1. Lets begin by imporint Libraries necessary for our analysis viz., Numpy, Pandas, Matplotlib and Seaborn.

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

1. Let us now verify the versions ensuring the latest versions have been imported for our use.

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

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

1. We will now mount the drive that contains the relevant data from Google Drive

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

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

1. We will now unmount the drive to free the RAM. Then reconnect and load the file from the local system. Then we will read the sampled_data.csv file from the RAM. We will also check the nature of the columns to get a better undertsanding of the data.

```
sampled_data=pd.read_csv('final_trip_sample_5percent.csv')
sampled_data.info()
print(f"Number of rows: {len(sampled_data)}")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1550457 entries, 0 to 1550456
Data columns (total 22 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   VendorID               1550457 non-null  int64
 1   tpep_pickup_datetime   1550457 non-null  object
 2   tpep_dropoff_datetime  1550457 non-null  object
 3   passenger_count        1501863 non-null  float64
 4   trip_distance          1550457 non-null  float64
 5   RatecodeID             1501863 non-null  float64
 6   store_and_fwd_flag     1501863 non-null  object
 7   PULocationID           1550457 non-null  int64
 8   DOLocationID           1550457 non-null  int64
```

```
 9   payment_type             1550457 non-null   int64
 10  fare_amount              1550456 non-null   float64
 11  extra                    1550456 non-null   float64
 12  mta_tax                  1550456 non-null   float64
 13  tip_amount               1550456 non-null   float64
 14  tolls_amount             1550456 non-null   float64
 15  improvement_surcharge    1550456 non-null   float64
 16  total_amount             1550456 non-null   float64
 17  congestion_surcharge     1501862 non-null   float64
 18  airport_fee              148483 non-null    float64
 19  pickup_date              1550456 non-null   object
 20  pickup_hour              1550456 non-null   float64
 21  Airport_fee              1353379 non-null   float64
dtypes: float64(14), int64(4), object(4)
memory usage: 260.2+ MB
Number of rows: 1550457
```

We will now take a look at the data to visualize the columns

```
sampled_data.head()
```

{"type":"dataframe","variable_name":"sampled_data"}

Lets fix the index first

```
sampled_data.reset_index(drop=True, inplace=True)
sampled_data.index += 1

sampled_data.head()
```

{"type":"dataframe","variable_name":"sampled_data"}

Lets try and understand what data each column holds by grouping all the columns based on the data that they contain.

```
pd.set_option('display.float_format', lambda x: '%.2f' % x)

for col in sampled_data.columns:
    print(f"\n===== Summary for Column: {col} =====\n")

    if pd.api.types.is_numeric_dtype(sampled_data[col]):
        print("Descriptive Stats:\n", sampled_data[col].describe())
        print("\nTop 5 Unique Values:\n",
sampled_data[col].value_counts().head())

    elif pd.api.types.is_datetime64_any_dtype(sampled_data[col]):
        print("Descriptive Stats:\n", sampled_data[col].describe())
        print("\nTrips per Date (Top 5):\n",
```

```
sampled_data[col].dt.date.value_counts().head())

    else:
        print("Top 5 Frequent Values:\n",
sampled_data[col].value_counts().head())
```

```
===== Summary for Column: VendorID =====

Descriptive Stats:
 count   1550457.00
mean          1.73
std           0.45
min           1.00
25%           1.00
50%           2.00
75%           2.00
max           6.00
Name: VendorID, dtype: float64

Top 5 Unique Values:
 VendorID
2    1131454
1     418603
6        400
Name: count, dtype: int64

===== Summary for Column: tpep_pickup_datetime =====

Top 5 Frequent Values:
 tpep_pickup_datetime
2023-09-15 11:56:59    4
2023-08-21 21:21:39    4
2023-09-16 23:52:36    4
2023-04-18 11:32:20    4
2023-02-16 08:22:29    4
Name: count, dtype: int64

===== Summary for Column: tpep_dropoff_datetime =====

Top 5 Frequent Values:
 tpep_dropoff_datetime
2023-09-04 00:00:00    5
2023-06-09 17:12:35    4
2023-01-19 18:53:36    4
2023-04-29 20:05:46    4
2023-02-09 22:53:29    4
Name: count, dtype: int64

===== Summary for Column: passenger_count =====
```

```
Descriptive Stats:
 count   1501863.00
mean          1.37
std           0.89
min           0.00
25%           1.00
50%           1.00
75%           1.00
max           9.00
Name: passenger_count, dtype: float64

Top 5 Unique Values:
 passenger_count
1.00    1130717
2.00     226227
3.00      56083
4.00      30531
0.00      25257
Name: count, dtype: int64

===== Summary for Column: trip_distance =====

Descriptive Stats:
 count   1550457.00
mean          3.90
std         135.02
min           0.00
25%           1.06
50%           1.80
75%           3.44
max      126360.46
Name: trip_distance, dtype: float64

Top 5 Unique Values:
 trip_distance
0.00    28747
1.00    21210
0.90    21160
1.10    20878
0.80    20675
Name: count, dtype: int64

===== Summary for Column: RatecodeID =====

Descriptive Stats:
 count   1501863.00
mean          1.61
std           7.23
min           1.00
```

```
25%            1.00
50%            1.00
75%            1.00
max           99.00
Name: RatecodeID, dtype: float64

Top 5 Unique Values:
 RatecodeID
1.00     1417830
2.00       59631
99.00       8208
5.00        8133
3.00        5012
Name: count, dtype: int64

===== Summary for Column: store_and_fwd_flag =====

Top 5 Frequent Values:
 store_and_fwd_flag
N    1492108
Y       9755
Name: count, dtype: int64

===== Summary for Column: PULocationID =====

Descriptive Stats:
 count   1550457.00
mean        165.14
std          64.01
min           1.00
25%         132.00
50%         162.00
75%         234.00
max         265.00
Name: PULocationID, dtype: float64

Top 5 Unique Values:
 PULocationID
132    81404
237    71323
161    71320
236    64173
162    54252
Name: count, dtype: int64

===== Summary for Column: DOLocationID =====

Descriptive Stats:
 count   1550457.00
mean        163.89
```

```
std           69.87
min            1.00
25%          113.00
50%          162.00
75%          234.00
max          265.00
Name: DOLocationID, dtype: float64

Top 5 Unique Values:
 DOLocationID
236    67194
237    63994
161    60078
230    47457
170    45780
Name: count, dtype: int64

===== Summary for Column: payment_type =====

Descriptive Stats:
 count   1550457.00
mean          1.17
std           0.50
min           0.00
25%           1.00
50%           1.00
75%           1.00
max           4.00
Name: payment_type, dtype: float64

Top 5 Unique Values:
 payment_type
1    1221291
2     262321
0      48594
4      10842
3       7409
Name: count, dtype: int64

===== Summary for Column: fare_amount =====

Descriptive Stats:
 count   1550456.00
mean         19.89
std         116.41
min           0.00
25%           9.30
50%          13.50
75%          21.90
max      143163.45
```

```
Name: fare_amount, dtype: float64

Top 5 Unique Values:
 fare_amount
8.60     69329
9.30     68557
10.00    68449
7.90     66896
10.70    64978
Name: count, dtype: int64

===== Summary for Column: extra =====

Descriptive Stats:
 count   1550456.00
mean          1.60
std           1.83
min          -2.50
25%           0.00
50%           1.00
75%           2.50
max          20.80
Name: extra, dtype: float64

Top 5 Unique Values:
 extra
0.00    611227
2.50    386156
1.00    297266
5.00    110647
3.50     88615
Name: count, dtype: int64

===== Summary for Column: mta_tax =====

Descriptive Stats:
 count   1550456.00
mean          0.50
std           0.05
min          -0.50
25%           0.50
50%           0.50
75%           0.50
max           4.00
Name: mta_tax, dtype: float64

Top 5 Unique Values:
 mta_tax
0.50     1536071
0.00       14256
```

```
-0.50         56
0.80          52
0.05          17
Name: count, dtype: int64

===== Summary for Column: tip_amount =====

Descriptive Stats:
 count   1550456.00
mean          3.54
std           4.05
min           0.00
25%           1.00
50%           2.82
75%           4.41
max         223.08
Name: tip_amount, dtype: float64

Top 5 Unique Values:
 tip_amount
0.00    355070
2.00     77842
1.00     62191
3.00     41220
5.00     23389
Name: count, dtype: int64

===== Summary for Column: tolls_amount =====

Descriptive Stats:
 count   1550456.00
mean          0.60
std           2.18
min           0.00
25%           0.00
50%           0.00
75%           0.00
max         143.00
Name: tolls_amount, dtype: float64

Top 5 Unique Values:
 tolls_amount
0.00     1423900
6.55       84269
6.94       31720
12.75       1644
14.75       1352
Name: count, dtype: int64

===== Summary for Column: improvement_surcharge =====
```

```
Descriptive Stats:
 count   1550456.00
mean          1.00
std           0.03
min          -1.00
25%           1.00
50%           1.00
75%           1.00
max           1.00
Name: improvement_surcharge, dtype: float64

Top 5 Unique Values:
 improvement_surcharge
1.00     1548465
0.30        1197
0.00         734
-1.00         60
Name: count, dtype: int64

===== Summary for Column: total_amount =====

Descriptive Stats:
 count   1550456.00
mean         28.95
std         117.21
min          -5.75
25%          15.96
50%          21.00
75%          30.72
max      143167.45
Name: total_amount, dtype: float64

Top 5 Unique Values:
 total_amount
16.80    22335
12.60    20256
21.00    18408
18.00    11852
15.12    11718
Name: count, dtype: int64

===== Summary for Column: congestion_surcharge =====

Descriptive Stats:
 count   1501862.00
mean          2.30
std           0.67
min          -2.50
25%           2.50
```

```
50%             2.50
75%             2.50
max             2.50
Name: congestion_surcharge, dtype: float64

Top 5 Unique Values:
 congestion_surcharge
2.50     1384054
0.00      117764
-2.50         44
Name: count, dtype: int64

===== Summary for Column: airport_fee =====

Descriptive Stats:
 count   148483.00
mean         0.11
std          0.35
min         -1.25
25%          0.00
50%          0.00
75%          0.00
max          1.25
Name: airport_fee, dtype: float64

Top 5 Unique Values:
 airport_fee
0.00     135529
1.25      12953
-1.25         1
Name: count, dtype: int64

===== Summary for Column: pickup_date =====

Top 5 Frequent Values:
 pickup_date
2023-05-18    6679
2023-05-17    6556
2023-10-28    6497
2023-05-11    6454
2023-10-26    6442
Name: count, dtype: int64

===== Summary for Column: pickup_hour =====

Descriptive Stats:
 count   1550456.00
mean         14.25
std           5.82
min           0.00
```

```
25%            11.00
50%            15.00
75%            19.00
max            23.00
Name: pickup_hour, dtype: float64

Top 5 Unique Values:
 pickup_hour
18.00    109476
17.00    104362
19.00     97891
16.00     95739
15.00     95302
Name: count, dtype: int64

===== Summary for Column: Airport_fee =====

Descriptive Stats:
 count   1353379.00
mean          0.15
std           0.47
min          -1.75
25%           0.00
50%           0.00
75%           0.00
max           1.75
Name: Airport_fee, dtype: float64

Top 5 Unique Values:
 Airport_fee
0.00     1232916
1.75       93582
1.25       26870
-1.75          7
-1.25          3
Name: count, dtype: int64
```

We can see there are columns with negative values. We will try to deal with them. For this we will first check their counts in respective columns.

```
for col in sampled_data.columns:
    if pd.api.types.is_numeric_dtype(sampled_data[col]):
        negative_count = (sampled_data[col] < 0).sum()
        if negative_count > 0:
            print(f"Column '{col}' has {negative_count} negative value(s).")

Column 'extra' has 2 negative value(s).
Column 'mta_tax' has 56 negative value(s).
Column 'improvement_surcharge' has 60 negative value(s).
```

```
Column 'total_amount' has 60 negative value(s).
Column 'congestion_surcharge' has 44 negative value(s).
Column 'airport_fee' has 1 negative value(s).
Column 'Airport_fee' has 10 negative value(s).
```

Since the count of these negative values is very small compared to size of the sample. We will replace all the negatives with 0.

```
numeric_cols = sampled_data.select_dtypes(include=[np.number]).columns
sampled_data[numeric_cols] =
sampled_data[numeric_cols].applymap(lambda x: 0 if x < 0 else x)

<ipython-input-24-29d9f3e422fe>:2: FutureWarning: DataFrame.applymap
has been deprecated. Use DataFrame.map instead.
  sampled_data[numeric_cols] =
sampled_data[numeric_cols].applymap(lambda x: 0 if x < 0 else x)
```

Now, we will go about finding missing values. For this, we will check their count and then decide the necessary course of action.

```
sampled_data.isnull().sum()

VendorID                       0
tpep_pickup_datetime           0
tpep_dropoff_datetime          0
passenger_count            48594
trip_distance                  0
RatecodeID                 48594
store_and_fwd_flag         48594
PULocationID                   0
DOLocationID                   0
payment_type                   0
fare_amount                    1
extra                          1
mta_tax                        1
tip_amount                     1
tolls_amount                   1
improvement_surcharge          1
total_amount                   1
congestion_surcharge       48595
airport_fee              1401974
pickup_date                    1
pickup_hour                    1
Airport_fee               197078
dtype: int64
```

The null values are present in 5 columns of the data. We will try and populate the values by considering mode for passenger_count and Ratecode, for congestion charge and airport_fee we will try and find the relationship between Drop location and the null value columns. Then we can decide the most optimal value to repopulate the null values. now, with store_and_fwd_flag column, we can just drop it as it holds no significance in our analysis.

