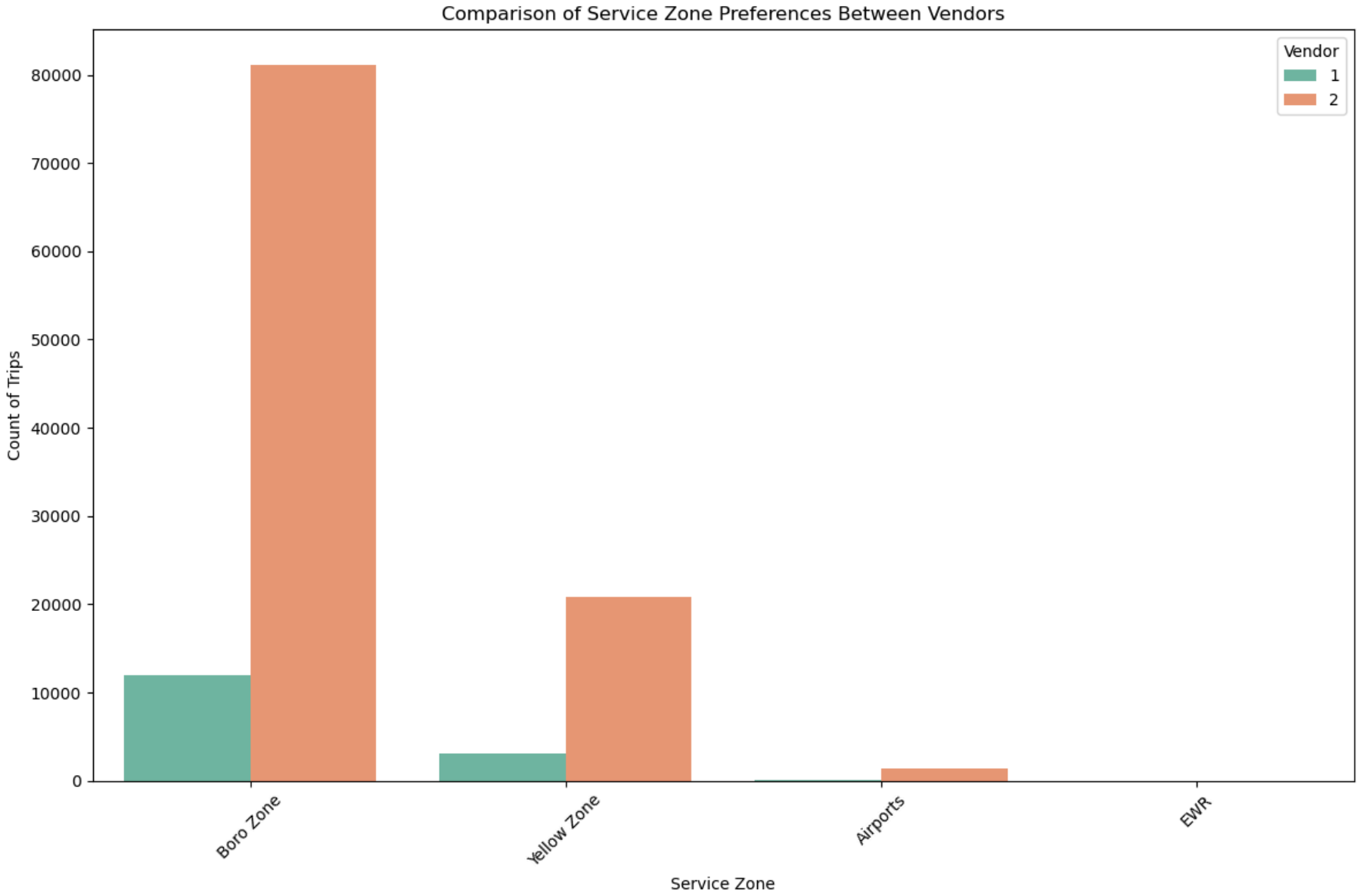
for Vendor 1. This suggests that Vendor 1 has less variability in fare amounts compared to Vendor 2

In [55]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.kdeplot(data_vendor1['fare_amount'], label='Vendor 1', fill=True)
sns.kdeplot(data_vendor2['fare_amount'], label='Vendor 2', fill=True)
plt.title('Density Plot of Fare Amounts')
plt.xlabel('Fare Amount ($)')
plt.legend()
plt.show()
```

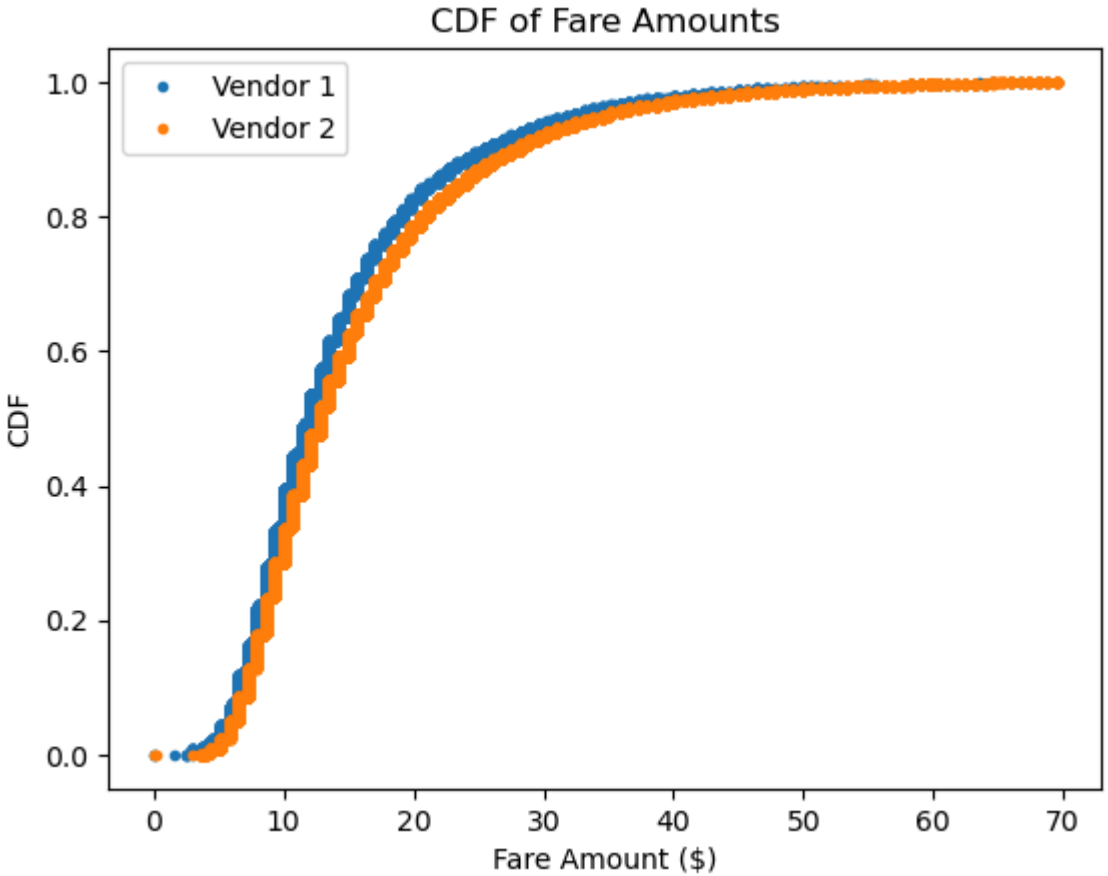
```
C:\Users\AMIT\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\AMIT\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```


Density Plot of Fare Amounts
Vendor 1
Vendor 2
Density
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0
10
20
30
40
50
60
70
Fare Amount ($)