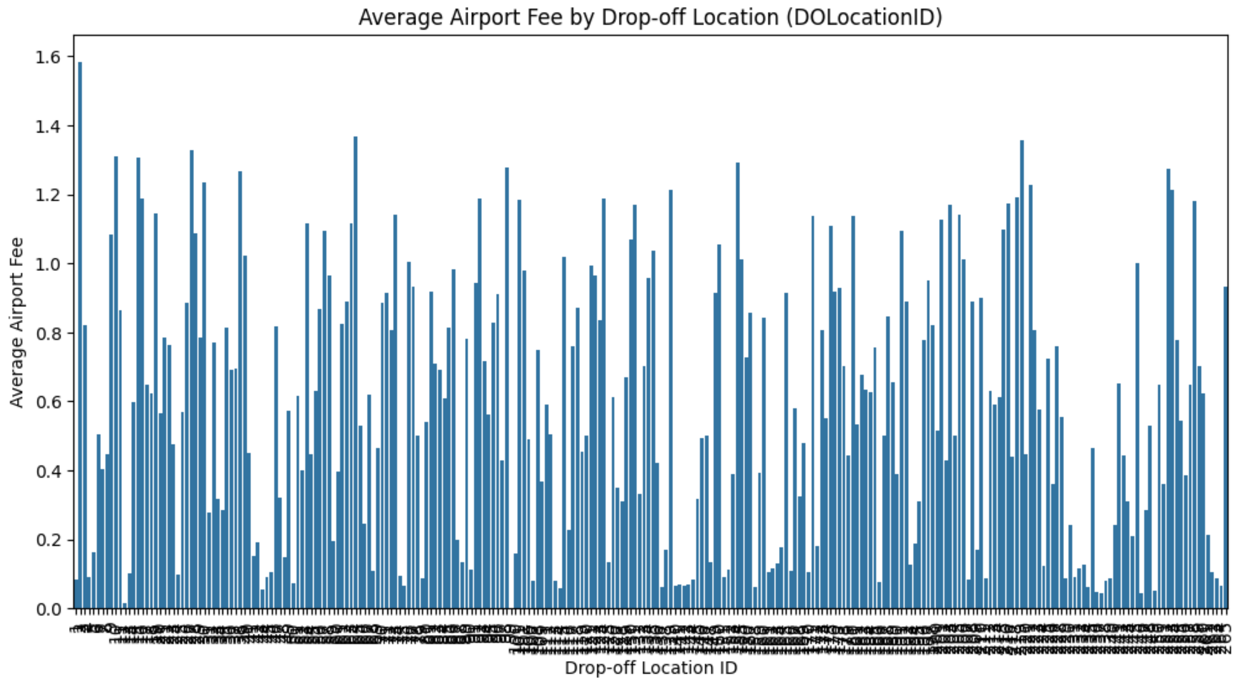
```
mode_passenger = sampled_data['passenger_count'].mode()[0]
sampled_data['passenger_count'].fillna(mode_passenger, inplace=True)

mode_ratecode = sampled_data['RatecodeID'].mode()[0]
sampled_data['RatecodeID'].fillna(mode_ratecode, inplace=True)

print("Nulls handled and column dropped successfully.")
```

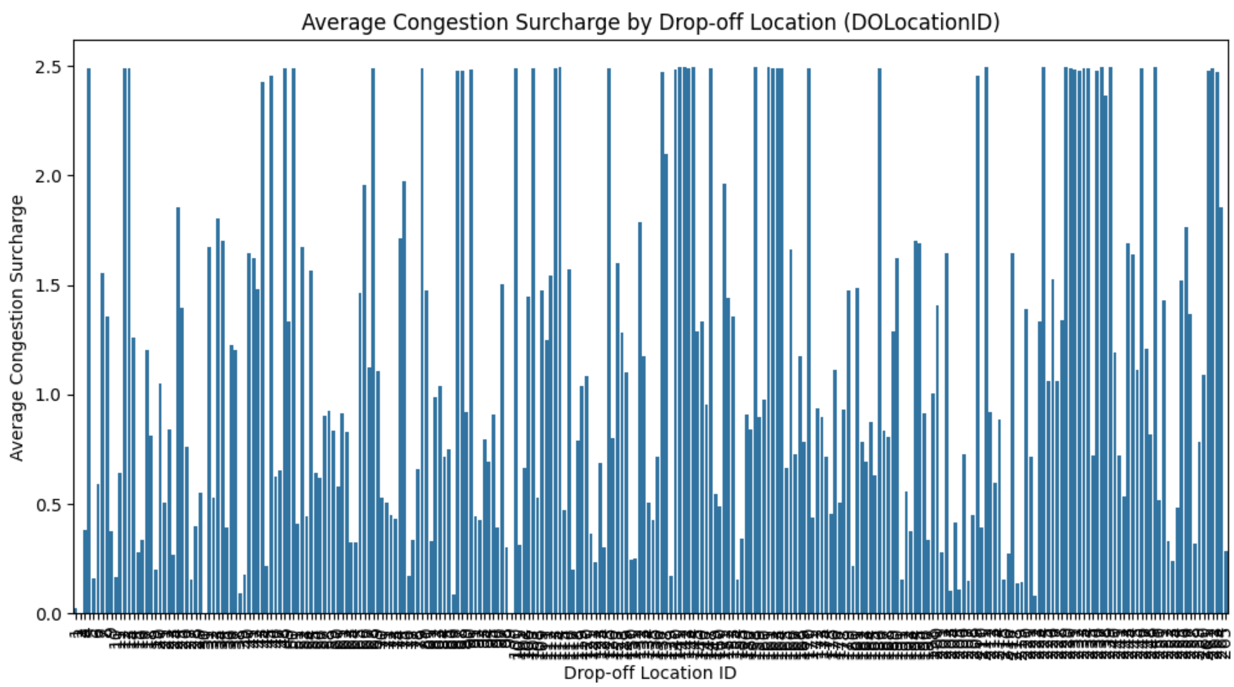
```
Nulls handled and column dropped successfully.

<ipython-input-152-4b64347b754f>:2: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  sampled_data['passenger_count'].fillna(mode_passenger, inplace=True)
<ipython-input-152-4b64347b754f>:5: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  sampled_data['RatecodeID'].fillna(mode_ratecode, inplace=True)
```

```
airport_fee_by_location = sampled_data.groupby('DOLocationID')
['Airport_fee'].agg(['count', 'sum', 'mean']).reset_index()

locations_with_airport_fee =
airport_fee_by_location[airport_fee_by_location['sum'] > 0]

print(locations_with_airport_fee.sort_values('mean', ascending=False))
```

```
     DOLocationID  count      sum  mean
1               2      3     4.75  1.58
63             64    157   214.50  1.37
213           219    585   794.00  1.36
26             27     32    42.50  1.33
9              10   1345  1760.25  1.31
..            ...    ...      ...   ...
243           249  22459  1171.75  0.05
230           236  58476  2781.50  0.05
240           246  26391  1109.50  0.04
231           237  56109  2342.50  0.04
11             12    747    11.75  0.02

[259 rows x 4 columns]
```

```
airport_fee_by_location = sampled_data.groupby('DOLocationID')
['Airport_fee'].mean().reset_index()
congestion_by_location = sampled_data.groupby('DOLocationID')
['congestion_surcharge'].mean().reset_index()

plt.figure(figsize=(12,6))
sns.barplot(data=airport_fee_by_location, x='DOLocationID',
y='Airport_fee')
plt.title('Average Airport Fee by Drop-off Location (DOLocationID)')
plt.xlabel('Drop-off Location ID')
plt.ylabel('Average Airport Fee')
plt.xticks(rotation=90)
plt.show()

plt.figure(figsize=(12,6))
sns.barplot(data=congestion_by_location, x='DOLocationID',
y='congestion_surcharge')
plt.title('Average Congestion Surcharge by Drop-off Location
(DOLocationID)')
plt.xlabel('Drop-off Location ID')
plt.ylabel('Average Congestion Surcharge')
plt.xticks(rotation=90)
plt.show()
```





We can see there is no significant relational strenght between DO Location and Aiport Fee or Congestion rate. We will therefore, replace all null values with 0 since we cannot be sure if all taxis with null airport fees travlled to the airport or if the congestion rate was due to the drop location.

```
sampled_data['congestion_surcharge'] =
sampled_data['congestion_surcharge'].fillna(0)
sampled_data['Airport_fee'] = sampled_data['Airport_fee'].fillna(0)
```

Lets verify once again to ensure elimination of all null values.

```
sampled_data.isnull().sum()
```

```
VendorID                           0
tpep_pickup_datetime               0
tpep_dropoff_datetime              0
passenger_count                    0
trip_distance                      0
RatecodeID                         0
PULocationID                       0
DOLocationID                       0
payment_type                       0
fare_amount                        1
extra                              1
mta_tax                            1
tip_amount                         1
tolls_amount                       1
improvement_surcharge              1
total_amount                       1
congestion_surcharge               0
airport_fee                  1401974
pickup_date                        1
pickup_hour                        1
Airport_fee                        0
dtype: int64
```

We have two columns one named airport_fee and the other Airport_fee, we will now merge the two. For that, we will first a common Airport_fee column then drop the original aiport_fee column and then finally ensure the merge.

```
if 'airport_fee' in sampled_data.columns:
    sampled_data.drop(columns=['airport_fee'], inplace=True)
print("\n Merged 'Airport_fee' column preview:")
print(sampled_data[['Airport_fee']].head())
if 'airport_fee' in sampled_data.columns:
    sampled_data.drop(columns=['airport_fee'], inplace=True)
print("\n Merged 'Airport_fee' column preview:")
print(sampled_data[['Airport_fee']].head())
```

```
 Merged 'Airport_fee' column preview:
```

```
   Airport_fee
0         0.00
1         0.00
2         0.00
3         0.00
4         0.00

 Merged 'Airport_fee' column preview:
   Airport_fee
0         0.00
1         0.00
2         0.00
3         0.00
4         0.00
```

```
sampled_data.isnull().sum()
```

```
VendorID                 0
tpep_pickup_datetime     0
tpep_dropoff_datetime    0
passenger_count          0
trip_distance            0
RatecodeID               0
PULocationID             0
DOLocationID             0
payment_type             0
fare_amount              1
extra                    1
mta_tax                  1
tip_amount               1
tolls_amount             1
improvement_surcharge    1
total_amount             1
congestion_surcharge     0
pickup_date              1
pickup_hour              1
Airport_fee              0
dtype: int64
```

We have ensured 0 negative values, nil nulls. We can now start Outlier Analysis. For this we will first describe the data and check the vital statistics.

```
sampled_data.describe()
```

{"summary":"{\n \"name\": \"sampled_data\",\n \"rows\": 8,\n
\"fields\": [\n   {\n   \"column\": \"VendorID\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
548168.6131113545,\n   \"min\": 0.4491971268022429,\n
\"max\": 1550457.0,\n   \"num_unique_values\": 6,\n
\"samples\": [\n   1550457.0,\n   1.7310451047658852,\n

```
6.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"passenger_count\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 548168.6102896177,\n        \"min\":
0.0,\n        \"max\": 1550457.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1550457.0,\n          1.3549031027626048,\n
9.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_distance\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 543580.5122017618,\n        \"min\":
0.0,\n        \"max\": 1550457.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          3.89983046933904,\n          1.8,\n
1550457.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"RatecodeID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 548163.6883011647,\n        \"min\":
1.0,\n        \"max\": 1550457.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          1.5906142511530472,\n          99.0,\n
7.116676392896066\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PULocationID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 548117.6586114698,\n        \"min\":
1.0,\n        \"max\": 1550457.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          165.14424714777644,\n          162.0,\n
1550457.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOLocationID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 548118.3857653191,\n        \"min\":
1.0,\n        \"max\": 1550457.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          163.89131398032967,\n          162.0,\n
1550457.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"payment_type\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 548168.8912882512,\n        \"min\":
0.0,\n        \"max\": 1550457.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1550457.0,\n          1.168383257323486,\n
4.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"fare_amount\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 543242.855074238,\n        \"min\":
0.0,\n        \"max\": 1550456.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          19.89479720804718,\n          13.5,\n
1550456.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"extra\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 548167.5749152862,\n        \"min\": 0.0,\n        \"max\":
1550456.0,\n        \"num_unique_values\": 7,\n        \"samples\": [\
n          1550456.0,\n          1.6040665391342936,\n          2.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
```

```
}\n    },\n    {\n      \"column\": \"mta_tax\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
548168.6705264231,\n        \"min\": 0.0,\n        \"max\":
1550456.0,\n        \"num_unique_values\": 6,\n        \"samples\": [\
n          1550456.0,\n          0.49539538690552987,\n          4.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tip_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
548156.9152908858,\n        \"min\": 0.0,\n        \"max\":
1550456.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          3.535883262730451,\n          2.82,\n          1550456.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tolls_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
548161.6151347755,\n        \"min\": 0.0,\n        \"max\":
1550456.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\
n          0.5961541443291523,\n          143.0,\n
2.1818403354979434\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"improvement_surcharge\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 548168.7217746994,\n
\"min\": 0.0,\n        \"max\": 1550456.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.9989474709375828,\n          1.0,\n          0.02982105500497115\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
543240.9554061996,\n        \"min\": 0.0,\n        \"max\":
1550456.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          28.948506974722253,\n          21.0,\n          1550456.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"congestion_surcharge\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
548168.6724428224,\n        \"min\": 0.0,\n        \"max\":
1550457.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\
n          2.231687173523677,\n          2.5,\n
0.7738156625108972\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"pickup_hour\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 548164.5279380416,\n        \"min\":
0.0,\n        \"max\": 1550456.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          14.248035416677416,\n          15.0,\n
1550456.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Airport_fee\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 548169.2121571288,\n        \"min\":
0.0,\n        \"max\": 1550457.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.1272895668825385,\n          1.75,\n
```

0.442405093597796\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe"}

Trip Distance has a max of 126360. This is highly unlikely, lets group the data and check the frequency of this value.

Let us now drop all rows with these erroneous values.

Now let us repeat the same with fare_amount. We will do this upto 500+ with custom bins.

Lets eliminate all values above 200 since the mean, 75th percentile and 50th percentile are well below 100 and values between 100-200 are also diminishing.

```
fare_bins = [0, 5, 10, 20, 40, 60, 100, 200, np.inf]


fare_labels = ['0-5', '5-10', '10-20', '20-40', '40-60', '60-100',
'100-200', '200+']

sampled_data['fare_bin'] = pd.cut(sampled_data['fare_amount'],
bins=fare_bins, labels=fare_labels, right=False)


print(sampled_data['fare_bin'].value_counts().sort_index())
```

```
fare_bin
0-5         28038
5-10       397338
10-20      682746
20-40      271552
40-60       78119
60-100      87504
100-200      4557
200+          602
Name: count, dtype: int64
```

```
bins_to_remove = ['200+']

sampled_data =
sampled_data[~sampled_data['fare_bin'].isin(bins_to_remove)]

print(sampled_data['fare_bin'].value_counts().sort_index())
```

```
fare_bin
0-5         28038
5-10       397338
10-20      682746
20-40      271552
40-60       78119
60-100      87504
```

```
100-200      4557
200+            0
Name: count, dtype: int64
```

```
sampled_data.describe()
```


```
{"summary":"{\n  \"name\": \"sampled_data\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"VendorID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547955.7739707057,\n        \"min\": 0.4492119382969846,\n
\"max\": 1549855.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1549855.0,\n          1.7310206438666842,\n
6.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"passenger_count\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547955.7711532768,\n        \"min\":
0.0,\n        \"max\": 1549855.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1549855.0,\n          1.3548506150575377,\n
9.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_distance\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 543368.3905784576,\n        \"min\":
0.0,\n        \"max\": 1549855.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          3.880636459539765,\n          1.8,\n
1549855.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"RatecodeID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547950.8492809926,\n        \"min\":
1.0,\n        \"max\": 1549855.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          1.5892293150004355,\n          99.0,\n
7.115674583040949\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PULocationID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547904.8192748892,\n        \"min\":
1.0,\n        \"max\": 1549855.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          165.14646531449716,\n          162.0,\n
1549855.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOLocationID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547905.5489293843,\n        \"min\":
1.0,\n        \"max\": 1549855.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          163.8592190882373,\n          162.0,\n
1549855.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"payment_type\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547956.0521555016,\n        \"min\":
0.0,\n        \"max\": 1549855.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1549855.0,\n          1.1683144552232305,\n
4.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"fare_amount\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547941.9179404634,\n        \"min\":
0.0,\n        \"max\": 1549854.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          19.701629230882403,\n          13.5,\n
1549854.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"extra\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 547954.7357658551,\n        \"min\": 0.0,\n        \"max\":
1549854.0,\n        \"num_unique_values\": 7,\n        \"samples\": [\
n          1549854.0,\n          1.6042981080798584,\n          2.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"mta_tax\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547955.8314121603,\n        \"min\": 0.0,\n        \"max\":
1549854.0,\n        \"num_unique_values\": 6,\n        \"samples\": [\
n          1549854.0,\n          0.4955239009609938,\n          4.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tip_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547944.0791694153,\n        \"min\": 0.0,\n        \"max\":
1549854.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          3.5296376884532368,\n          2.82,\n          1549854.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tolls_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547948.7768503543,\n        \"min\": 0.0,\n        \"max\":
1549854.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\
n          0.5931735053753447,\n          143.0,\n
2.1678779761342515\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"improvement_surcharge\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 547955.8826366144,\n
\"min\": 0.0,\n        \"max\": 1549854.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.9989513850982096,\n          1.0,\n          0.029756711944775215\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547934.9597034161,\n        \"min\": 0.0,\n        \"max\":
1549854.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          28.746824946091685,\n          21.0,\n          1549854.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"congestion_surcharge\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547955.8333121182,\n        \"min\": 0.0,\n        \"max\":
1549855.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\
n          2.2324523907075178,\n          2.5,\n
0.7728438945793085\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"pickup_hour\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547951.688803613,\n        \"min\":
0.0,\n        \"max\": 1549854.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          14.248189829493617,\n          15.0,\n
1549854.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Airport_fee\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547956.3730503618,\n        \"min\":
0.0,\n        \"max\": 1549855.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.12702881882498684,\n          1.75,\n
0.44198534953147456\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


The data now looks uniform, we will go ahead with our analysis.

Remove Trips with 7+ Passengers.

```
sampled_data = sampled_data[sampled_data['passenger_count'] < 7]
```

Lets find out trips wheretrip_distance is close to zero but fare is 300+ and remove them.

```
condition = (sampled_data['trip_distance'] < 0.1) &
(sampled_data['fare_amount'] > 300)
sampled_data = sampled_data[~condition]
```

Now we will remove trips where trip_distance=0 and fare_amount is also 0 but pickup and drop locations are different.

```
condition = (
    (sampled_data['trip_distance'] == 0) &
    (sampled_data['fare_amount'] == 0) &
    (sampled_data['PULocationID'] != sampled_data['DOLocationID'])
)
sampled_data = sampled_data[~condition]
```

We have already removed all trips where distance>200, but we will ensure the same by running it again.

```
sampled_data = sampled_data[sampled_data['trip_distance'] <= 250]

sampled_data.describe()
```


```
{"summary":"{\n  \"name\": \"sampled_data\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"VendorID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547919.7115245999,\n        \"min\": 0.44920715141349743,\n
\"max\": 1549753.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1549753.0,\n          1.731030686825578,\n
6.0\n        ],\n        \"semantic_type\": \"\",\n
```

```
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"passenger_count\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547919.8602397443,\n        \"min\":
0.0,\n        \"max\": 1549753.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1549753.0,\n          1.3547991196016398,\n
6.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_distance\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547914.4418604654,\n        \"min\":
0.0,\n        \"max\": 1549753.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          3.4632751670750124,\n          1.8,\n
1549753.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"RatecodeID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547914.7869755216,\n        \"min\":
1.0,\n        \"max\": 1549753.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          1.5888151208611954,\n          99.0,\n
7.113311111571072\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PULocationID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547868.7568396934,\n        \"min\":
1.0,\n        \"max\": 1549753.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          165.14706408053414,\n          162.0,\n
1549753.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOLocationID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547869.4866204822,\n        \"min\":
1.0,\n        \"max\": 1549753.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          163.8576560264765,\n          162.0,\n
1549753.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"payment_type\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547919.9897130381,\n        \"min\":
0.0,\n        \"max\": 1549753.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1549753.0,\n          1.1683074657703518,\n
4.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"fare_amount\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 547905.8555641032,\n        \"min\":
0.0,\n        \"max\": 1549752.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          19.701446876661553,\n          13.5,\n
1549752.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"extra\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 547918.6733162447,\n        \"min\": 0.0,\n        \"max\":
1549752.0,\n        \"num_unique_values\": 7,\n        \"samples\": [\
n          1549752.0,\n          1.604371757545724,\n          2.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"mta_tax\",\n
```

```
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547919.7689699387,\n        \"min\": 0.0,\n        \"max\":
1549752.0,\n        \"num_unique_values\": 6,\n        \"samples\": [\
n          1549752.0,\n          0.49554102850004395,\n          4.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tip_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547908.0167278005,\n        \"min\": 0.0,\n        \"max\":
1549752.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          3.529677671008064,\n          2.82,\n          1549752.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tolls_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547912.7144123715,\n        \"min\": 0.0,\n        \"max\":
1549752.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\
n          0.5931354177958793,\n          143.0,\n
2.1677634613653822\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"improvement_surcharge\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 547919.8202135095,\n
\"min\": 0.0,\n        \"max\": 1549752.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.9989799658267906,\n          1.0,\n          0.029277988791451\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547898.8973282474,\n        \"min\": 0.0,\n        \"max\":
1549752.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          28.746818374810925,\n          21.0,\n          1549752.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"congestion_surcharge\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547919.770867894,\n        \"min\": 0.0,\n        \"max\": 1549753.0,\
n        \"num_unique_values\": 5,\n        \"samples\": [\n
2.232570287007026,\n          2.5,\n          0.772693999178199\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"pickup_hour\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547915.6263547871,\n        \"min\": 0.0,\n        \"max\":
1549752.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          14.24823068465148,\n          15.0,\n          1549752.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Airport_fee\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
547920.310604433,\n        \"min\": 0.0,\n        \"max\": 1549753.0,\
n        \"num_unique_values\": 5,\n        \"samples\": [\n
0.12702959761974972,\n          1.75,\n          0.44198631861365956\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```

Let us take a count of all payment_type with value '0'

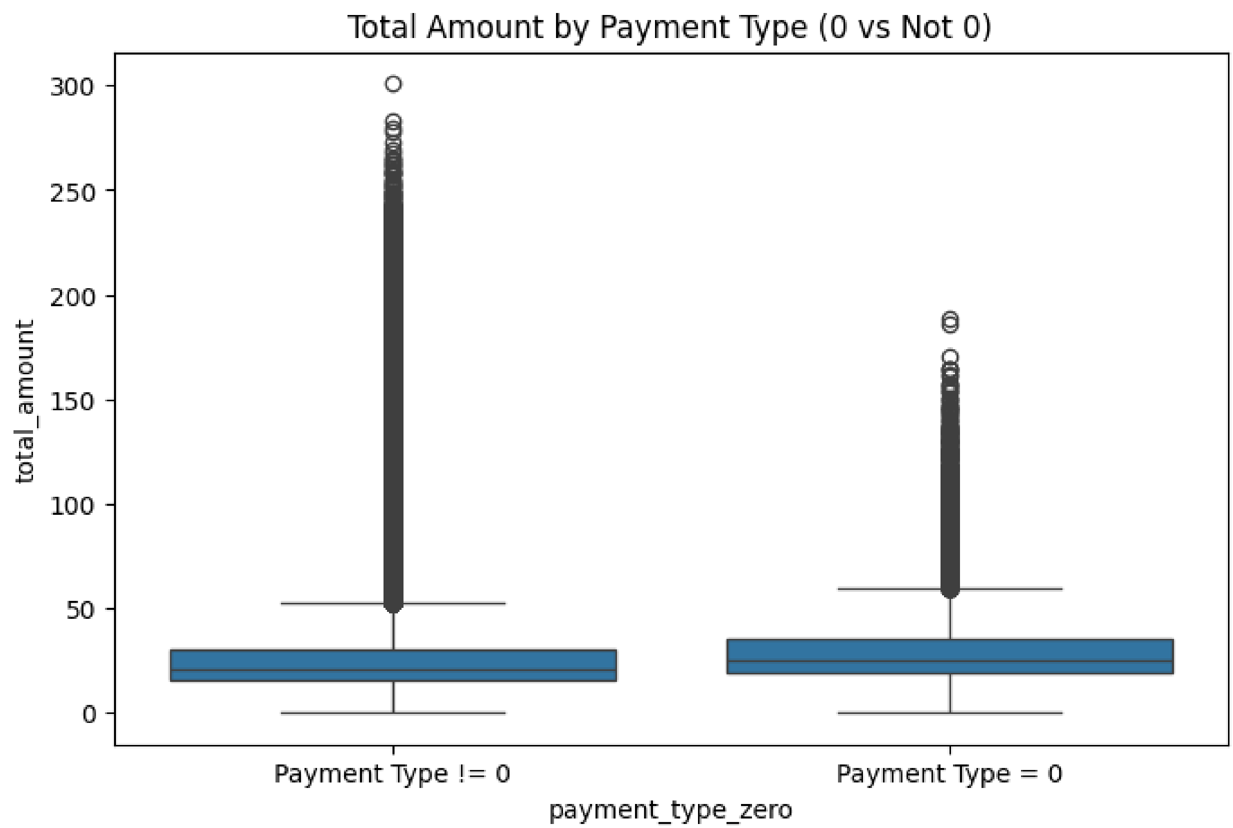
```
num_payment_type_0 = sampled_data[sampled_data['payment_type'] ==
0].shape[0]
print(f"Number of rows with payment_type 0: {num_payment_type_0}")
```

```
Number of rows with payment_type 0: 48569
```

The count is high to just drop the rows with payment_type=0, we will therefore first try and see if there is a relationship between the fare_amount and payment_type to confirm if dropping the same will be wise.

```
sampled_data['payment_type_zero'] = sampled_data['payment_type'] == 0


print(sampled_data.groupby('payment_type_zero')
['total_amount'].describe())


plt.figure(figsize=(8,5))
sns.boxplot(x='payment_type_zero', y='total_amount',
data=sampled_data)
plt.xticks([0,1], ['Payment Type != 0', 'Payment Type = 0'])
plt.title('Total Amount by Payment Type (0 vs Not 0)')
plt.show()
```

```
                        count  mean   std  min   25%   50%   75%    max
payment_type_zero
False              1501183.00 28.70 22.20 0.00 15.95 21.00 30.60 300.95
True                 48569.00 30.30 17.76 0.00 19.16 25.01 35.35 188.40
```



The Payment type is not defined and more importantly, the plotting above is very close to the plotting values without 0. Hence deleting them wouldn't affect our analysis.

```
sampled_data = sampled_data[sampled_data['payment_type'] != 0]
```

Let us now standardise using the formula: z=(x-u)/sd for each column. We will them include the standardised columns in the data set with the suffix _std. The columns that need standardisation are trip_distance, fare_amount, extra, tip_amount, tolls_amount, total_amount, congestion_surcharge

```
cols_to_standardize = ['trip_distance', 'fare_amount', 'extra',
'tip_amount', 'tolls_amount', 'total_amount', 'congestion_surcharge']

for col in cols_to_standardize:
    mean = sampled_data[col].mean()
    std = sampled_data[col].std()
    sampled_data[col + '_std'] = (sampled_data[col] - mean) / std

sampled_data.describe()

{"type":"dataframe"}
```

The standardised values are now presenting negative values, so we will not consider them and just drop standardisation altogether.

```
sampled_data.drop(columns=[col for col in sampled_data.columns if
col.endswith('_std')], inplace=True)

sampled_data.describe()
```


```
{"summary":"{\n  \"name\": \"sampled_data\",\n  \"rows\": 8,\n
\"fields\": [\n    {\n      \"column\": \"VendorID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
530748.1791690629,\n        \"min\": 0.4423770238302506,\n
\"max\": 1501184.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          1.7330293954638472,\n          2.0,\n
0.4423770238302506\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"passenger_count\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 530748.124436977,\n        \"min\":
0.0,\n        \"max\": 1501184.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1501184.0,\n          1.3662782177268076,\n
6.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_distance\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 530742.7087505853,\n        \"min\":
0.0,\n        \"max\": 1501184.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          3.457505828732521,\n          1.8,\n
1501184.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"RatecodeID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 530743.0456940726,\n        \"min\":
1.0,\n        \"max\": 1501184.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          1.6078655248124147,\n          99.0,\n
7.226665342417576\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"PULocationID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 530697.0803416754,\n        \"min\":
1.0,\n        \"max\": 1501184.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          165.31675930465553,\n          162.0,\n
1501184.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"DOLocationID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 530697.6912980205,\n        \"min\":
1.0,\n        \"max\": 1501184.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          164.1216266626876,\n          162.0,\n
1501184.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"payment_type\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 530748.2046150799,\n        \"min\":
0.4655100596798764,\n        \"max\": 1501184.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
1.2061066464870396,\n          4.0,\n          0.4655100596798764\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"fare_amount\",\n
```

```
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
530734.1214412883,\n        \"min\": 0.0,\n        \"max\":
1501183.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          19.606649882126298,\n          13.5,\n          1501183.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"extra\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
530747.2668512461,\n        \"min\": 0.0,\n        \"max\":
1501183.0,\n        \"num_unique_values\": 7,\n        \"samples\": [\
n          1501183.0,\n          1.650448099931854,\n          2.5\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"mta_tax\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
530748.0343449261,\n        \"min\": 0.0,\n        \"max\":
1501183.0,\n        \"num_unique_values\": 6,\n        \"samples\": [\
n          1501183.0,\n          0.49555597152379177,\n          4.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tip_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
530736.2787179672,\n        \"min\": 0.0,\n        \"max\":
1501183.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          3.544712743216518,\n          2.85,\n          1501183.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tolls_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
530740.9802127405,\n        \"min\": 0.0,\n        \"max\":
1501183.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\
n          0.5913388507597007,\n          143.0,\n
2.1625395578613094\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"improvement_surcharge\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 530748.0857443786,\n
\"min\": 0.0,\n        \"max\": 1501183.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.9992202816045751,\n          1.0,\n          0.025897806554895163\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"total_amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
530727.1658223224,\n        \"min\": 0.0,\n        \"max\":
1501183.0,\n        \"num_unique_values\": 8,\n        \"samples\": [\
n          28.69672645506911,\n          21.0,\n          1501183.0\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"congestion_surcharge\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
530748.0377414159,\n        \"min\": 0.0,\n        \"max\":
1501184.0,\n        \"num_unique_values\": 5,\n        \"samples\": [\
n          2.304802409298260 6,\n          2.5,\n
0.6707400219477264\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

```
\"pickup_hour\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 530743.8914752286,\n        \"min\":
0.0,\n        \"max\": 1501183.0,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          14.279278409094694,\n          15.0,\n
1501183.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Airport_fee\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 530748.5754412136,\n        \"min\":
0.0,\n        \"max\": 1501184.0,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          0.131139487231412,\n          1.75,\n
0.4484788918911382\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


lets begin the EDA process now.