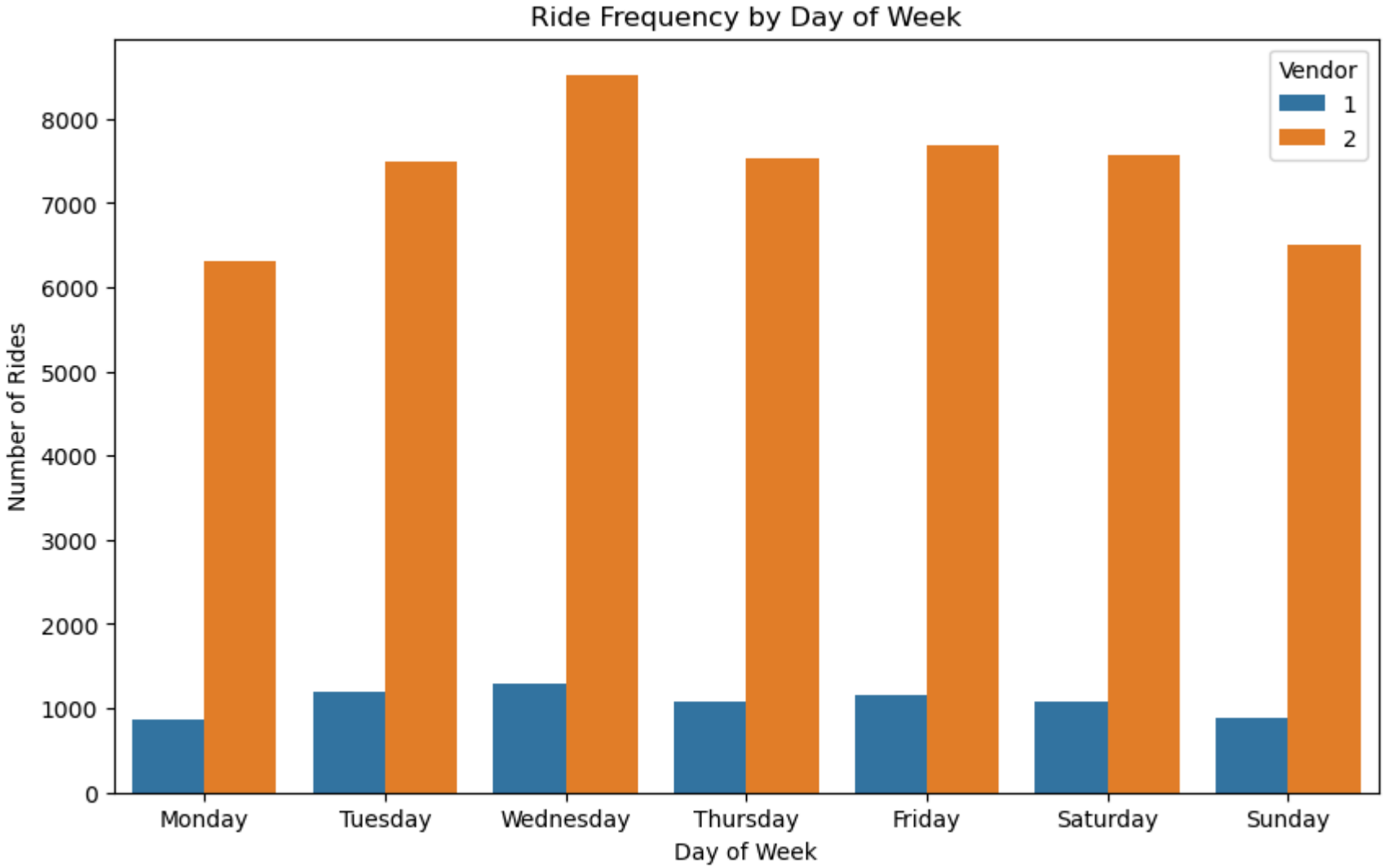
In [56]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'data' is your combined DataFrame and includes a 'VendorID' or similar column
# and 'PUservice_zone' and 'DOservice_zone' for the service zones

# Melt the dataframe to make 'PUservice_zone' and 'DOservice_zone' in one column
data_melted = data.melt(id_vars=['VendorID'], value_vars=['PUservice_zone', 'DOservice_zone'],
                        var_name='Zone_Type', value_name='Service_Zone')

# Create a count plot
plt.figure(figsize=(12, 8))
sns.countplot(data=data_melted, x='Service_Zone', hue='VendorID', palette='Set2',
              order=data_melted['Service_Zone'].value_counts().index)
plt.title('Comparison of Service Zone Preferences Between Vendors')
plt.xlabel('Service Zone')
plt.ylabel('Count of Trips')
plt.xticks(rotation=45)
plt.legend(title='Vendor')
plt.tight_layout()
plt.show()
```


Comparison of Service Zone Preferences Between Vendors
Vendor
1
2
Count of Trips
0
10000
20000
30000
40000
50000
60000
70000
80000
Boro Zone
Yellow Zone
Airports
EWR
Service Zone

In [57]:
```
import numpy as np
import matplotlib.pyplot as plt

# Calculate the CDF for vendor 1
x1 = np.sort(data_vendor1['fare_amount'])
y1 = np.arange(1, len(x1)+1) / len(x1)

# Calculate the CDF for vendor 2
x2 = np.sort(data_vendor2['fare_amount'])
y2 = np.arange(1, len(x2)+1) / len(x2)

plt.plot(x1, y1, marker='.', linestyle='none', label='Vendor 1')
plt.plot(x2, y2, marker='.', linestyle='none', label='Vendor 2')
plt.title('CDF of Fare Amounts')
plt.xlabel('Fare Amount ($)')
plt.ylabel('CDF')
plt.legend()
plt.show()
```

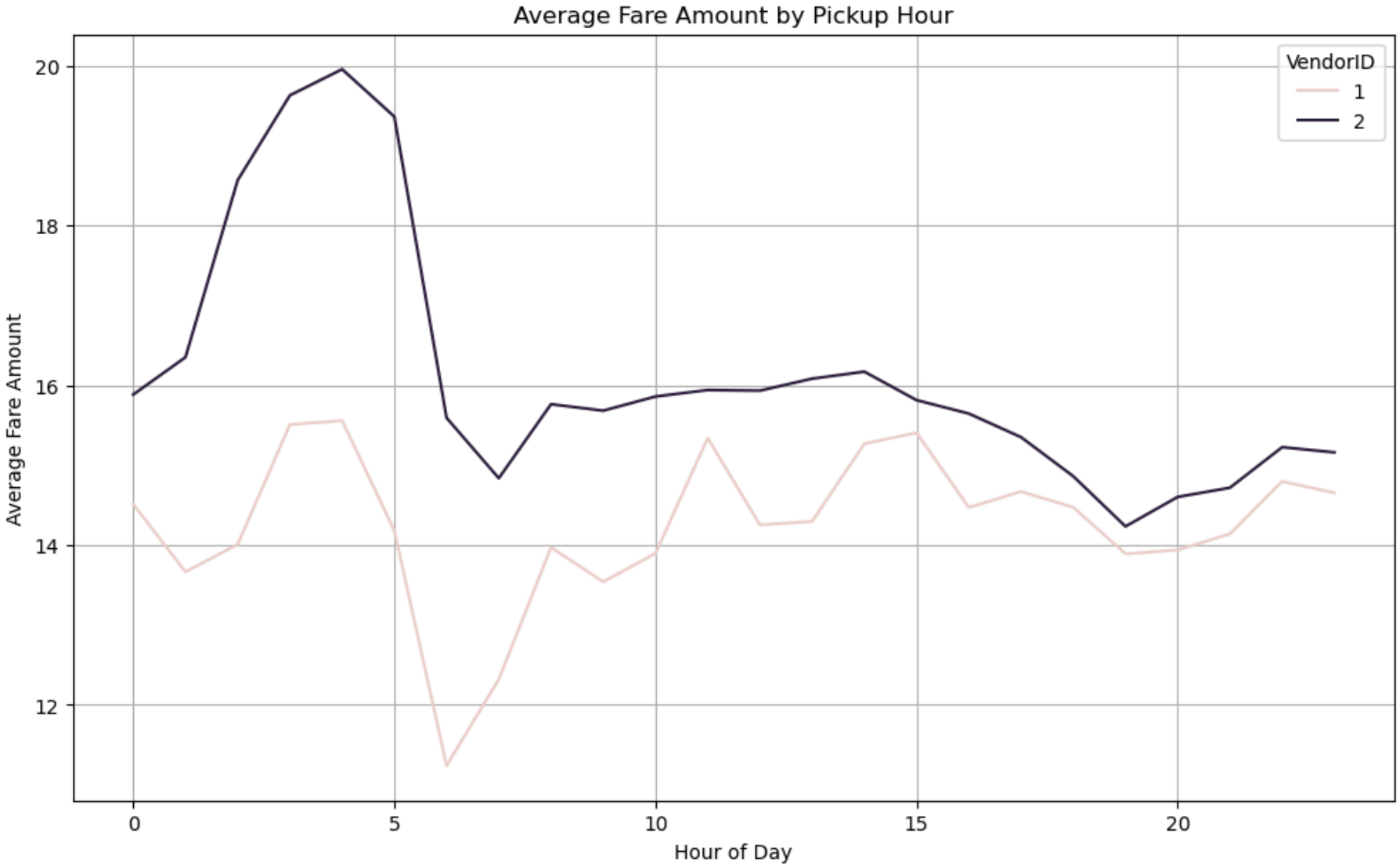

CDF of Fare Amounts
Vendor 1
Vendor 2
CDF
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
50
60
70
Fare Amount ($)

In [58]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is your DataFrame
plt.figure(figsize=(10, 6))
sns.countplot(data=data, x='Day_Of_Ride_char', order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturda
plt.title('Ride Frequency by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Number of Rides')
plt.legend(title='Vendor')
plt.show()
```

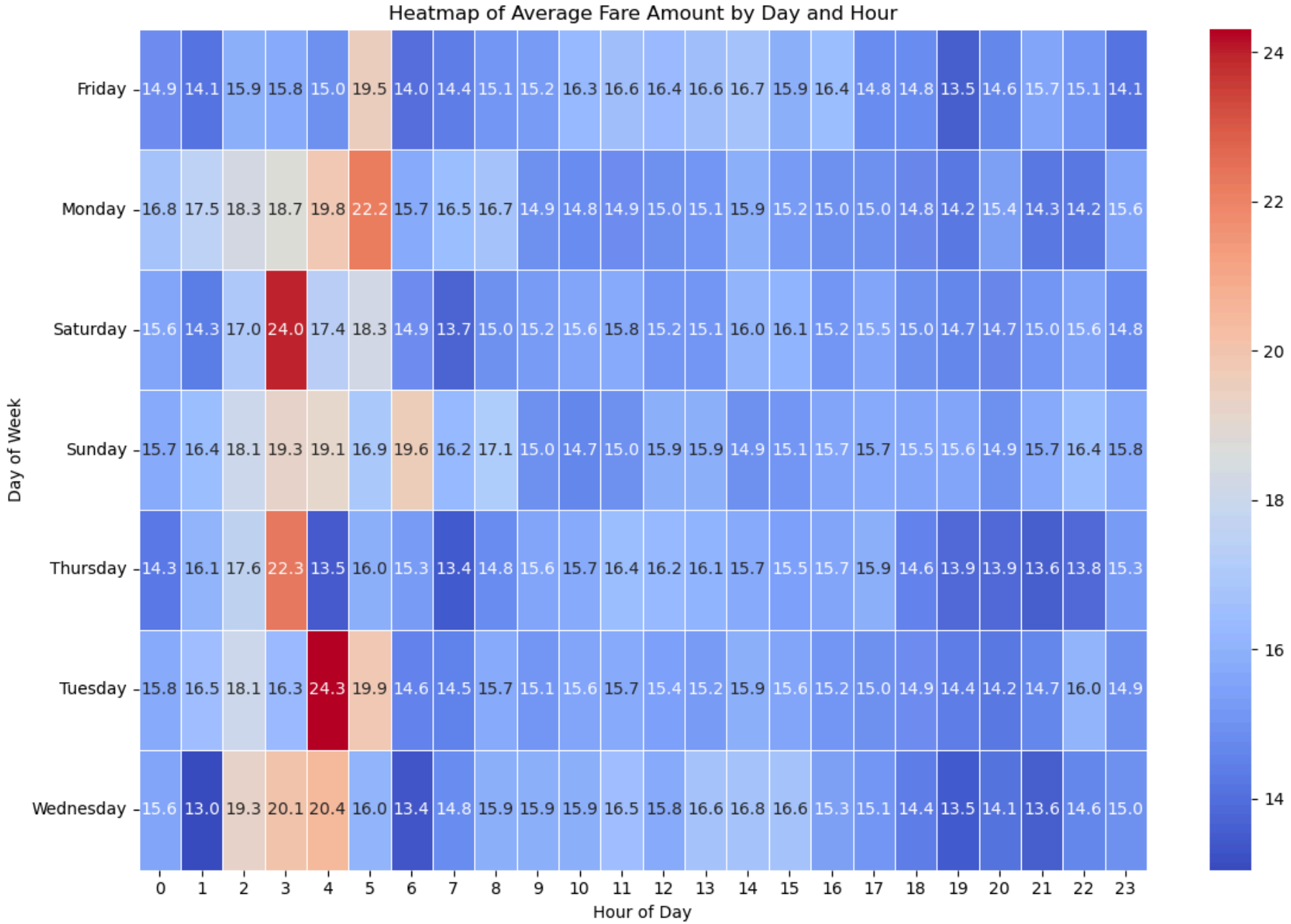

Ride Frequency by Day of Week
Vendor
1
2
Number of Rides
0
1000
2000
3000
4000
5000
6000
7000
8000
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day of Week

In [59]:
```
import pandas as pd

# Assuming 'data' is your DataFrame and 'Pickup_time' is a column of type datetime.time or formatted as 'HH:MM:SS'
# Convert time to string and extract the hour part
data['Pickup_hour'] = data['Pickup_time'].astype(str).str.extract(r'^(\d+)', expand=False).astype(int)

# Now you can plot the average fare amount by pickup hour
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 7))
sns.lineplot(data=data, x='Pickup_hour', y='fare_amount', hue='VendorID', ci=None)
plt.title('Average Fare Amount by Pickup Hour')
plt.xlabel('Hour of Day')
plt.ylabel('Average Fare Amount')
plt.grid(True)
plt.show()
```

```
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\3618432694.py:12: FutureWarning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(data=data, x='Pickup_hour', y='fare_amount', hue='VendorID', ci=None)
C:\Users\AMIT\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\AMIT\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated
and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\AMIT\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning: When grouping with a length-1 list
-like, you will need to pass a length-1 tuple to get_group in a future version of pandas. Pass `(name,)` instead of `
name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
C:\Users\AMIT\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning: When grouping with a length-1 list
-like, you will need to pass a length-1 tuple to get_group in a future version of pandas. Pass `(name,)` instead of `
name` to silence this warning.
  data_subset = grouped_data.get_group(pd_key)
```


Average Fare Amount by Pickup Hour
VendorID
1
2
Average Fare Amount
20
18
16
14
12
0
5
10
15
20
Hour of Day

In [60]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'data' is your combined DataFrame and it includes 'Pickup_time', 'Day_Of_Ride_char', and 'fare_amount'
# If 'Pickup_time' is a string or in a different format, convert it to datetime and extract the hour
data['Pickup_hour'] = pd.to_datetime(data['Pickup_time'], format='%H:%M:%S', errors='coerce').dt.hour

# Ensure 'Pickup_hour' is not null after conversion
data = data[data['Pickup_hour'].notna()]

# Group data by day of the ride and pickup hour, then calculate the average fare amount
grouped_data = data.groupby(['Day_Of_Ride_char', 'Pickup_hour']).agg({'fare_amount': 'mean'}).reset_index()

# Pivot the data for heatmap creation
pivot_table = grouped_data.pivot(index='Day_Of_Ride_char', columns='Pickup_hour', values='fare_amount')

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(pivot_table, annot=True, cmap='coolwarm', fmt=".1f", linewidths=.5)
plt.title('Heatmap of Average Fare Amount by Day and Hour')
plt.xlabel('Hour of Day')
plt.ylabel('Day of Week')
plt.yticks(rotation=0)  # Rotate y-axis labels for better readability
plt.tight_layout()  # Adjust layout to fit everything nicely
plt.show()
```


Heatmap of Average Fare Amount by Day and Hour
Day of Week
Hour of Day
Friday 14.9 14.1 15.9 15.8 15.0 19.5 14.0 14.4 15.1 15.2 16.3 16.6 16.4 16.6 16.7 15.9 16.4 14.8 14.8 13.5 14.6 15.7 15.1 14.1
Monday 16.8 17.5 18.3 18.7 19.8 22.2 15.7 16.5 16.7 14.9 14.8 14.9 15.0 15.1 15.9 15.2 15.0 15.0 14.8 14.2 15.4 14.3 14.2 15.6
Saturday 15.6 14.3 17.0 24.0 17.4 18.3 14.9 13.7 15.0 15.2 15.6 15.8 15.2 15.1 16.0 16.1 15.2 15.5 15.0 14.7 14.7 15.0 15.6 14.8
Sunday 15.7 16.4 18.1 19.3 19.1 16.9 19.6 16.2 17.1 15.0 14.7 15.0 15.9 15.9 14.9 15.1 15.7 15.7 15.5 15.6 14.9 15.7 16.4 15.8
Thursday 14.3 16.1 17.6 22.3 13.5 16.0 15.3 13.4 14.8 15.6 15.7 16.4 16.2 16.1 15.7 15.5 15.7 15.9 14.6 13.9 13.9 13.6 13.8 15.3
Tuesday 15.8 16.5 18.1 16.3 24.3 19.9 14.6 14.5 15.7 15.1 15.6 15.7 15.4 15.2 15.9 15.6 15.2 15.0 14.9 14.4 14.2 14.7 16.0 14.9
Wednesday 15.6 13.0 19.3 20.1 20.4 16.0 13.4 14.8 15.9 15.9 15.9 16.5 15.8 16.6 16.8 16.6 15.3 15.1 14.4 13.5 14.1 13.6 14.6 15.0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 22 20 18 16 14

hypothesis testing

In [61]:
```
vendor1_fares = data[data['VendorID'] == 1]['fare_amount']
vendor2_fares = data[data['VendorID'] == 2]['fare_amount']
```

Check for Normality

In [62]:
```
from scipy.stats import shapiro

# Normality test for Vendor 1
stat, p = shapiro(vendor1_fares.sample(min(5000, len(vendor1_fares))))  # sample due to Shapiro test limitations
print('Vendor 1 Fares Normality Test: Statistics=%.3f, p=%.3f' % (stat, p))

# Normality test for Vendor 2
stat, p = shapiro(vendor2_fares.sample(min(5000, len(vendor2_fares))))
print('Vendor 2 Fares Normality Test: Statistics=%.3f, p=%.3f' % (stat, p))
```

```
Vendor 1 Fares Normality Test: Statistics=0.824, p=0.000
Vendor 2 Fares Normality Test: Statistics=0.831, p=0.000
```

In [63]:
```
from scipy.stats import kstest, norm

# Normalize data and perform K-S test
normalized_data = (vendor1_fares - vendor1_fares.mean()) / vendor1_fares.std()
ks_statistic, p_value = kstest(normalized_data, 'norm')
print(f'KS Statistic: {ks_statistic}, p-value: {p_value}')
```

```
KS Statistic: 0.1533727997266776, p-value: 1.5963338994660176e-155
```

In [64]:
```
from scipy.stats import normaltest

stat, p = normaltest(vendor1_fares.sample(min(5000, len(vendor1_fares))))
print(f'D\'Agostino\'s K^2 Test Statistic: {stat}, p-value: {p}')
```