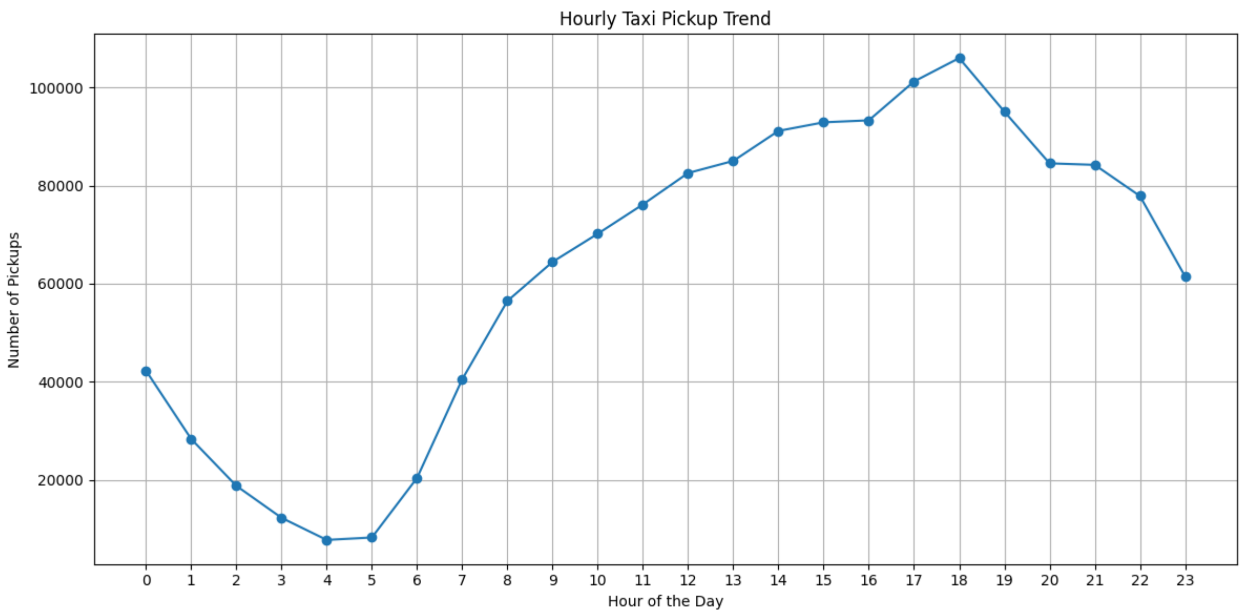
```
sampled_data.columns.tolist()
```

```
['VendorID',
 'tpep_pickup_datetime',
 'tpep_dropoff_datetime',
 'passenger_count',
 'trip_distance',
 'RatecodeID',
 'PULocationID',
 'DOLocationID',
 'payment_type',
 'fare_amount',
 'extra',
 'mta_tax',
 'tip_amount',
 'tolls_amount',
 'improvement_surcharge',
 'total_amount',
 'congestion_surcharge',
 'pickup_date',
 'pickup_hour',
 'Airport_fee',
 'fare_bin',
 'payment_type_zero']
```

We will now analyse the hourly trends in Taxi Pickup

```
hourly_counts = sampled_data.groupby('pickup_hour').size()

plt.figure(figsize=(12, 6))
plt.plot(hourly_counts.index, hourly_counts.values, marker='o')
plt.title('Hourly Taxi Pickup Trend')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Pickups')
```

```
plt.xticks(range(0, 24))
plt.grid(True)
plt.tight_layout()
plt.show()
```




```
print(hourly_counts.sort_index())
```

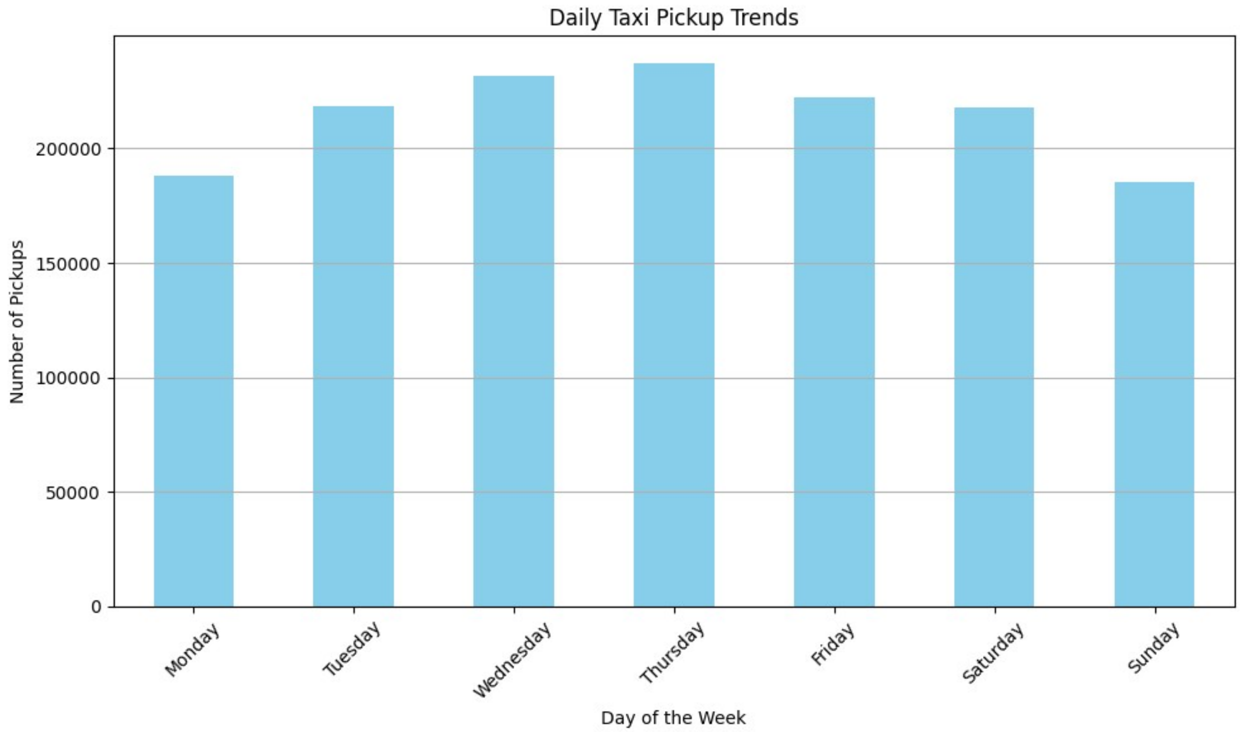
```
pickup_hour
0.00      42254
1.00      28315
2.00      18751
3.00      12227
4.00       7716
5.00       8210
6.00      20368
7.00      40495
8.00      56473
9.00      64447
10.00     70186
11.00     76166
12.00     82576
13.00     85047
14.00     91170
15.00     92919
16.00     93324
17.00    101249
18.00    106025
19.00     95099
20.00     84550
21.00     84240
```

```
22.00     77892
23.00     61484
dtype: int64
```

The above data and the graph shows that the taxi use increases its lowest from 4 am in the morning, increases gradually till 6pm and then starts to decrease gradually again.

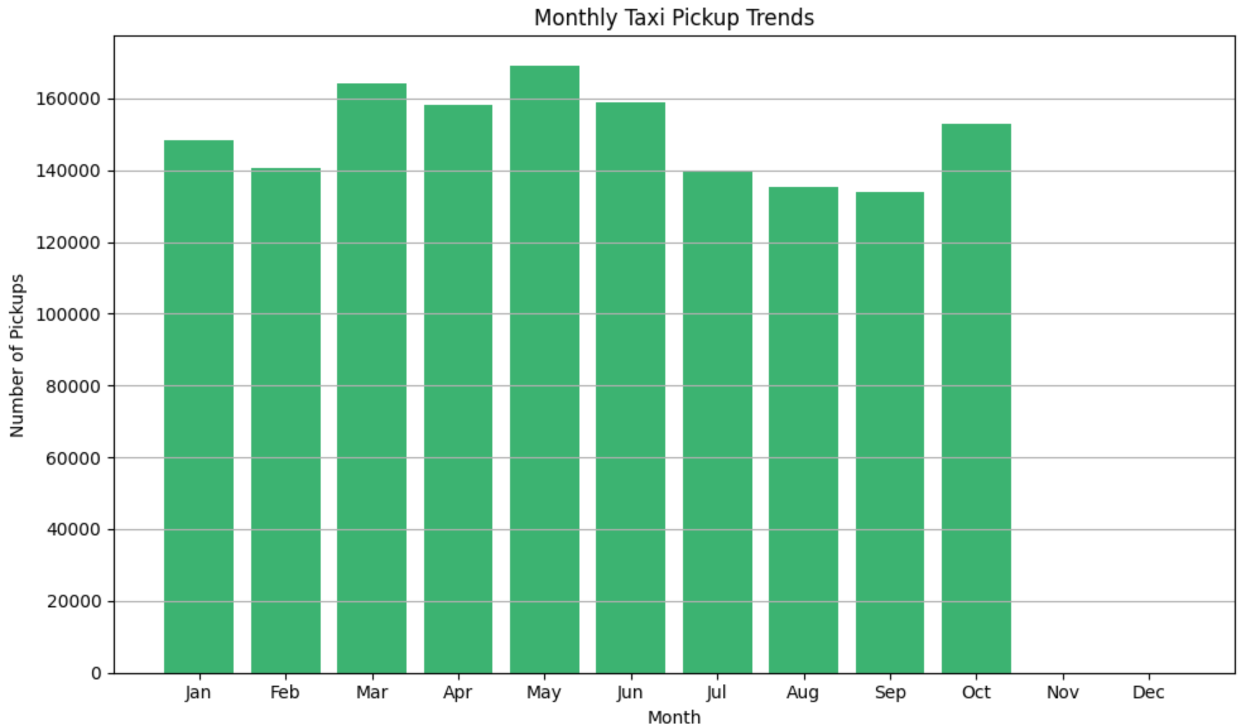
Now we will do weekly trend analysis by the days of the week. For this we will first extract the days, then group the data by days and finally make a plot.

```
sampled_data['pickup_date'] =
pd.to_datetime(sampled_data['pickup_date'])

sampled_data['pickup_day'] = sampled_data['pickup_date'].dt.dayofweek

day_map = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday',
           4: 'Friday', 5: 'Saturday', 6: 'Sunday'}
sampled_data['pickup_day_name'] =
sampled_data['pickup_day'].map(day_map)

daily_counts = sampled_data['pickup_day_name'].value_counts().reindex(
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday'])

plt.figure(figsize=(10, 6))
daily_counts.plot(kind='bar', color='skyblue')
plt.title('Daily Taxi Pickup Trends')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



For the monthly trends we will first extract the months, group the data by month and plot the data.

```
sampled_data['pickup_date'] =
pd.to_datetime(sampled_data['pickup_date'])

sampled_data['pickup_month'] = sampled_data['pickup_date'].dt.month

monthly_counts =
sampled_data['pickup_month'].value_counts().sort_index()

months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
          'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']

plt.figure(figsize=(10, 6))
plt.bar(monthly_counts.index, monthly_counts.values,
color='mediumseagreen')
plt.xticks(ticks=range(1, 13), labels=months)
plt.title('Monthly Taxi Pickup Trends')
plt.xlabel('Month')
plt.ylabel('Number of Pickups')
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



Financial Analysis.