```
D'Agostino's K^2 Test Statistic: 2104.985345793599, p-value: 0.0
```

In [65]:
```
from scipy.stats import anderson

result = anderson(vendor1_fares)
print('Statistic: %.3f' % result.statistic)
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < cv:
        print('Probably normal at the %.1f%% level' % sl)
    else:
        print('Probably not normal at the %.1f%% level' % sl)
```

```
Statistic: 348.567
Probably not normal at the 15.0% level
Probably not normal at the 10.0% level
Probably not normal at the 5.0% level
Probably not normal at the 2.5% level
Probably not normal at the 1.0% level
```

In [166]:
```
vendor2_fares
```

Out[166]:
```
0        14.9
1        10.7
5        17.7
7        14.2
8         7.2
         ...
63882    15.0
63883    13.5
63884     9.3
63885    13.5
63886    11.4
Name: fare_amount, Length: 51609, dtype: float64
```

In [170]:
```
import pandas as pd
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.anova import anova_lm


# Debug print to check contents of the arrays
print("Vendor 1 Fares:", vendor1_fares)
print("Vendor 2 Fares:", vendor2_fares)

# Convert to pandas Series
vendor1_fares_series = pd.Series(vendor1_fares)
vendor2_fares_series = pd.Series(vendor2_fares)

# Check if the series are empty
if vendor1_fares_series.empty or vendor2_fares_series.empty:
    print("Error: One or both of the data lists are empty. Please provide valid data.")
else:
    # Combine fare data into a single DataFrame
    data = pd.DataFrame({
        'Fare': pd.concat([vendor1_fares_series, vendor2_fares_series], ignore_index=True),
        'Vendor': ['Vendor1'] * len(vendor1_fares) + ['Vendor2'] * len(vendor2_fares)
    })

    # Perform ANOVA
    model = ols('Fare ~ Vendor', data=data).fit()
    anova_table = anova_lm(model, typ=2)  # typ=2 gives you a traditional ANOVA table

    print(anova_table)
```

```
Vendor 1 Fares: 3         6.5
4          6.0
6         19.1
10        26.8
15        32.5
          ...
63823      7.2
63840      7.5
63843     12.8
63844     14.9
63846      9.3
Name: fare_amount, Length: 7549, dtype: float64
Vendor 2 Fares: 0        14.9
1         10.7
5         17.7
7         14.2
8          7.2
          ...
63882     15.0
63883     13.5
63884      9.3
63885     13.5
63886     11.4
Name: fare_amount, Length: 51609, dtype: float64
                sum_sq       df           F        PR(>F)
Vendor    1.029811e+04      1.0  117.934625  1.902207e-27
Residual  5.165530e+06  59156.0         NaN           NaN
```

Krushkal Wallis Test

In [185]:
```
stat, p = kruskal(vendor1_fares, vendor1_fares, vendor2_fares)

# Print the results
print(f"Kruskal-Wallis H Test: Statistics={stat:.3f}, p={p:.3f}")

# Interpretation
alpha = 0.05
if p < alpha:
    print('There is a statistically significant difference between the groups.')
else:
    print('There is no statistically significant difference between the groups.')
```

```
Kruskal-Wallis H Test: Statistics=271.784, p=0.000
There is a statistically significant difference between the groups.
```

In [67]:
```
from scipy.stats import mannwhitneyu

# Perform Mann-Whitney U test
stat, p = mannwhitneyu(vendor1_fares, vendor2_fares)

print('Mann-Whitney U Test: Statistics=%.3f, p=%.3f' % (stat, p))

# Interpretation
alpha = 0.05
if p < alpha:
    print('There is a statistically significant difference in fare amounts between Vendor 1 and Vendor 2.')
else:
    print('There is no statistically significant difference in fare amounts between Vendor 1 and Vendor 2.')
```

```
Mann-Whitney U Test: Statistics=177652869.500, p=0.000
There is a statistically significant difference in fare amounts between Vendor 1 and Vendor 2.
```

In [68]:
```
data_vendor1
```

Out[68]:

	VendorID	PULocationID	PUBorough	PUZone	PUservice_zone	DOLocationID	DOBorough	DOZone	DOservice_zone	passenge
3	1	41	Manhattan	Central Harlem	Boro Zone	238	Manhattan	Upper West Side North	Yellow Zone	
4	1	41	Manhattan	Central Harlem	Boro Zone	74	Manhattan	East Harlem North	Boro Zone	
6	1	181	Brooklyn	Park Slope	Boro Zone	45	Manhattan	Chinatown	Yellow Zone	
10	1	255	Brooklyn	Williamsburg (North Side)	Boro Zone	234	Manhattan	Union Sq	Yellow Zone	
15	1	195	Brooklyn	Red Hook	Boro Zone	210	Brooklyn	Sheepshead Bay	Boro Zone	
...	...	...	...	...	...	...	...	...	...	...
63823	1	129	Queens	Jackson Heights	Boro Zone	83	Queens	Elmhurst/Maspeth	Boro Zone	
63840	1	25	Brooklyn	Boerum Hill	Boro Zone	49	Brooklyn	Clinton Hill	Boro Zone	
63843	1	74	Manhattan	East Harlem North	Boro Zone	24	Manhattan	Bloomingdale	Yellow Zone	
63844	1	166	Manhattan	Morningside Heights	Boro Zone	244	Manhattan	Washington Heights South	Boro Zone	
63846	1	181	Brooklyn	Park Slope	Boro Zone	181	Brooklyn	Park Slope	Boro Zone	

7549 rows × 28 columns

In [69]:
```
data_vendor2
```

Out[69]:

	VendorID	PULocationID	PUBorough	PUZone	PUservice_zone	DOLocationID	DOBorough	DOZone	DOservice_zone	passeng
0	2	166	Manhattan	Morningside Heights	Boro Zone	143	Manhattan	Lincoln Square West	Yellow Zone	
1	2	24	Manhattan	Bloomingdale	Yellow Zone	43	Manhattan	Central Park	Yellow Zone	
5	2	41	Manhattan	Central Harlem	Boro Zone	262	Manhattan	Yorkville East	Yellow Zone	
7	2	24	Manhattan	Bloomingdale	Yellow Zone	75	Manhattan	East Harlem South	Boro Zone	
8	2	41	Manhattan	Central Harlem	Boro Zone	166	Manhattan	Morningside Heights	Boro Zone	
...	...	...	...	...	...	...	...	...	...	...
63882	2	130	Queens	Jamaica	Boro Zone	205	Queens	Saint Albans	Boro Zone	
63883	2	65	Brooklyn	Downtown Brooklyn/MetroTech	Boro Zone	181	Brooklyn	Park Slope	Boro Zone	
63884	2	244	Manhattan	Washington Heights South	Boro Zone	116	Manhattan	Hamilton Heights	Boro Zone	
63885	2	74	Manhattan	East Harlem North	Boro Zone	238	Manhattan	Upper West Side North	Yellow Zone	
63886	2	95	Queens	Forest Hills	Boro Zone	95	Queens	Forest Hills	Boro Zone	

51609 rows × 28 columns

In [70]:
```
data
```

Out[70]:

	VendorID	PULocationID	PUBorough	PUZone	PUservice_zone	DOLocationID	DOBorough	DOZone	DOservice_zone	passenger_
0	2	166	Manhattan	Morningside Heights	Boro Zone	143	Manhattan	Lincoln Square West	Yellow Zone	
1	2	24	Manhattan	Bloomingdale	Yellow Zone	43	Manhattan	Central Park	Yellow Zone	
3	1	41	Manhattan	Central Harlem	Boro Zone	238	Manhattan	Upper West Side North	Yellow Zone	
4	1	41	Manhattan	Central Harlem	Boro Zone	74	Manhattan	East Harlem North	Boro Zone	
5	2	41	Manhattan	Central Harlem	Boro Zone	262	Manhattan	Yorkville East	Yellow Zone	
...	...	...	...	...	...	...	...	...	...	...
63882	2	130	Queens	Jamaica	Boro Zone	205	Queens	Saint Albans	Boro Zone	
63883	2	65	Brooklyn	Downtown Brooklyn/MetroTech	Boro Zone	181	Brooklyn	Park Slope	Boro Zone	
63884	2	244	Manhattan	Washington Heights South	Boro Zone	116	Manhattan	Hamilton Heights	Boro Zone	
63885	2	74	Manhattan	East Harlem North	Boro Zone	238	Manhattan	Upper West Side North	Yellow Zone	
63886	2	95	Queens	Forest Hills	Boro Zone	95	Queens	Forest Hills	Boro Zone	

59158 rows × 29 columns

In [71]:
```
model_data = data
```

In [72]:
```
#model_data = model_data.drop(columns ='VendorID' )
```

In [73]:
```
model_data['Day_Of_Ride_7'] = model_data.Pickup_day % 7
```

In [74]:
```
drop = ['Day_Of_Ride_char','Pickup_day','Pickup_year','Dropoff_day','Dropoff_year','PUZone','PUservice_zone','DOZone',
```

In [75]:
```
model_data = model_data.drop(columns=drop)
```

In [76]:
```
model_data
```

Out[76]:

	VendorID	PULocationID	PUBorough	DOLocationID	DOBorough	passenger_count	trip_distance	fare_amount	extra	mta_tax	tip_amount
0	2	166	Manhattan	143	Manhattan	1.0	2.58	14.9	1.0	0.5	4.03
1	2	24	Manhattan	43	Manhattan	1.0	1.81	10.7	1.0	0.5	2.64
3	1	41	Manhattan	238	Manhattan	1.0	1.30	6.5	0.5	1.5	1.70
4	1	41	Manhattan	74	Manhattan	1.0	1.10	6.0	0.5	1.5	0.00
5	2	41	Manhattan	262	Manhattan	1.0	2.78	17.7	1.0	0.5	0.00
...	...	...	...	...	...	...	...	...	...	...	...
63882	2	130	Queens	205	Queens	1.0	2.75	15.0	0.0	0.0	2.00
63883	2	65	Brooklyn	181	Brooklyn	1.0	2.44	13.5	1.0	0.5	3.20
63884	2	244	Manhattan	116	Manhattan	1.0	1.40	9.3	1.0	0.5	2.36
63885	2	74	Manhattan	238	Manhattan	1.0	2.47	13.5	1.0	0.5	3.75
63886	2	95	Queens	95	Queens	1.0	1.39	11.4	1.0	0.5	4.17

59158 rows × 21 columns

In [77]:

```
# Convert columns to datetime format
model_data['Pickup_time'] = pd.to_datetime(model_data['Pickup_time'], format='%H:%M:%S').dt.time
model_data['Dropoff_time'] = pd.to_datetime(model_data['Dropoff_time'], format='%H:%M:%S').dt.time

# Convert time to timedelta and then calculate the difference
pickup_td = pd.to_timedelta(model_data['Pickup_time'].astype(str))
dropoff_td = pd.to_timedelta(model_data['Dropoff_time'].astype(str))
model_data['ride_duration_minutes'] = (dropoff_td - pickup_td).dt.total_seconds() / 60

# Handle negative values if the ride crosses midnight
model_data['ride_duration_minutes'] = model_data['ride_duration_minutes'].apply(lambda x: x + 24 * 60 if x < 0 else x)

# The 'ride_duration_minutes' column will have the duration in minutes
```

In [78]:

```
model_data.describe()
#to check the max value or duration
```

Out[78]:

	VendorID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount	extra	mta_tax	tip_amount	tolls_
count	59158.000000	59158.000000	59158.000000	59158.000000	59158.000000	59158.000000	59158.000000	59158.000000	59158.000000	59158
mean	1.872393	96.138764	138.026353	1.324098	2.596931	15.396912	0.921427	0.621091	2.076723	0
std	0.333655	58.483043	76.371527	0.992160	2.204801	9.353774	1.305405	0.338462	2.659227	0
min	1.000000	3.000000	1.000000	0.000000	0.270000	0.010000	0.000000	0.000000	0.000000	0
25%	2.000000	74.000000	74.000000	1.000000	1.220000	9.300000	0.000000	0.500000	0.000000	0
50%	2.000000	75.000000	138.000000	1.000000	1.890000	12.800000	0.000000	0.500000	1.740000	0
75%	2.000000	97.000000	220.000000	1.000000	3.130000	18.400000	2.500000	0.500000	3.280000	0
max	2.000000	263.000000	263.000000	7.000000	28.670000	69.500000	7.500000	1.500000	80.880000	14

In [79]:
```
model_data
```

Out[79]:

	VendorID	PULocationID	PUBorough	DOLocationID	DOBorough	passenger_count	trip_distance	fare_amount	extra	mta_tax	tip_amount
0	2	166	Manhattan	143	Manhattan	1.0	2.58	14.9	1.0	0.5	4.03
1	2	24	Manhattan	43	Manhattan	1.0	1.81	10.7	1.0	0.5	2.64
3	1	41	Manhattan	238	Manhattan	1.0	1.30	6.5	0.5	1.5	1.70
4	1	41	Manhattan	74	Manhattan	1.0	1.10	6.0	0.5	1.5	0.00
5	2	41	Manhattan	262	Manhattan	1.0	2.78	17.7	1.0	0.5	0.00
...	...	...	...	...	...	...	...	...	...	...	...
63882	2	130	Queens	205	Queens	1.0	2.75	15.0	0.0	0.0	2.00
63883	2	65	Brooklyn	181	Brooklyn	1.0	2.44	13.5	1.0	0.5	3.20
63884	2	244	Manhattan	116	Manhattan	1.0	1.40	9.3	1.0	0.5	2.36
63885	2	74	Manhattan	238	Manhattan	1.0	2.47	13.5	1.0	0.5	3.75
63886	2	95	Queens	95	Queens	1.0	1.39	11.4	1.0	0.5	4.17

59158 rows × 22 columns

In [80]:
```
#MSE is low so ill try to remove the outliers for ride duration
```

In [81]:
```
model_data1 = model_data
```

In [82]:
```
model_data = model_data[model_data['ride_duration_minutes'] < 720]
```

In [83]:
```
model_data.describe()
```

Out[83]:

	VendorID	PULocationID	DOLocationID	passenger_count	trip_distance	fare_amount	extra	mta_tax	tip_amount	tolls_
count	58978.000000	58978.000000	58978.000000	58978.000000	58978.000000	58978.000000	58978.000000	58978.000000	58978.000000	58978
mean	1.872003	96.138933	138.006850	1.324579	2.594944	15.386143	0.922111	0.621494	2.078004	0
std	0.334089	58.493600	76.374797	0.992978	2.202617	9.341595	1.306022	0.338875	2.658856	0
min	1.000000	3.000000	1.000000	0.000000	0.270000	0.010000	0.000000	0.000000	0.000000	0
25%	2.000000	74.000000	74.000000	1.000000	1.220000	9.300000	0.000000	0.500000	0.000000	0
50%	2.000000	75.000000	138.000000	1.000000	1.890000	12.800000	0.000000	0.500000	1.740000	0
75%	2.000000	97.000000	220.000000	1.000000	3.130000	18.400000	2.500000	0.500000	3.280000	0
max	2.000000	263.000000	263.000000	7.000000	28.670000	69.500000	7.500000	1.500000	80.880000	14

In [84]:
```
drop1 = ['Dropoff_time','Pickup_time','fare_amount']
```

In [85]:
```
model_data = model_data.drop(columns=drop1)
```

In [86]:
```
model_data
```

Out[86]:

	VendorID	PULocationID	PUBorough	DOLocationID	DOBorough	passenger_count	trip_distance	extra	mta_tax	tip_amount	tolls_amount
0	2	166	Manhattan	143	Manhattan	1.0	2.58	1.0	0.5	4.03	0.0
1	2	24	Manhattan	43	Manhattan	1.0	1.81	1.0	0.5	2.64	0.0
3	1	41	Manhattan	238	Manhattan	1.0	1.30	0.5	1.5	1.70	0.0
4	1	41	Manhattan	74	Manhattan	1.0	1.10	0.5	1.5	0.00	0.0
5	2	41	Manhattan	262	Manhattan	1.0	2.78	1.0	0.5	0.00	0.0
...	...	...	...	...	...	...	...	...	...	...	...
63882	2	130	Queens	205	Queens	1.0	2.75	0.0	0.0	2.00	0.0
63883	2	65	Brooklyn	181	Brooklyn	1.0	2.44	1.0	0.5	3.20	0.0
63884	2	244	Manhattan	116	Manhattan	1.0	1.40	1.0	0.5	2.36	0.0
63885	2	74	Manhattan	238	Manhattan	1.0	2.47	1.0	0.5	3.75	0.0
63886	2	95	Queens	95	Queens	1.0	1.39	1.0	0.5	4.17	0.0

58978 rows × 19 columns

In [87]:
```
y = model_data['total_amount']
y
```

Out[87]:
```
0        24.18
1        15.84
3        10.20
4         8.00
5        22.95
         ...
63882    17.30
63883    19.20
63884    14.16
63885    22.50
63886    18.07
Name: total_amount, Length: 58978, dtype: float64
```

In [90]:
```
# Filter rows where 'VendorID' is 1 and select the 'total_amount' column
y_vendor1 = model_data[model_data['VendorID'] == 1]['total_amount']

print(y_vendor1)
```

```
3        10.20
4         8.00
6        29.20
10       32.05
15       34.50
         ...
63823     9.70
63840    11.85
63843    18.35
63844    20.85
63846    11.80
Name: total_amount, Length: 7549, dtype: float64
```

In [92]:
```
# Filter rows where 'VendorID' is 1 and select the 'total_amount' column
y_vendor2 = model_data[model_data['VendorID'] == 2]['total_amount']

print(y_vendor2)
```

```
0        24.18
1        15.84
5        22.95
7        16.70
8        10.70
         ...
63882    17.30
63883    19.20
63884    14.16
63885    22.50
63886    18.07
Name: total_amount, Length: 51429, dtype: float64
```

In [93]:
```
model_data = model_data.drop(columns ='total_amount' )
```

In [94]:
```
columns_to_encode = ['PUBorough','DOBorough']
```

In [95]:
```
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

# Assuming `model_data` is your dataframe
X = model_data.copy()


# Initialize the OneHotEncoder
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

# Apply the encoder
encoded_array = encoder.fit_transform(X[columns_to_encode])

# Convert the encoded array to a DataFrame with proper column names
encoded_df = pd.DataFrame(encoded_array, columns=encoder.get_feature_names_out(columns_to_encode), index=X.index)