Checking to see if there are negative or zero values.

```
financial_cols = ['fare_amount', 'tip_amount', 'total_amount',
'trip_distance']

for col in financial_cols:
    zeros = (sampled_data[col] == 0).sum()
    negatives = (sampled_data[col] < 0).sum()
    print(f"{col} - Zeros: {zeros}, Negatives: {negatives}")

fare_amount - Zeros: 473, Negatives: 0
tip_amount - Zeros: 338805, Negatives: 0
total_amount - Zeros: 266, Negatives: 0
trip_distance - Zeros: 18917, Negatives: 0
```

We can now drop fare_amount=0 and total_amount=0 as they are not relevant for financial analysis. trip distance=0 has to be verified for differnce in pick up and drop locations. If they are same then they can be dropped too. First lets drop fare_amount=0, total_amount=0 and trip_distance=0 where pickup location and drop location are the same.

```
zero_fare_or_total = sampled_data[
    (sampled_data['fare_amount'] == 0) | (sampled_data['total_amount']
== 0)
]
```

```
print("Rows with fare_amount = 0 or total_amount = 0:
{len(zero_fare_or_total)}")

sampled_data =
sampled_data.drop(zero_fare_or_total.index).reset_index(drop=True)


print("Cleaned dataset shape:", sampled_data.shape)
```

```
Rows with fare_amount = 0 or total_amount = 0:
{len(zero_fare_or_total)}
Cleaned dataset shape: (1500711, 25)
```

```
financial_cols = ['fare_amount', 'tip_amount', 'total_amount',
'trip_distance']

for col in financial_cols:
    zeros = (sampled_data[col] == 0).sum()
    negatives = (sampled_data[col] < 0).sum()
    print(f"{col} - Zeros: {zeros}, Negatives: {negatives}")
```

```
fare_amount - Zeros: 0, Negatives: 0
tip_amount - Zeros: 338348, Negatives: 0
total_amount - Zeros: 0, Negatives: 0
trip_distance - Zeros: 18694, Negatives: 0
```

```
suspicious_zero_distance = sampled_data[
    (sampled_data['trip_distance'] == 0) &
    (sampled_data['PULocationID'] != sampled_data['DOLocationID'])
]

print("Number of trips with zero distance but different pickup and
dropoff: {len(suspicious_zero_distance)}")

suspicious_zero_distance[['PULocationID', 'DOLocationID',
'trip_distance', 'fare_amount', 'total_amount']].head()
```

```
Number of trips with zero distance but different pickup and dropoff:
{len(suspicious_zero_distance)}
```


```
{"summary":"{\n  \"name\": \"suspicious_zero_distance[['PULocationID',
'DOLocationID', 'trip_distance', 'fare_amount', 'total_amount']]\",\n
\"rows\": 5,\n  \"fields\": [\n    {\n      \"column\":
\"PULocationID\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 89,\n        \"min\": 43,\n
\"max\": 264,\n        \"num_unique_values\": 5,\n        \"samples\":
[\n          74,\n          68,\n          43\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"DOLocationID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
97,\n        \"min\": 77,\n        \"max\": 264,\n
```

```
\"num_unique_values\": 4,\n        \"samples\": [\n          77,\n
264,\n          79\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"trip_distance\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.0,\n        \"min\": 0.0,\n
\"max\": 0.0,\n        \"num_unique_values\": 1,\n        \"samples\":
[\n          0.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"fare_amount\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 11.57181921739188,\n        \"min\":
12.8,\n        \"max\": 41.2,\n        \"num_unique_values\": 5,\n
\"samples\": [\n          41.2\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"total_amount\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 11.41880160086863,\n
\"min\": 22.2,\n        \"max\": 49.25,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          49.25\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```


```
financial_cols = ['fare_amount', 'tip_amount', 'total_amount',
'trip_distance']

for col in financial_cols:
    zeros = (sampled_data[col] == 0).sum()
    negatives = (sampled_data[col] < 0).sum()
    print(f"{col} - Zeros: {zeros}, Negatives: {negatives}")
```

```
fare_amount - Zeros: 0, Negatives: 0
tip_amount - Zeros: 338348, Negatives: 0
total_amount - Zeros: 0, Negatives: 0
trip_distance - Zeros: 18694, Negatives: 0
```

```
zero_distance_same_location = sampled_data[
    (sampled_data['trip_distance'] == 0) &
    (sampled_data['PULocationID'] == sampled_data['DOLocationID'])
]

print("Rows with trip_distance=0 and same pickup/dropoff:
{len(zero_distance_same_location)}")

sampled_data =
sampled_data.drop(zero_distance_same_location.index).reset_index(drop=
True)

print("Dataset shape after dropping:", sampled_data.shape)
```

```
Rows with trip_distance=0 and same pickup/dropoff:
{len(zero_distance_same_location)}
Dataset shape after dropping: (1488368, 25)
```

```
financial_cols = ['fare_amount', 'tip_amount', 'total_amount',
'trip_distance']

for col in financial_cols:
    zeros = (sampled_data[col] == 0).sum()
    negatives = (sampled_data[col] < 0).sum()
    print(f"{col} - Zeros: {zeros}, Negatives: {negatives}")
```

```
fare_amount - Zeros: 0, Negatives: 0
tip_amount - Zeros: 330322, Negatives: 0
total_amount - Zeros: 0, Negatives: 0
trip_distance - Zeros: 6351, Negatives: 0
```

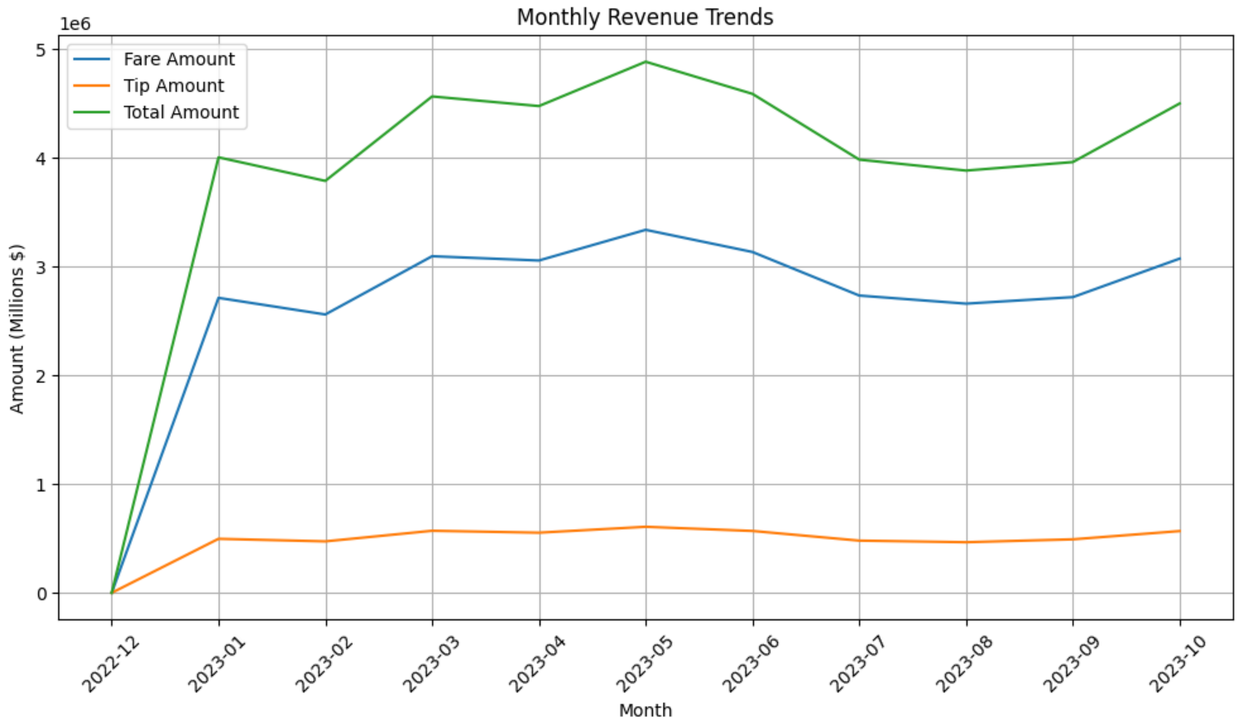
Let us now Analyse the monthly revenue. To analyze monthly revenue, first we need to group your data by the month extracted from the pickup datetime, then sum up relevant financial columns like fare_amount, tip_amount, and total_amount. Finally we will create a plot.

```
sampled_data['tpep_pickup_datetime'] =
pd.to_datetime(sampled_data['tpep_pickup_datetime'])

sampled_data['pickup_month'] =
sampled_data['tpep_pickup_datetime'].dt.to_period('M')

monthly_revenue = sampled_data.groupby('pickup_month').agg({
    'fare_amount': 'sum',
    'tip_amount': 'sum',
    'total_amount': 'sum'
}).reset_index()

print(monthly_revenue)
```

```
   pickup_month  fare_amount  tip_amount  total_amount
0       2022-12         6.50        2.00         13.50
1       2023-01   2713439.18   497080.22    4006018.16
2       2023-02   2560646.25   473327.29    3789163.94
3       2023-03   3096407.99   570952.52    4566123.18
4       2023-04   3056969.65   553280.09    4477577.24
5       2023-05   3339269.97   607484.61    4885496.42
6       2023-06   3135310.65   569282.90    4589835.35
7       2023-07   2734553.43   480241.56    3985003.45
8       2023-08   2660198.05   465615.59    3884163.61
9       2023-09   2720088.05   492293.13    3962623.59
10      2023-10   3074377.14   568180.97    4501257.12
```

```
plt.figure(figsize=(12,6))
plt.plot(monthly_revenue['pickup_month'].astype(str),
monthly_revenue['fare_amount'], label='Fare Amount')
plt.plot(monthly_revenue['pickup_month'].astype(str),
monthly_revenue['tip_amount'], label='Tip Amount')
```

```
plt.plot(monthly_revenue['pickup_month'].astype(str),
monthly_revenue['total_amount'], label='Total Amount')

plt.xlabel('Month')
plt.ylabel('Amount (Millions $)')
plt.title('Monthly Revenue Trends')
plt.xticks(rotation=45)

plt.legend()
plt.grid(True)
plt.show()
```



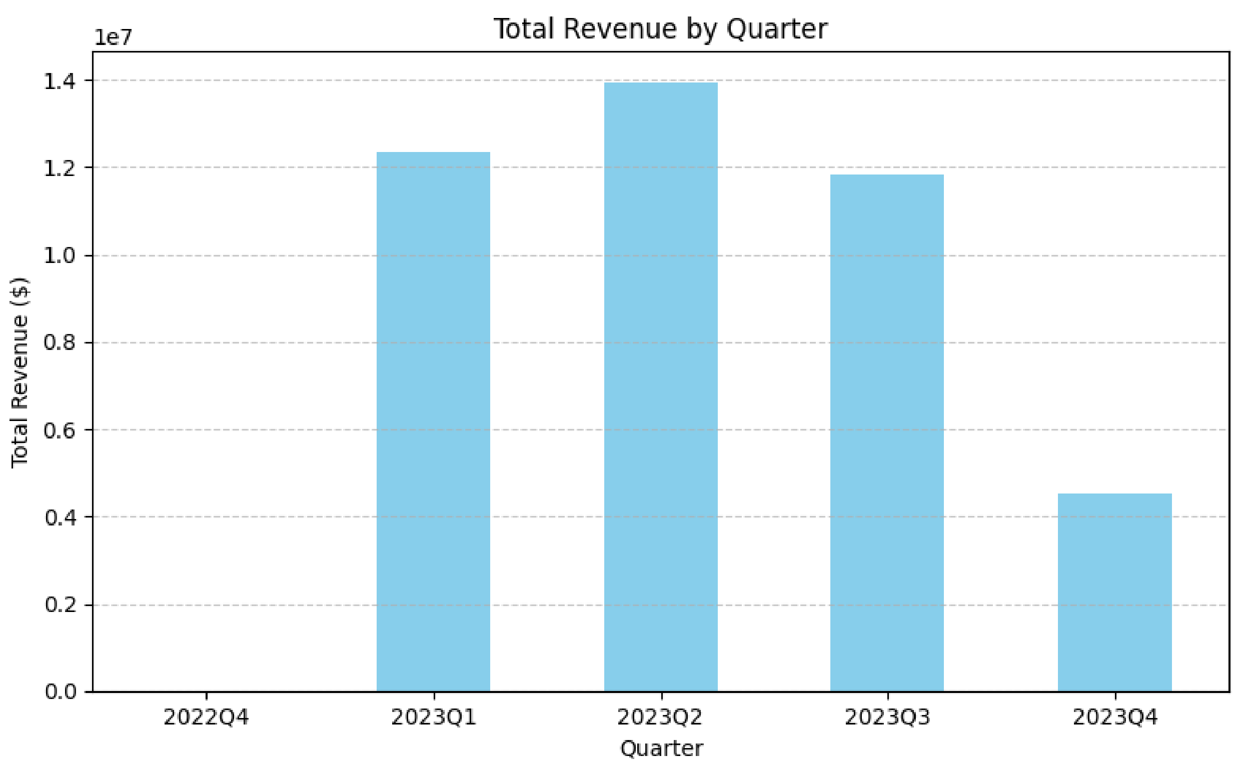
Let us now make a quarterly analysis and make a plot.

```
sampled_data['pickup_quarter'] =
sampled_data['tpep_pickup_datetime'].dt.to_period('Q')

quarterly_revenue = sampled_data.groupby('pickup_quarter')
['total_amount'].sum().reset_index()

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

```
print(quarterly_revenue)
```

	pickup_quarter	total_amount	proportion
0	2022Q4	13.50	0.00
1	2023Q1	12361305.28	0.29
2	2023Q2	13952909.01	0.33
3	2023Q3	11831790.65	0.28
4	2023Q4	4501257.12	0.11

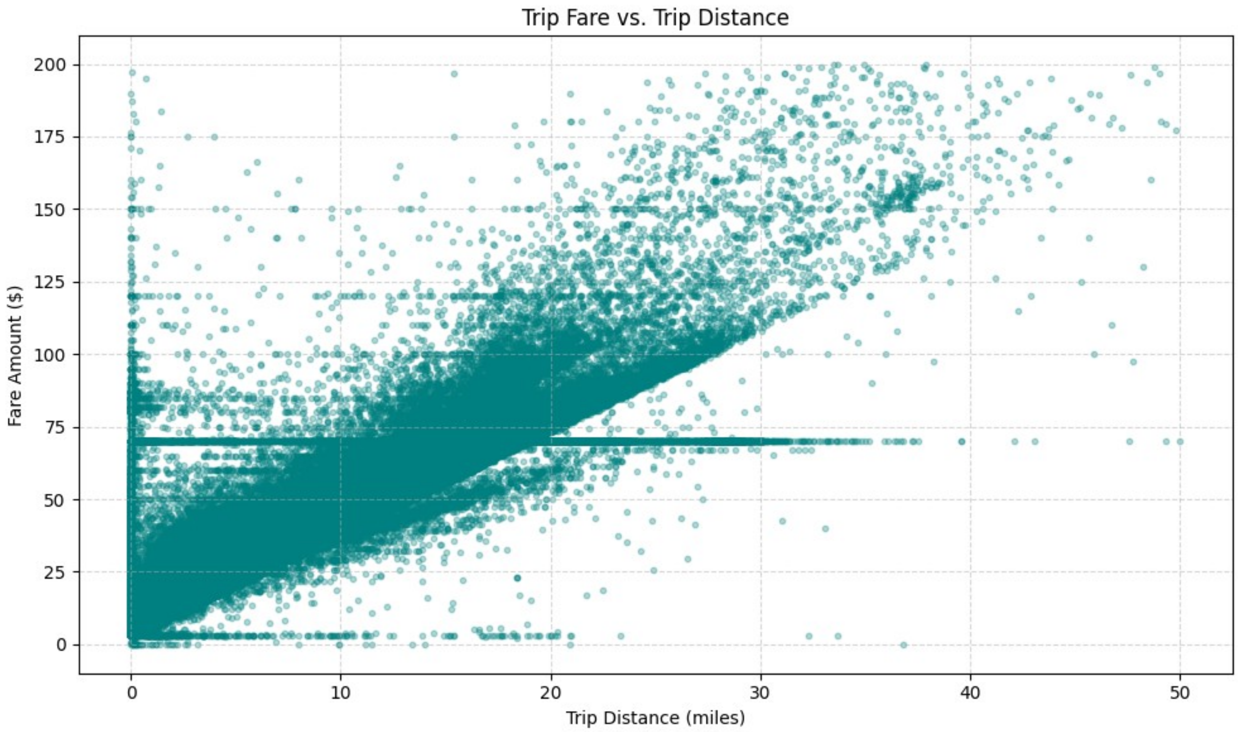
```
quarterly_sum = quarterly_revenue.groupby('pickup_quarter')
['total_amount'].sum()

plt.figure(figsize=(8,5))
quarterly_sum.plot(kind='bar', color='skyblue')

plt.title('Total Revenue by Quarter')
plt.xlabel('Quarter')
plt.ylabel('Total Revenue ($)')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()
```




```
monthly_revenue['pickup_month_str'] =
monthly_revenue['pickup_month'].astype(str)
```

```
month_labels = monthly_revenue['pickup_month_str'].tolist()
month_positions = np.arange(len(month_labels))

quarterly_sum = quarterly_revenue.groupby('pickup_quarter')
['total_amount'].sum()

quarter_positions = {
    'Q1': (0 + 2) / 2,
    'Q2': (3 + 5) / 2,
    'Q3': (6 + 8) / 2,
    'Q4': (9 + 11) / 2
}
quarter_labels = quarterly_sum.index.tolist()

quarter_pos_vals = [quarter_positions[f'Q{q.quarter}'] for q in
quarter_labels]

plt.figure(figsize=(14, 7))

plt.plot(month_positions, monthly_revenue['fare_amount'] / 1e6,
marker='o', label='Fare Amount (Millions)')
plt.plot(month_positions, monthly_revenue['tip_amount'] / 1e6,
marker='o', label='Tip Amount (Millions)')
plt.plot(month_positions, monthly_revenue['total_amount'] / 1e6,
marker='o', label='Total Amount (Millions)')

plt.bar(quarter_pos_vals, quarterly_sum / 1e6, width=2.5, alpha=0.3,
color='orange', label='Quarterly Total Revenue')

plt.xticks(month_positions, month_labels, rotation=45)
plt.xlabel('Month')
plt.ylabel('Revenue (Millions $)')
plt.title('Monthly Revenue (Fare, Tips, Total) with Quarterly Total
Revenue Overlay')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```