# Drop the original columns that were encoded
X.drop(columns_to_encode, axis=1, inplace=True)

# Concatenate the original DataFrame with the encoded columns
X_encoded = pd.concat([X, encoded_df], axis=1)

# Now `X_encoded` contains both the original and the one-hot encoded columns
print(X_encoded)
```

```
       VendorID  PULocationID  DOLocationID  passenger_count  trip_distance  \
0             2           166           143              1.0           2.58
1             2            24            43              1.0           1.81
3             1            41           238              1.0           1.30
4             1            41            74              1.0           1.10
5             2            41           262              1.0           2.78
...         ...           ...           ...              ...            ...
63882         2           130           205              1.0           2.75
63883         2            65           181              1.0           2.44
63884         2           244           116              1.0           1.40
63885         2            74           238              1.0           2.47
63886         2            95            95              1.0           1.39

       extra  mta_tax  tip_amount  tolls_amount  improvement_surcharge  \
0        1.0      0.5        4.03           0.0                    1.0
1        1.0      0.5        2.64           0.0                    1.0
3        0.5      1.5        1.70           0.0                    1.0
4        0.5      1.5        0.00           0.0                    1.0
5        1.0      0.5        0.00           0.0                    1.0
...      ...      ...         ...           ...                    ...
63882    0.0      0.0        2.00           0.0                    0.3
63883    1.0      0.5        3.20           0.0                    1.0
63884    1.0      0.5        2.36           0.0                    1.0
63885    1.0      0.5        3.75           0.0                    1.0
63886    1.0      0.5        4.17           0.0                    1.0

       payment_type  trip_type  congestion_surcharge  Pickup_hour  \
0               1.0        1.0                  2.75            0
1               1.0        1.0                  0.00            0
3               1.0        1.0                  0.00            0
4               1.0        1.0                  0.00            0
5               2.0        1.0                  2.75            0
...             ...        ...                   ...          ...
63882           1.0        2.0                  0.00           23
63883           1.0        1.0                  0.00           23
63884           1.0        1.0                  0.00           23
63885           1.0        1.0                  2.75           23
63886           1.0        1.0                  0.00           23

       Day_Of_Ride_7  ride_duration_minutes  PUBorough_Bronx  \
0                  1                   11.0              0.0
```

```
1                 1                  6.0             0.0
3                 1                  6.0             0.0
4                 1                  6.0             0.0
5                 1                 18.0             0.0
...             ...                  ...             ...
63882             3                  8.0             0.0
63883             3                 11.0             0.0
63884             3                  6.0             0.0
63885             3                  9.0             0.0
63886             3                 10.0             0.0

       PUBorough_Brooklyn  PUBorough_Manhattan  PUBorough_Queens  \
0                     0.0                  1.0               0.0
1                     0.0                  1.0               0.0
3                     0.0                  1.0               0.0
4                     0.0                  1.0               0.0
5                     0.0                  1.0               0.0
...                   ...                  ...               ...
63882                 0.0                  0.0               1.0
63883                 1.0                  0.0               0.0
63884                 0.0                  1.0               0.0
63885                 0.0                  1.0               0.0
63886                 0.0                  0.0               1.0

       PUBorough_Staten Island  DOBorough_Bronx  DOBorough_Brooklyn  \
0                          0.0              0.0                 0.0
1                          0.0              0.0                 0.0
3                          0.0              0.0                 0.0
4                          0.0              0.0                 0.0
5                          0.0              0.0                 0.0
...                        ...              ...                 ...
63882                      0.0              0.0                 0.0
63883                      0.0              0.0                 1.0
63884                      0.0              0.0                 0.0
63885                      0.0              0.0                 0.0
63886                      0.0              0.0                 0.0

       DOBorough_EWR  DOBorough_Manhattan  DOBorough_Queens  \
0                0.0                  1.0               0.0
1                0.0                  1.0               0.0
3                0.0                  1.0               0.0
4                0.0                  1.0               0.0
```

```
5                0.0                1.0                0.0
...              ...                ...                ...
63882            0.0                0.0                1.0
63883            0.0                0.0                0.0
63884            0.0                1.0                0.0
63885            0.0                1.0                0.0
63886            0.0                0.0                1.0

       DOBorough_Staten Island
0                           0.0
1                           0.0
3                           0.0
4                           0.0
5                           0.0
...                         ...
63882                       0.0
63883                       0.0
63884                       0.0
63885                       0.0
63886                       0.0

[58978 rows x 27 columns]
```

In [96]:
```
'''from sklearn.preprocessing import OneHotEncoder
import pandas as pd

X = model_data

# Initialize the OneHotEncoder
encoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')

# Apply the encoder
X_encoded = pd.DataFrame(encoder.fit_transform(X[columns_to_encode]))

# Assign the encoded column names
X_encoded.columns = encoder.get_feature_names_out(columns_to_encode)

# Concatenate the encoded columns back to the original dataframe
X = X.drop(columns_to_encode, axis=1)
X = pd.concat([X, X_encoded], axis=1)

'''
```

Out[96]:
```
"from sklearn.preprocessing import OneHotEncoder\nimport pandas as pd\n\nX = model_data\n\n# Initialize the OneHotEnc
oder\nencoder = OneHotEncoder(sparse_output=False, handle_unknown='ignore')\n\n# Apply the encoder\nX_encoded = pd.Da
taFrame(encoder.fit_transform(X[columns_to_encode]))\n\n# Assign the encoded column names\nX_encoded.columns = encode
r.get_feature_names_out(columns_to_encode)\n\n# Concatenate the encoded columns back to the original dataframe\nX =
X.drop(columns_to_encode, axis=1)\nX = pd.concat([X, X_encoded], axis=1)\n\n"
```

In [97]:
```
X_encoded.describe()
```

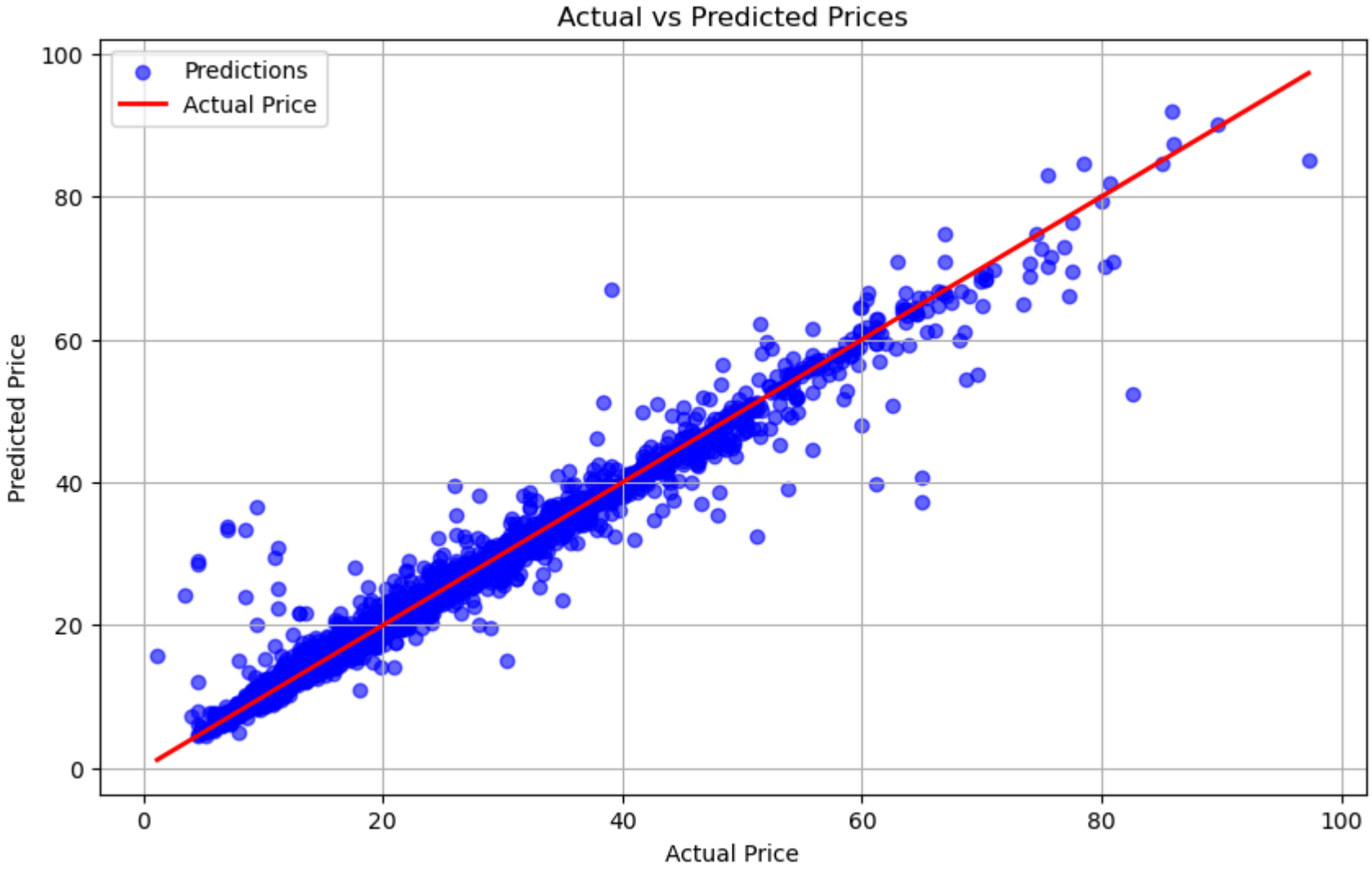
Out[97]:

	VendorID	PULocationID	DOLocationID	passenger_count	trip_distance	extra	mta_tax	tip_amount	tolls_amount	impro
count	58978.000000	58978.000000	58978.000000	58978.000000	58978.000000	58978.000000	58978.000000	58978.000000	58978.000000	
mean	1.872003	96.138933	138.006850	1.324579	2.594944	0.922111	0.621494	2.078004	0.105603	
std	0.334089	58.493600	76.374797	0.992978	2.202617	1.306022	0.338875	2.658856	0.820672	
min	1.000000	3.000000	1.000000	0.000000	0.270000	0.000000	0.000000	0.000000	0.000000	
25%	2.000000	74.000000	74.000000	1.000000	1.220000	0.000000	0.500000	0.000000	0.000000	
50%	2.000000	75.000000	138.000000	1.000000	1.890000	0.000000	0.500000	1.740000	0.000000	
75%	2.000000	97.000000	220.000000	1.000000	3.130000	2.500000	0.500000	3.280000	0.000000	
max	2.000000	263.000000	263.000000	7.000000	28.670000	7.500000	1.500000	80.880000	14.750000	

In [155]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from category_encoders import TargetEncoder

from sklearn.metrics import mean_squared_error



# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.1, random_state=42)

# Initialize and train a random forest regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Compute Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Compute Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# Compute R2 Score
r2 = r2_score(y_test, y_pred)

# Print the results
print("The prediction metric using Random Forest are :")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R2 Score: {r2}")
```

```
The prediction metric using Random Forest are :
Mean Squared Error (MSE): 3.5895829185686314
Root Mean Squared Error (RMSE): 1.894619465372567
R2 Score: 0.9738761847060177
```

In [154]:
```
import matplotlib.pyplot as plt

# Plotting the actual vs predicted values
plt.figure(figsize=(10, 6))

# Scatter plot of predicted vs actual
plt.scatter(y_test, y_pred, color='blue', alpha=0.6, label='Predictions')

# Line representing perfect predictions
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', lw=2, label='Actual Price')

# Adding labels and title
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted Prices')
plt.legend()
plt.grid(True)
plt.show()
```


Actual vs Predicted Prices
Predictions
Actual Price
Predicted Price
Actual Price
0
20
40
60
80
100

In [99]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Train the model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict on the training dataset
y_train_pred = model.predict(X_train)
train_mse = mean_squared_error(y_train, y_train_pred)
train_rmse = np.sqrt(train_mse)
train_r2 = r2_score(y_train, y_train_pred)

# Predict on the test dataset
y_test_pred = model.predict(X_test)
test_mse = mean_squared_error(y_test, y_test_pred)
test_rmse = np.sqrt(test_mse)
test_r2 = r2_score(y_test, y_test_pred)

# Output the results
print("Training Data Evaluation:")
print(f"Mean Squared Error (MSE): {train_mse}")
print(f"Root Mean Squared Error (RMSE): {train_rmse}")
print(f"R2 Score: {train_r2}\n")

print("Test Data Evaluation:")
print(f"Mean Squared Error (MSE): {test_mse}")
print(f"Root Mean Squared Error (RMSE): {test_rmse}")
print(f"R2 Score: {test_r2}")

# Check for overfitting
if test_rmse > train_rmse and train_r2 > test_r2:
    print("The model might be overfitting.")
else:
    print("The model doesn't seem to be overfitting.")
```

```
Training Data Evaluation:
Mean Squared Error (MSE): 8.923538689776585
Root Mean Squared Error (RMSE): 2.987229266356465
R2 Score: 0.9313873378804818

Test Data Evaluation:
Mean Squared Error (MSE): 8.450015126555792
Root Mean Squared Error (RMSE): 2.9068909725952556
R2 Score: 0.9385035422205751
The model doesn't seem to be overfitting.
```

In [100]:
```python
from sklearn.ensemble import GradientBoostingRegressor

model = GradientBoostingRegressor(random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R2 Score: {r2}")
```

```
Mean Squared Error (MSE): 3.581416714236963
Root Mean Squared Error (RMSE): 1.8924631341817368
R2 Score: 0.9739356156812738
```

In [101]:
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Initialize the XGBoost regressor
xgb_model = xgb.XGBRegressor(random_state=42)

# Train the model
xgb_model.fit(X_train, y_train)

# Make predictions
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate the model
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
rmse_xgb = np.sqrt(mse_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

# Print the results
print("XGBoost Results:")
print(f"Mean Squared Error (MSE): {mse_xgb}")
print(f"Root Mean Squared Error (RMSE): {rmse_xgb}")
print(f"R2 Score: {r2_xgb}")
```

```
XGBoost Results:
Mean Squared Error (MSE): 3.469530383066424
Root Mean Squared Error (RMSE): 1.86266754496513
R2 Score: 0.9747498879562784
```

In [102]:
```
data_vendor1 = model_data[data['VendorID'] == 1]
```

```
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\2571694340.py:1: UserWarning: Boolean Series key will be reindexed t
o match DataFrame index.
  data_vendor1 = model_data[data['VendorID'] == 1]
```

In [141]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from category_encoders import TargetEncoder
from sklearn.metrics import mean_squared_error



# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.1, random_state=42)

# Initialize and train a random forest regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train, y_train)

# Make predictions
y_pred = rf_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Compute Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred)

# Compute Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# Compute R2 Score
r2 = r2_score(y_test, y_pred)

# Print the results

print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R2 Score: {r2}")
```

```
Mean Squared Error (MSE): 3.5895829185686314
Root Mean Squared Error (RMSE): 1.894619465372567
R2 Score: 0.9738761847060177
```

In [115]:
```
data_vendor1 = X_encoded[data['VendorID'] == 1]
```

```
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\1374024510.py:1: UserWarning: Boolean Series key will be reindexed t
o match DataFrame index.
  data_vendor1 = X_encoded[data['VendorID'] == 1]
```

In [116]:
```
data_vendor2 = X_encoded[data['VendorID'] == 2]
```

```
C:\Users\AMIT\AppData\Local\Temp\ipykernel_21576\4143764919.py:1: UserWarning: Boolean Series key will be reindexed t
o match DataFrame index.
  data_vendor2 = X_encoded[data['VendorID'] == 2]
```

In [121]:
```
data_vendor2
```

Out[121]:

	VendorID	PULocationID	DOLocationID	passenger_count	trip_distance	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge
0	2	166	143	1.0	2.58	1.0	0.5	4.03	0.0	1.0
1	2	24	43	1.0	1.81	1.0	0.5	2.64	0.0	1.0
5	2	41	262	1.0	2.78	1.0	0.5	0.00	0.0	1.0
7	2	24	75	1.0	1.88	1.0	0.5	0.00	0.0	1.0
8	2	41	166	2.0	1.11	1.0	0.5	1.00	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...
63882	2	130	205	1.0	2.75	0.0	0.0	2.00	0.0	0.3
63883	2	65	181	1.0	2.44	1.0	0.5	3.20	0.0	1.0
63884	2	244	116	1.0	1.40	1.0	0.5	2.36	0.0	1.0
63885	2	74	238	1.0	2.47	1.0	0.5	3.75	0.0	1.0
63886	2	95	95	1.0	1.39	1.0	0.5	4.17	0.0	1.0

51429 rows × 27 columns

FOR VENDOR 2

In [156]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from category_encoders import TargetEncoder
from sklearn.metrics import mean_squared_error



# Split the dataset into training and testing sets
X_train_vendor2, X_test_vendor2, y_train_vendor2, y_test_vendor2 = train_test_split(data_vendor2, y_vendor2, test_size

# Initialize and train a random forest regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train_vendor2, y_train_vendor2)

# Make predictions
y_pred_vendor2 = rf_model.predict(X_test)

#TESTING ON MAIN DATASET
# Evaluate the model
mse = mean_squared_error(y_test, y_pred_vendor2)
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Compute Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred_vendor2)

# Compute Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# Compute R2 Score
r2 = r2_score(y_test, y_pred_vendor2)

# Print the results
print("The prediction metric for vendor 2 using Random Forest are :")
print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R2 Score: {r2}")
```