```
filtered_data = sampled_data[(sampled_data['trip_distance'] <= 50) &
(sampled_data['fare_amount'] <= 200)]

plt.figure(figsize=(10, 6))
plt.scatter(filtered_data['trip_distance'],
filtered_data['fare_amount'], alpha=0.3, s=10, color='teal')

plt.title('Trip Fare vs. Trip Distance')
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Fare Amount ($)')
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```



Let us now find the correlation between:

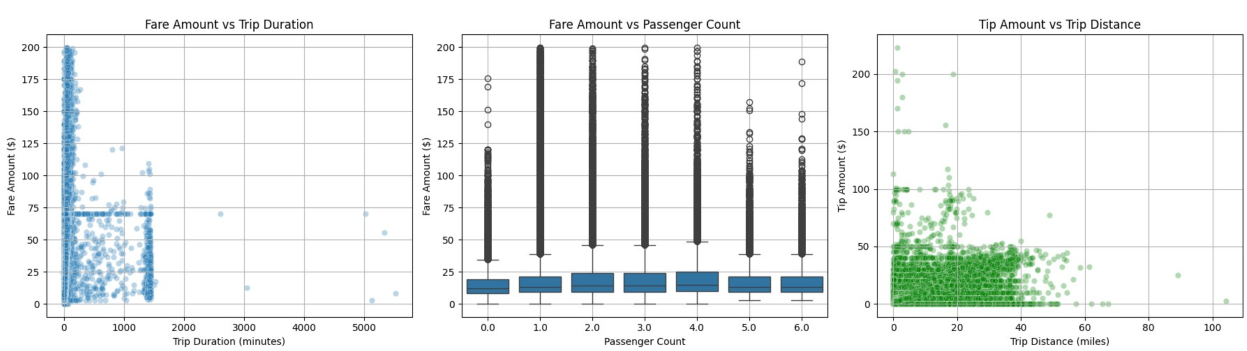
1. fare_amount and trip duration (pickup time to dropoff time)
2. fare_amount and passenger_count
3. tip_amount and trip_distance

And before that lets create a trip duration index

```
sampled_data['tpep_pickup_datetime'] =
pd.to_datetime(sampled_data['tpep_pickup_datetime'])
sampled_data['tpep_dropoff_datetime'] =
pd.to_datetime(sampled_data['tpep_dropoff_datetime'])

sampled_data['trip_duration'] = (sampled_data['tpep_dropoff_datetime']
- sampled_data['tpep_pickup_datetime']).dt.total_seconds() / 60

correlations = {
    'Fare vs Duration': sampled_data[['fare_amount',
'trip_duration']].corr().iloc[0, 1],
    'Fare vs Passenger Count': sampled_data[['fare_amount',
'passenger_count']].corr().iloc[0, 1],
    'Tip vs Distance': sampled_data[['tip_amount',
'trip_distance']].corr().iloc[0, 1]
}

for key, value in correlations.items():
    print(f"{key}: {value:.3f}")
```

```
Fare vs Duration: 0.273
Fare vs Passenger Count: 0.044
Tip vs Distance: 0.589
```

```
plt.figure(figsize=(18, 5))


plt.subplot(1, 3, 1)
sns.scatterplot(data=sampled_data, x='trip_duration', y='fare_amount',
alpha=0.3)
plt.title('Fare Amount vs Trip Duration')
plt.xlabel('Trip Duration (minutes)')
plt.ylabel('Fare Amount ($)')
plt.grid(True)


plt.subplot(1, 3, 2)
sns.boxplot(data=sampled_data, x='passenger_count', y='fare_amount')
plt.title('Fare Amount vs Passenger Count')
plt.xlabel('Passenger Count')
plt.ylabel('Fare Amount ($)')
plt.grid(True)


plt.subplot(1, 3, 3)
sns.scatterplot(data=sampled_data, x='trip_distance', y='tip_amount',
alpha=0.3, color='green')
plt.title('Tip Amount vs Trip Distance')
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Tip Amount ($)')
plt.grid(True)

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




```
payment_counts =
sampled_data['payment_type'].value_counts().sort_index()
```

```python
print("Payment Type Distribution:")
print(payment_counts)
```

```
Payment Type Distribution:
payment_type
1    1215014
2     257815
3       5732
4       9807
Name: count, dtype: int64
```

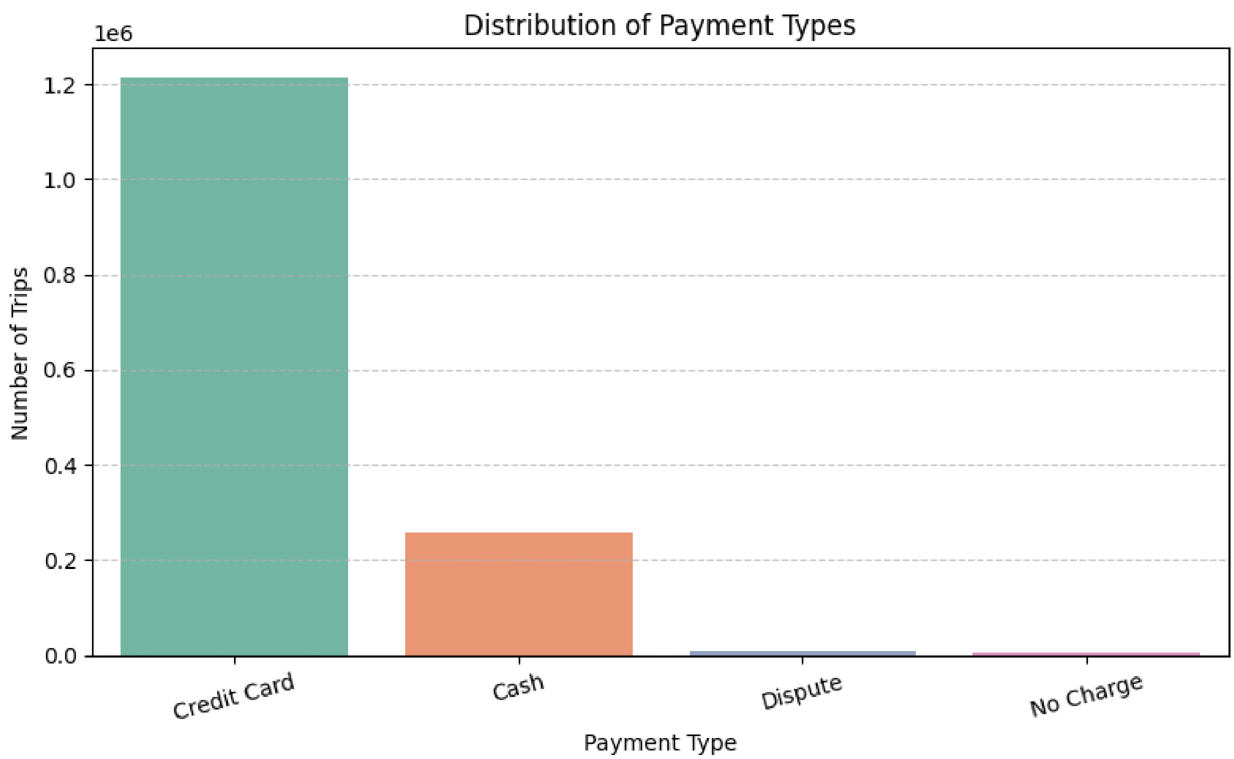
```python
payment_labels = {
    1: 'Credit Card',
    2: 'Cash',
    3: 'No Charge',
    4: 'Dispute'
}


sampled_data['payment_type_label'] =
sampled_data['payment_type'].map(payment_labels)

payment_counts = sampled_data['payment_type_label'].value_counts()

print("Payment Type Distribution:")
print(payment_counts)
```

```
Payment Type Distribution:
payment_type_label
Credit Card    1215014
Cash            257815
Dispute           9807
No Charge         5732
Name: count, dtype: int64
```

```python
plt.figure(figsize=(8, 5))
sns.countplot(data=sampled_data, x='payment_type_label',
              order=payment_counts.index, palette='Set2')

plt.title('Distribution of Payment Types')
plt.xlabel('Payment Type')
plt.ylabel('Number of Trips')
plt.xticks(rotation=15)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

```
<ipython-input-74-0ef7242ac56c>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
```

```
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(data=sampled_data, x='payment_type_label',
```



Let us now begin with Geo Analysis

```
import geopandas as gpd



!pip install geopandas

Requirement already satisfied: geopandas in
/usr/local/lib/python3.11/dist-packages (1.0.1)
Requirement already satisfied: numpy>=1.22 in
/usr/local/lib/python3.11/dist-packages (from geopandas) (2.0.2)
Requirement already satisfied: pyogrio>=0.7.2 in
/usr/local/lib/python3.11/dist-packages (from geopandas) (0.11.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.11/dist-packages (from geopandas) (24.2)
Requirement already satisfied: pandas>=1.4.0 in
/usr/local/lib/python3.11/dist-packages (from geopandas) (2.2.2)
Requirement already satisfied: pyproj>=3.3.0 in
/usr/local/lib/python3.11/dist-packages (from geopandas) (3.7.1)
Requirement already satisfied: shapely>=2.0.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from geopandas) (2.1.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.4.0-
>geopandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.4.0-
>geopandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.4.0-
>geopandas) (2025.2)
Requirement already satisfied: certifi in
/usr/local/lib/python3.11/dist-packages (from pyogrio>=0.7.2-
>geopandas) (2025.4.26)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2-
>pandas>=1.4.0->geopandas) (1.17.0)
```

```
zones = gpd.read_file("/content/drive/MyDrive/Datasets and
Dictionary/taxi_zones/taxi_zones.shp")
zones.head()
```

{"summary":"{\n \"name\": \"zones\",\n \"rows\": 263,\n \"fields\": [\n   {\n   \"column\": \"OBJECTID\",\n   \"properties\": {\n   \"dtype\": \"int32\",\n   \"num_unique_values\": 263,\n   \"samples\": [\n   116,\n   121,\n   260\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Shape_Leng\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.054593642765557934,\n   \"min\": 0.0143055167343,\n   \"max\": 0.43346966679,\n   \"num_unique_values\": 263,\n   \"samples\": [\n   0.0681164844265,\n   0.0969153373445,\n   0.133514154636\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"Shape_Area\",\n   \"properties\": {\n   \"dtype\": \"number\",\n   \"std\": 0.00048228767540393826,\n   \"min\": 6.33056361314e-06,\n   \"max\": 0.00486634037837,\n   \"num_unique_values\": 263,\n   \"samples\": [\n   0.000260415337217,\n   0.000384563286473,\n   0.000422345326907\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"zone\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 260,\n   \"samples\": [\n   \"Bronx Park\",\n   \"Pelham Parkway\",\n   \"Sunset Park East\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"LocationID\",\n   \"properties\": {\n   \"dtype\": \"int32\",\n   \"num_unique_values\": 260,\n   \"samples\": [\n   31,\n   185,\n   227\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\":

```
\"borough\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 6,\n        \"samples\":
[\n          \"EWR\",\n          \"Queens\",\n          \"Brooklyn\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"geometry\",\n
\"properties\": {\n        \"dtype\": \"geometry\",\n
\"num_unique_values\": 263,\n        \"samples\": [\n
\"POLYGON ((1001062.7085697055 241053.7687214315, 1000940.9143327624
240820.59030981362, 1000816.5282615274 240595.38718770444,
1000690.708065629 240367.5598036796, 1000565.3018446267
240141.23152861, 1000439.6060729623 239914.7984405905,
1000314.5016672015 239689.94618412852, 1000183.5141002685
239454.75960591435, 1000046.9102521539 239218.3725283742,
999885.9035241008 238987.70369449258, 999735.4967380017
238777.69075754285, 999582.3606063128 238562.77672584355,
999430.0119010061 238349.28326679766, 999174.2605487853
238490.10477472842, 998853.0479185432 238668.7145024836,
998619.2490667552 238797.56298334897, 998492.7859230638
238571.45447479188, 998110.2583885789 238784.2569360435,
997984.4111534953 238555.85720984638, 997859.5195685923
238330.7431896031, 997283.2379503846 238647.03841787577,
997099.8657316715 238748.57032687962, 997072.6215375662
238763.7275954038, 997045.4236918688 238778.70821593702,
996511.8387433589 239076.38163869083, 996502.3865499645
239081.76175142825, 996491.6944718659 239086.5160405934,
996333.6993343234 239154.93456672132, 996285.375727132
239165.03908087313, 996198.5837806612 239181.37343524396,
996128.168464303 239194.62581719458, 996109.0050586164
239199.06789776683, 996089.5856537074 239203.50076009333,
995456.5686577559 239337.94240055978, 995450.4661865234
239346.46258547902, 995448.5303954929 239350.16040037572,
995447.1885986179 239353.74279785156, 995446.6669922322
239355.84600830078, 995446.5292358398 239356.71881105006,
995449.764038071 239370.4487915337, 995443.606201157
239373.64459231496, 995592.8182373196 239668.96881102026,
995598.4382324219 239666.14379884303, 996078.6704101562
240621.25659181178, 996164.5690307766 240578.7601929009,
996338.9783935696 240922.65698242188, 996371.4487915188
240907.01080322266, 996370.5660400391 240905.4653930664,
996364.9790038913 240895.66241456568, 996692.9603881687
240729.45220945776, 996787.4486083835 240913.47399902344,
996830.4898071438 240892.76220703125, 996851.8281860948
240933.831603989, 996851.8612060398 240933.89080809057,
996880.5881958157 240986.2066040039, 996905.0498047024
241029.4515991062, 996920.4436035007 241055.301208511,
996933.4487915486 241074.76300050318, 996942.5794067234
241086.62341308594, 996963.4255982041 241111.67260742188,
996971.280029282 241121.98419189453, 996974.5018310845
241126.56079101562, 996984.157409668 241141.33459475636,
```

```
996993.0288086534 241156.58380128443, 997093.3272094429
241343.30920410156, 997263.3562011868 241704.99780274928,
997437.854003936 242043.7998046875, 997478.6018066555
242116.90661619604, 997491.7875976712 242136.6835937649,
997616.432800293 242370.39379882812, 997644.2514038235
242418.16821289062, 997698.2305908501 242518.40838624537,
997722.2059936672 242565.19598388672, 997756.1832275391
242636.85180665553, 997783.0704345554 242694.41821287572,
997793.2109985948 242728.14880371094, 997795.2124023587
242737.1766357422, 997810.3859863728 242805.6221923977,
997818.9816284031 242844.4017944187, 997827.3707885891
242872.5678100586, 997902.577392593 242997.21319578588,
997969.0932006985 243147.02600097656, 998018.5317993164
243227.32739257812, 998043.1414586306 243274.46646507084,
998092.0184808522 243287.94003783166, 998125.9527703226
243297.29484848678, 998148.0200701207 243296.09716293216,
998166.8617767245 243293.67602309585, 998185.3720495701
243288.23617462814, 998203.8823324591 243282.7934256196,
998210.1283718795 243279.64696355164, 998247.7865017056
243260.67724588513, 998309.0333628953 243206.81594325602,
998389.052574262 243153.90244962275, 998559.1019114256
243079.63448256254, 999213.9297980666 242695.76689320803,
999242.0626745969 242680.8358937353, 999271.022579357
242664.8055060953, 1000029.4429909587 242245.04906240106,
1000475.6101781279 241998.6357729286, 1000758.4215184897
241841.07030822337, 1000807.0167533159 241814.34254337847,
1000957.9516826719 241731.17545683682, 1001131.8851139992
241634.0238532126, 1001230.7659743577 241579.45004203916,
1001142.3771669865 241305.20881572366, 1001062.7085697055
241053.7687214315))\",\n          \"POLYGON ((1039920.4318903834
208288.84205247462, 1040167.475295186 208277.93010389805,
1040424.6300606877 208282.45788386464, 1040547.8091351688
208301.00067821145, 1040671.3020197451 208318.34435164928,
1040674.7181050926 208300.58087652922, 1040682.8251272589
208260.7318995893, 1040761.7979261428 208268.25509795547,
1040643.4615314305 208216.80501885712, 1040641.6604443192
208182.11673790216, 1040825.0459897667 208243.03831408918,
1040797.4181161076 208147.89734619856, 1041015.8823509067
207637.49823243916, 1041177.2729392648 207142.69294068217,
1041175.0846890509 206957.60474674404, 1041167.5089961588
206612.14267110825, 1041176.311698541 206544.13116331398,
1041190.579980433 206476.71650385857, 1041210.232528016
206410.55540265143, 1041235.0777765214 206346.2802682072,
1041264.8283353448 206284.4923182428, 1041348.4766645879
206139.2139286399, 1041362.5908596963 206030.98286873102,
1041367.7252630442 205928.28275234997, 1041392.7350696474
205906.34030804038, 1041384.6443556696 205756.7246130556,
1041401.8654666692 205557.98510748148, 1041475.6517071426
205322.30262501538, 1041558.7551116645 205065.3085409552,
```

```
1041667.1963874996 204730.7850421667, 1041817.799834773
204261.776632905, 1041888.8790836036 204047.1186941713,
1041926.0477378219 203922.27313274145, 1041968.0553676486
203788.11296509206, 1041972.8027120382 203769.6154562086,
1041978.176729694 203745.1833769232, 1041997.687852323
203673.2935140878, 1042020.9291393459 203602.41482113302,
1042047.812618196 203532.83783610165, 1042078.2193099111
203464.83590087295, 1042213.6585756987 203187.26137815416,
1042333.1992835402 202956.44147072732, 1042359.8591346443
202756.4025222212, 1042378.9009697437 202162.89650779963,
1042376.7718719393 201986.30561880767, 1042356.9621760398
201699.2591368556, 1042320.8011466712 201590.8834000826,
1042314.1668102741 201570.99797531962, 1042265.7352719754
201394.7977798134, 1042259.0946888626 201372.9602675736,
1042069.151097551 201370.15875725448, 1042030.5684686005
201369.58745615184, 1041971.559187606 201361.2904535234,
1041876.9174588472 201346.58602772653, 1041770.3719701618
201320.30992093682, 1041737.2193593085 201312.13377441466,
1041605.710290432 201271.10034239292, 1041587.8430627137
201265.28057403862, 1041355.6661450565 201165.47529032826,
1041232.709089905 201112.61994086206, 1041130.0879382193
201066.58321806788, 1040900.3179394007 200957.11165966094,
1040650.5157145113 200851.08823023736, 1040409.4300026745
200758.04717484117, 1040237.7932814211 200691.89554437995,
1039934.3162062019 200575.31115351617, 1039804.231004715
200525.71805033088, 1039679.1460093558 200482.75586940348,
1039632.1153491884 200466.60048733652, 1039435.2360238433
200417.35880434513, 1039170.4199023396 200345.66934594512,
1038837.4680216014 200276.18783582747, 1038822.3137684017
200273.87488123775, 1038806.0080091506 200269.6914217621,
1038598.1771828979 200232.9332382977, 1038405.1885367483
200207.5399094075, 1038322.9644478858 200193.45160362124,
1038087.1283282787 200173.40599133074, 1037833.6904215962
200151.92952749133, 1037528.0872219652 200139.83749878407,
1037524.426055789 200160.80402345955, 1037518.3793324083
200192.9495602697, 1037447.1359142512 200485.1343512684,
1037408.8849358708 200643.89689867198, 1037303.265109688
201083.5844886303, 1037187.5829052478 201599.6597314179,
1037183.5998924524 201617.4351644665, 1037448.2746437341
201732.10657462478, 1037510.1772509068 201760.54209542274,
1037687.6636919975 201842.06396064162, 1037924.77295506
201950.10192763805, 1038083.9817211479 202022.7753084451,
1038163.2765175551 202058.68452559412, 1038369.1386627257
202156.39390213788, 1038413.3941127062 202177.39829692245,
1038433.4760879129 202185.24890771508, 1038431.4582955539
202205.2979492098, 1038413.6040894389 202454.68724410236,
1038393.3841305077 202711.05808869004, 1038372.8658757359
202975.34560254216, 1038352.9250969887 203238.65912929177,
1038332.7773957998 203491.89955370128, 1038304.9854853898
```

```
203487.86790397763, 1037877.840593189 203426.57532152534,
1037841.1510379463 203674.2352269441, 1037803.5723438263
203921.58757339418, 1037768.0374606401 204169.42505811155,
1037731.7910430729 204418.48921188712, 1037695.2295131832
204663.46895051003, 1037447.5757415444 204626.80881854892,
1037200.0696722418 204591.09369947016, 1036951.0183701515
204554.17144826055, 1036601.4719642252 204496.14660975337,
1036631.6717116088 204359.05353045464, 1036666.176338777
204261.45243608952, 1036688.6797593832 204215.45822146535,
1036688.7059272975 204215.40544985235, 1036688.7254564911
204215.34647026658, 1036705.713613376 204166.50804457068,
1036705.7303497493 204166.4588959366, 1036705.7398726493
204166.41301065683, 1036716.747055456 204115.64528645575,
1036716.7534799576 204115.6223474145, 1036716.7571398467
204115.59612344205, 1036721.5338580608 204064.05421429873,
1036721.5372557938 204064.0210674256, 1036721.5373258144
204063.988642022, 1036720.1099769771 204012.7779842764,
1036706.271373868 203931.18365056813, 1036492.0231681168
204073.86077198386, 1036227.8046370149 204246.3699068129,
1036135.2127964199 204305.41185848415, 1035818.2546892315
204507.524071306, 1035680.5748886764 204595.3194695711,
1035637.0381423533 204616.93017084897, 1035636.9657264352
204616.96608641744, 1035636.903550908 204617.00894598663,
1035595.7160933167 204644.6634953618, 1035595.6503155828
204644.70598310232, 1035595.5911689252 204644.7583218217,
1035557.9109734595 204678.0817067176, 1035557.8748730421
204678.11442029476, 1035557.842083931 204678.1540631801,
1035524.787709251 204716.39847017825, 1035524.7515866309
204716.44174937904, 1035524.7251662165 204716.48432043195,
1035497.2154236585 204758.5903712958, 1035497.1695397794
204758.6624121815, 1035497.1333426684 204758.74103146791,
1035475.7358905971 204803.50469526649, 1035475.7097190768
204803.5607471913, 1035475.6901863366 204803.62300673127,
1035460.6112381369 204849.84641464055, 1035483.5508294106
205174.89823381603, 1035511.0703890175 205434.67767992616,
1035517.5764274448 205459.04154871404, 1035517.9011718482
205469.3171263337, 1035529.6686570942 205711.1203749925,
1035530.6511912048 205731.31227596104, 1035532.6380425245
205772.1402453184, 1035533.6906891018 205793.77614298463,
1035538.4670726508 205891.92272222042, 1035539.5515861064
205914.2119359225, 1035542.2651370615 205969.9725022167,
1035543.3793966919 205992.87094286084, 1035544.2495472282
206010.75602258742, 1035545.4142557085 206031.81250719726,
1035546.2277393937 206046.74018378556, 1035550.2220661938
206130.77891145647, 1035551.6936770529 206161.74988728762,
1035553.6325640231 206202.5515318662, 1035555.1710473597
206234.9227784425, 1035556.7097819448 206267.30422738194,
1035558.2514337897 206299.74579778314, 1035559.7232996076
206330.7171409577, 1035561.2614895403 206363.08802463114,
```

```
1035562.6033382416 206391.31931610405, 1035564.2052004486
206425.03107763827, 1035566.2106473744 206467.2326322794,
1035567.8191311508 206501.08467690647, 1035569.1601548046
206529.30540290475, 1035571.0260224044 206568.5668682456,
1035572.5647127628 206600.94832228124, 1035575.9276313186
206673.662292704, 1035582.5833228081 206836.76822070777,
1035587.0973277837 206947.39449512959, 1035588.8963089287
207002.2660059482, 1035591.385745272 207078.17989449203,
1035596.5066658854 207234.3657578528, 1035604.7120497078
207457.3413604796, 1035603.160168113 207891.07062344253,
1035562.192474708 208172.9532136172, 1035524.9810830057
208408.3433574587, 1035506.283731848 208529.60908704996,
1035503.0978923589 208547.62442953885, 1035517.6353178471
208543.81214904785, 1035546.077018246 208537.73578219116,
1035773.7372996658 208489.09133924544, 1036009.294596374
208460.9380723387, 1036271.656255871 208447.72604177892,
1036515.572985068 208435.4458860606, 1036790.765925169
208421.59125821292, 1037065.9589239657 208407.7328272909,
1037199.3379331976 208401.61405472457, 1037332.7170120329
208395.49565070868, 1037442.559264943 208390.4558236003,
1037589.9375662357 208383.69440439343, 1037851.0065024048
208372.01440873742, 1038109.6739447564 208362.18233670294,
1038368.134788841 208353.7566974908, 1038381.2940824926
208353.43482156098, 1038395.1229823679 208352.95304699242,
1038666.5958792865 208342.3656013608, 1038929.7973152846
208332.0999301374, 1039173.1265984625 208322.30330112576,
1039308.0409317166 208315.8858962655, 1039427.2271265835
208310.6237602681, 1039673.4995826632 208299.74766007066,
1039920.4318903834 208288.84205247462))\",\n          \"POLYGON
((1011466.966050446 216463.0052037984, 1011545.8893444091
216046.87074047327, 1011571.9624619037 216050.17470617592,
1011600.1937469691 216053.6461749524, 1012225.1987177283
216141.70162980258, 1012266.9881703705 216146.48398335278,
1012293.2110627741 216150.65673843026, 1012323.9949972034
216152.67430335283, 1012213.611178264 215936.497042194,
1012116.9709669501 215747.23259560764, 1012096.4002358019
215701.31360079348, 1012073.5623051226 215651.15337578952,
1012062.5519055575 215618.12199233472, 1012051.134619981
215590.79560130835, 1012043.1818924844 215560.61863543093,
1012032.3749582022 215511.6818191707, 1012020.1403993517
215447.246476233, 1012012.3911368698 215415.4386368245,
1012006.8662881106 215373.6338646561, 1011996.820307225
215292.88621532917, 1011992.8834942877 215211.0197675377,
1011995.1994477957 215128.73638747633, 1012003.8020511717
215046.74510735273, 1012011.2066633105 215006.25944180787,
1012018.6115803868 214965.77086479962, 1012030.330918476
214921.20087239146, 1012039.4516560882 214886.50570131838,
1012066.0362293124 214809.59963840246, 1012166.6923317611
214552.70723341405, 1012198.2176172286 214421.31995545328,
```

```
1012217.0007415563 214341.382219553, 1012240.3600262254
214255.9658421278, 1012355.0382594019 214045.43793426454,
1012436.3962585032 213873.68312707543, 1012537.1704796255
213663.32577347755, 1012591.3206740469 213534.83199845254,
1012614.8144516349 213457.73934157193, 1012627.1113297194
213407.02076724172, 1012652.8329382688 213272.39827032387,
1012693.8466600329 213055.7485857308, 1012707.8524001837
212856.55271561444, 1012729.7159414589 212708.51144693792,
1012767.2520690709 212581.95034877956, 1012932.1958949268
212200.4547379315, 1012998.9463357776 212072.40200960636,
1013036.4181043506 211992.7680207789, 1013113.4421155751
211815.35384459794, 1013159.7812040299 211699.4992044121,
1013245.2927162945 211485.7075202316, 1013337.9463407099
211254.05931323767, 1013354.3734343946 211217.5925836861,
1013370.7972556949 211181.12294714153, 1013397.6149125695
211104.81732326746, 1013417.9297914356 211025.90331216156,
1013431.4008263052 210945.22756025195, 1013437.840941608
210863.68936777115, 1013437.2143308669 210782.20316003263,
1013429.648682639 210701.67482112348, 1013415.4133439511
210622.9612261355, 1013403.3397380561 210566.45883591473,
1013388.0149021596 210510.6517033279, 1013369.5013769269
210455.8226313889, 1013347.9038492292 210402.2333432585,
1013276.5686623901 210287.57166153193, 1013232.7396745831
210228.39182434976, 1013185.292404443 210171.97813601792,
1013134.4982894361 210118.6577464193, 1012963.009432599
209932.04412442446, 1012772.7078897357 209727.64806483686,
1012734.374771893 209687.313754946, 1012700.7132362276
209651.72639876604, 1012619.7710956931 209566.15790753067,
1012464.2175755799 209401.71386668086, 1012326.7016204298
209251.6092105806, 1012284.401790306 209207.70823018253,
1012267.5251632631 209191.3896251768, 1012246.5573971719
209169.16850708425, 1012213.4635763317 209134.09619824588,
1012192.9943623245 209112.72823815048, 1012180.2680896819
209093.37157195807, 1012031.0199518651 208942.63028723001,
1011801.4073882103 208709.44827862084, 1011678.2360977083
208626.88671652973, 1011478.8194533288 208493.2183752358,
1011218.6495204568 208371.64368605614, 1011085.496837318
208307.07678849995, 1010971.3893319815 208253.228727445,
1010951.3729505986 208243.7832594663, 1010938.5975329578
208237.06076861918, 1010823.5602698922 208176.54984650016,
1010678.4634354413 208100.22715234756, 1010614.9990222752
208063.52421486378, 1010419.8385564238 207950.65691438317,
1010304.1303122491 207879.21414366364, 1010171.6534670144
207808.87907211483, 1010097.7954617292 207767.81299635768,
1009913.3173972517 207693.23311769962, 1009851.3750863522
207671.66485485435, 1009775.5256806463 207652.30782577395,
1009495.8607197404 207594.9329726845, 1009396.7108810097
207581.1563296169, 1009389.6995393932 207555.96609579027,
1009384.1485417932 207538.9190967232, 1009377.5474843532
```

```
207518.93231263757, 1009303.5616716146 207309.0051652789,
1009216.9276389033 207068.77908980846, 1009126.2519551963
206828.91062238812, 1009001.8560215384 206512.49411414564,
1010221.6205336303 206123.81376847625, 1010305.9509488195
206439.02614158392, 1010388.0276956558 206675.98127140105,
1010296.0232024342 206710.46255649626, 1010374.0448441654
206937.41459660232, 1010530.5338848829 207384.3295428902,
1010659.8905821592 207318.98494651914, 1010700.894490689
207291.8395151645, 1010914.0305557549 207150.7276544124,
1011030.1261712611 207076.28914134204, 1011039.3421776891
207070.38001306355, 1011169.2073383182 206987.1124857664,
1011285.788481772 206916.33508367836, 1011425.2466900647
206831.669650957, 1011664.7410289794 206676.95552659035,
1011245.1093099564 206016.97008895874, 1010796.4125478268
205295.61016458273, 1010530.1828550845 204869.3245945722,
1010377.0797241628 204706.7335179299, 1010101.1418649703
204422.96106037498, 1010045.7090166062 204365.82851320505,
1010007.4542109072 204325.8910740465, 1009992.5427261591
204310.64803376794, 1009375.9658705592 204812.2087597102,
1009246.1776564121 204910.47667306662, 1009145.4547724277
204984.14080543816, 1009041.6909889281 205053.43792644143,
1008927.5253422409 205116.90945973992, 1008781.594032228
205186.3320981413, 1008709.7865285724 205215.52492883801,
1008703.487163052 205193.93668577075, 1008669.301137954
205087.41477812827, 1008637.309351638 204966.5521310568,
1008626.3581196815 204703.3113913983, 1007794.2927731276
204737.03503493965, 1007788.928684637 204507.36997002363,
1006986.4598567635 204699.6829353422, 1007051.5580635816
205202.0012947321, 1007085.5442598462 205454.38262043893,
1007109.185700804 205611.33114343882, 1007151.9419404566
205729.73287782073, 1007158.5460102111 205749.86769095063,
1007097.3947172165 205772.8299817145, 1006750.7218884975
205944.29284843802, 1006688.3598676026 205976.45806063712,
1006692.8908165693 206006.29079325497, 1006756.3650970459
205963.21671828628, 1006878.41551058 206394.37402272224,
1007087.9360705763 207154.88756607473, 1007097.5684948117
207173.95918868482, 1007109.7994364947 207195.54715016484,
1007280.4980133921 207229.85474288464, 1007287.4696949869
207265.07437442243, 1007385.9600763619 207708.19060973823,
1007202.410927102 207759.21371921897, 1007226.3344593048
207849.15778823197, 1007386.1211359501 208505.13441208005,
1007444.5002905726 208550.49842718244, 1007617.4823117703
209175.78894464672, 1007697.9477362633 209155.65121780336,
1007961.1526966691 209083.62708452344, 1008128.6883334666
209769.6953909546, 1008138.0551704317 209806.43073017895,
1008388.3041139841 209794.93150249124, 1008468.5532875806
209780.43669967353, 1008651.4629226774 209759.01608906686,
1008654.2157320082 209776.60479806364, 1008661.1087773591
209820.6208499521, 1008770.5671199262 210519.77255567908,
```