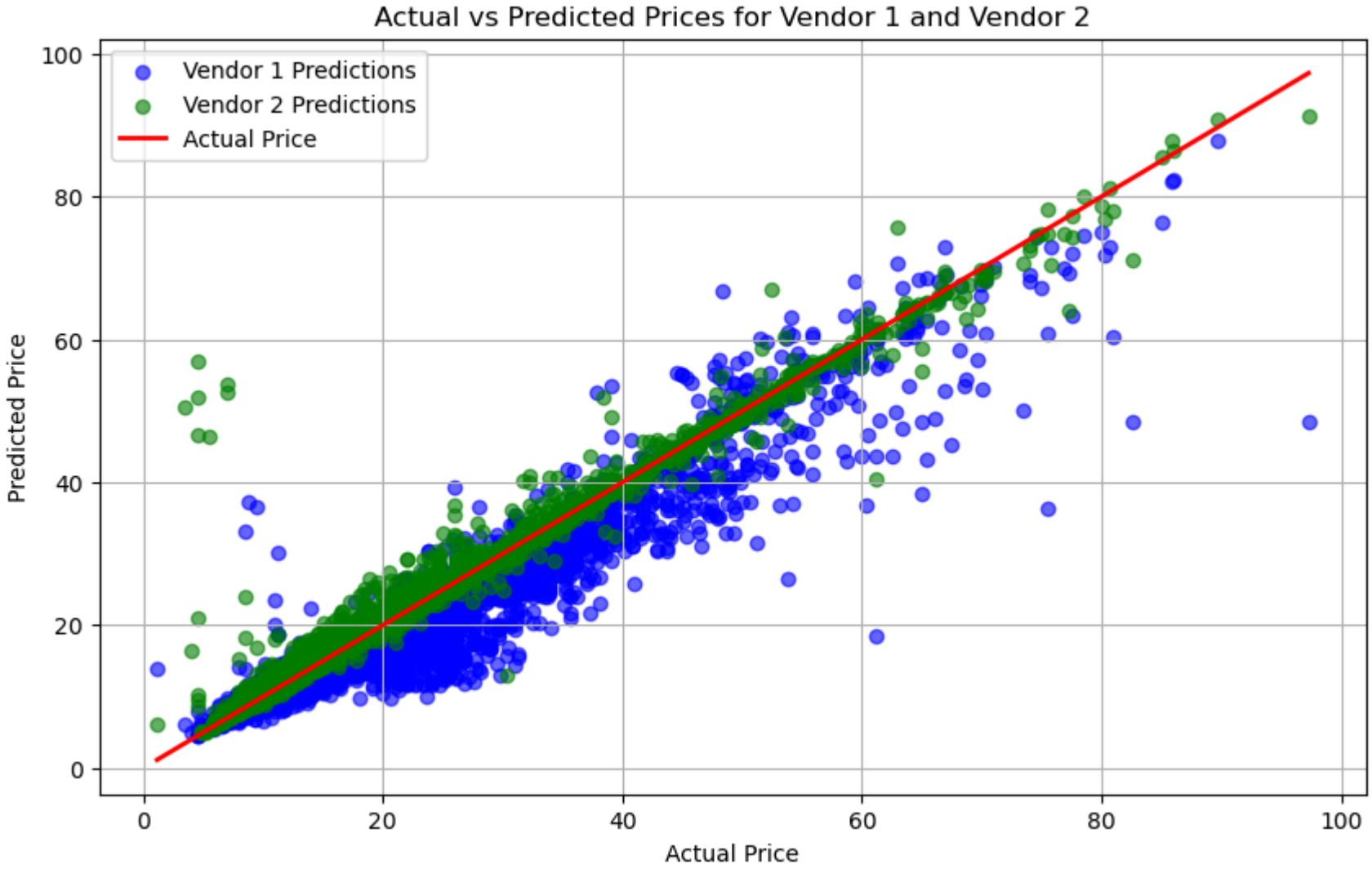
```
The prediction metric for vendor 2 using Random Forest are :
Mean Squared Error (MSE): 3.989536501867395
Root Mean Squared Error (RMSE): 1.9973824125258024
R2 Score: 0.9709654527983592
```

FOR VENDOR 1

In [157]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from category_encoders import TargetEncoder
from sklearn.metrics import mean_squared_error



# Split the dataset into training and testing sets
X_train_vendor1, X_test_vendor1, y_train_vendor1, y_test_vendor1 = train_test_split(data_vendor1, y_vendor1, test_size

# Initialize and train a random forest regressor
rf_model = RandomForestRegressor(random_state=42)
rf_model.fit(X_train_vendor1, y_train_vendor1)

# Make predictions
y_pred_vendor1 = rf_model.predict(X_test)

#TESTING ON MAIN DATASET
# Evaluate the model
mse = mean_squared_error(y_test, y_pred_vendor1)
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Compute Mean Squared Error (MSE)
mse = mean_squared_error(y_test, y_pred_vendor1)

# Compute Root Mean Squared Error (RMSE)
rmse = np.sqrt(mse)

# Compute R2 Score
r2 = r2_score(y_test, y_pred_vendor1)

# Print the results
print("The prediction metric for vendor 1 using Random Forest are :")

print(f"Mean Squared Error (MSE): {mse}")
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R2 Score: {r2}")
```

```
The prediction metric for vendor 1 using Random Forest are :
Mean Squared Error (MSE): 14.280498953030941
Root Mean Squared Error (RMSE): 3.7789547434483706
R2 Score: 0.8960711800178586
```

In [153]:
```
import matplotlib.pyplot as plt


plt.figure(figsize=(10, 6))

# Scatter plot for predictions from Vendor 1
plt.scatter(y_test, y_pred_vendor1, color='blue', alpha=0.6, label='Vendor 1 Predictions')

# Scatter plot for predictions from Vendor 2
plt.scatter(y_test, y_pred_vendor2, color='green', alpha=0.6, label='Vendor 2 Predictions')

# Line representing perfect predictions
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', lw=2, label='Actual Price')

# Adding labels and title
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted Prices for Vendor 1 and Vendor 2')
plt.legend()
plt.grid(True)
plt.show()
```


Actual vs Predicted Prices for Vendor 1 and Vendor 2
Vendor 1 Predictions
Vendor 2 Predictions
Actual Price
Predicted Price
Actual Price
0
20
40
60
80
100

In [171]:
```
model_data
```

Out[171]:

	VendorID	PULocationID	PUBorough	DOLocationID	DOBorough	passenger_count	trip_distance	extra	mta_tax	tip_amount	tolls_amount
0	2	166	Manhattan	143	Manhattan	1.0	2.58	1.0	0.5	4.03	0.0
1	2	24	Manhattan	43	Manhattan	1.0	1.81	1.0	0.5	2.64	0.0
3	1	41	Manhattan	238	Manhattan	1.0	1.30	0.5	1.5	1.70	0.0
4	1	41	Manhattan	74	Manhattan	1.0	1.10	0.5	1.5	0.00	0.0
5	2	41	Manhattan	262	Manhattan	1.0	2.78	1.0	0.5	0.00	0.0
...	...	...	...	...	...	...	...	...	...	...	...
63882	2	130	Queens	205	Queens	1.0	2.75	0.0	0.0	2.00	0.0
63883	2	65	Brooklyn	181	Brooklyn	1.0	2.44	1.0	0.5	3.20	0.0
63884	2	244	Manhattan	116	Manhattan	1.0	1.40	1.0	0.5	2.36	0.0
63885	2	74	Manhattan	238	Manhattan	1.0	2.47	1.0	0.5	3.75	0.0
63886	2	95	Queens	95	Queens	1.0	1.39	1.0	0.5	4.17	0.0

58978 rows × 18 columns

In [175]:
```
X_encoded.to_csv('final_data_aqaaaaaa.csv', index=False)
```

In [173]:
```
X_encoded['total_amount'] = y
```

In [174]:
```
X_encoded
```

Out[174]:

	VendorID	PULocationID	DOLocationID	passenger_count	trip_distance	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge
0	2	166	143	1.0	2.58	1.0	0.5	4.03	0.0	1.0
1	2	24	43	1.0	1.81	1.0	0.5	2.64	0.0	1.0
3	1	41	238	1.0	1.30	0.5	1.5	1.70	0.0	1.0
4	1	41	74	1.0	1.10	0.5	1.5	0.00	0.0	1.0
5	2	41	262	1.0	2.78	1.0	0.5	0.00	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...
63882	2	130	205	1.0	2.75	0.0	0.0	2.00	0.0	0.3
63883	2	65	181	1.0	2.44	1.0	0.5	3.20	0.0	1.0
63884	2	244	116	1.0	1.40	1.0	0.5	2.36	0.0	1.0
63885	2	74	238	1.0	2.47	1.0	0.5	3.75	0.0	1.0
63886	2	95	95	1.0	1.39	1.0	0.5	4.17	0.0	1.0

58978 rows × 28 columns

In [176]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

In [179]:
```
model = LinearRegression()
model.fit(X_encoded, y)
y_pred = model.predict(X_encoded)
```

In [180]:
```
r2 = r2_score(y, y_pred)
print(f"R-squared: {r2}")
```

```
R-squared: 1.0
```

In [181]:
```
intercept = model.intercept_
slope = model.coef_[0]
print(f"Intercept (MLE): {intercept}, Slope (MLE): {slope}")
```

```
Intercept (MLE): 1.4210854715202004e-13, Slope (MLE): 1.4107261673121111e-14
```

In [182]:
```
rmse = np.sqrt(mean_squared_error(y, y_pred))
print(f"RMSE: {rmse}")
```

```
RMSE: 1.3774934564787173e-13
```

In [183]:
```
mse = mean_squared_error(y, y_pred)
print(f"MSE: {mse}")
```

```
MSE: 1.8974882226416837e-26
```

In [ ]:
```

```