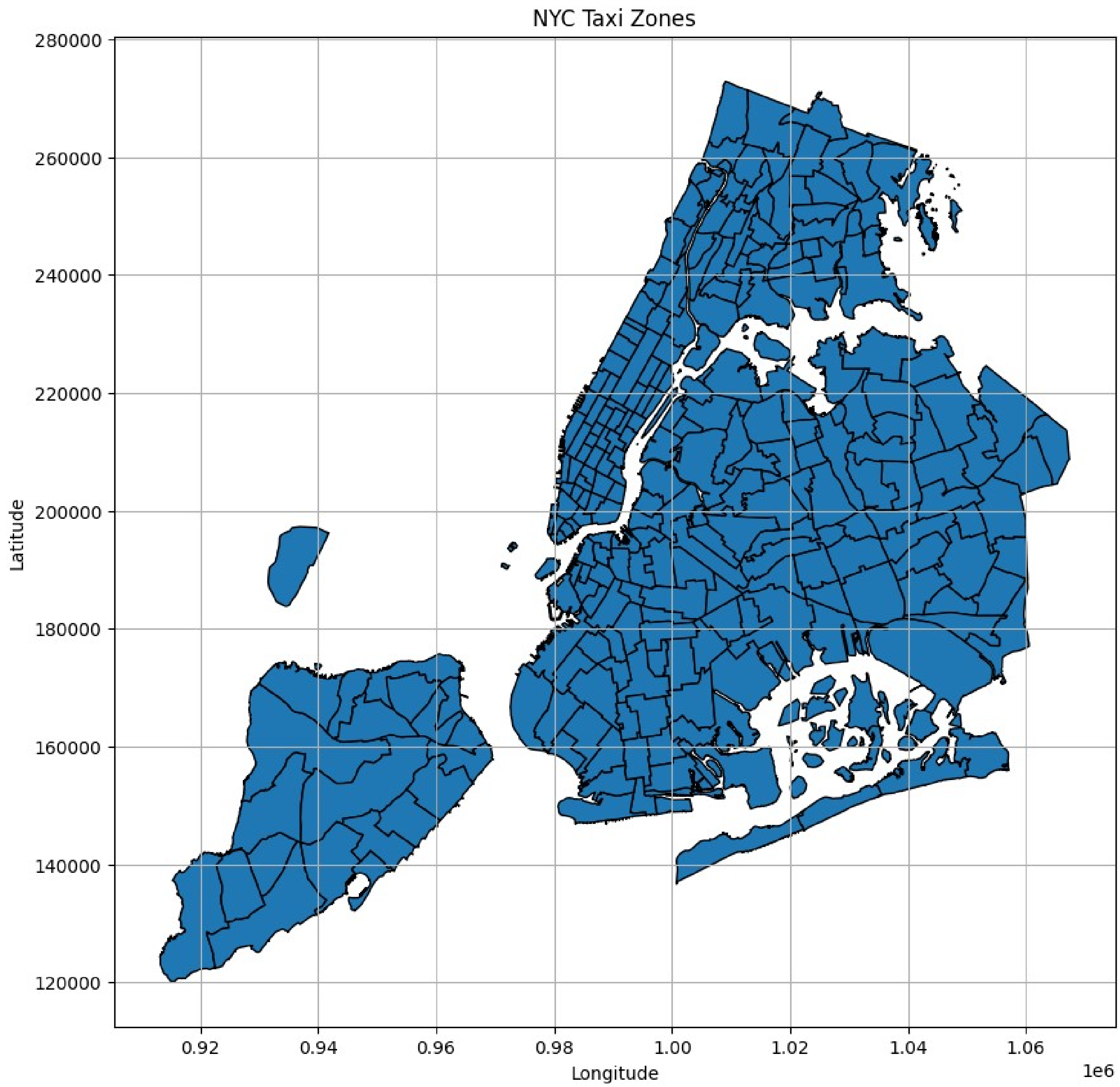
```
1008842.5387111753 210998.3184068054, 1008597.6803102046
211035.47025170922, 1008509.6163477004 211047.68156418204,
1008535.79818362 211192.63690763712, 1008544.2813374251
211239.60417541862, 1008600.0687080175 211548.49103312194,
1008652.4636996388 211801.1938779056, 1008658.0738986582
211843.7695619166, 1008658.0835337788 211843.83187243342,
1008658.0970484614 211843.89382249117, 1008667.6833020747
211885.94777671993, 1008681.131474182 211926.8889439255,
1008681.1610260159 211926.9840644151, 1008681.2005559653
211927.07591594756, 1008682.663754195 211930.43944740295,
1008698.319928065 211966.4235561639, 1008732.076136902
212054.98268590868, 1009143.167988196 211988.8708308339,
1009214.7098809332 212202.1905825287, 1009266.5864791274
212445.41181328893, 1009320.4282642007 212695.74588517845,
1009329.6799471229 212739.1842341125, 1009338.1872341484
212773.65632508695, 1009338.190539062 212773.67563813925,
1009340.5100879073 212809.9385869652, 1009340.5133754462
212809.974659279, 1009340.5102900416 212810.01108933985,
1009336.2285399586 212846.38234390318, 1009336.2254654616
212846.4082083404, 1009336.2187886834 212846.43443337083,
1009325.5431578755 212881.16869463027, 1009325.5364875048
212881.18872602284, 1009325.5264965296 212881.20474632084,
1009309.2504749894 212912.75005695224, 1009229.3883112818
213131.76525959373, 1009087.1871741414 213536.47581781447,
1008998.4866250753 213625.70825375617, 1009121.3703354746
213619.07404634356, 1009197.984121114 213707.06612937152,
1009270.2707015276 213798.9195651859, 1009337.9508790374
213894.30649197102, 1009398.6621343344 213989.5650664717,
1009400.7723668069 213992.87575775385, 1009413.7388291955
214015.63958486915, 1009458.5183759034 214094.25365906954,
1009519.1574899256 214242.16327349842, 1009526.0607494563
214259.00226673484, 1009534.8304748833 214280.39226421714,
1009541.114688158 214295.7208224982, 1009550.2447464317
214317.99070046842, 1009556.8655341864 214334.1400911361,
1009565.7009744197 214355.69229702652, 1009571.8046930283
214370.58056160808, 1009586.5437329262 214406.53116981685,
1009591.536508292 214418.70983071625, 1009597.681094408
214433.69797202945, 1009609.7223617285 214463.06884399056,
1009626.2780839354 214521.63951444626, 1009639.7579446733
214569.3294621706, 1009653.3558921814 214617.43706724048,
1009662.1862625778 214648.67727690935, 1009671.3696473688
214681.16606390476, 1009681.2709474862 214716.19537055492,
1009690.7540431023 214749.74468816817, 1009700.6552866846
214784.7743680179, 1009710.5567790419 214819.80368834734,
1009721.410515815 214858.20336198807, 1009731.5496022999
214894.07309266925, 1009741.631797865 214929.74238540232,
1009752.3049212545 214967.50175224245, 1009785.6241279691
215085.38022491336, 1009896.5787454098 215486.5504193157,
1009998.397337392 215697.2761490047, 1010154.420878604
```

```
216020.18467529118, 1010203.7170425355 216122.20903506875,
1010014.5637255907 216230.9849230349, 1009793.7274937928
216367.30673770607, 1009866.6343706697 216486.47602652013,
1009897.2479246408 216536.5123371184, 1010002.3266281039
216708.2674613148, 1010074.6916491687 216826.54784607887,
1010148.7400942445 216947.58096428216, 1010218.815560773
217062.6892861724, 1010271.026891008 217148.45020391047,
1010383.674078986 217333.48597851396, 1010513.0570248961
217546.01476244628, 1010701.252334848 217851.5751977265,
1010923.7285969704 218219.09760442376, 1010950.5824793875
218266.2824292183, 1010962.0356397629 218286.41023896635,
1011033.9449728876 218062.92004556954, 1011122.4491540641
217787.4690375626, 1011184.7485518008 217593.57367776334,
1011328.0584866107 217143.55184607208, 1011466.966050446
216463.0052037984))\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"zones"}
```


```python
print(zones.info())
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 263 entries, 0 to 262
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   OBJECTID    263 non-null    int32
 1   Shape_Leng  263 non-null    float64
 2   Shape_Area  263 non-null    float64
 3   zone        263 non-null    object
 4   LocationID  263 non-null    int32
 5   borough     263 non-null    object
 6   geometry    263 non-null    geometry
dtypes: float64(2), geometry(1), int32(2), object(2)
memory usage: 12.5+ KB
None
```

```python
zones.plot(figsize=(12, 10), edgecolor='black')
plt.title("NYC Taxi Zones")
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.grid(True)
plt.show()
```




```
print(zones.columns)

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

```
zones = zones.rename(columns={'LocationID': 'PULocationID'})
```

```
merged_gdf = sampled_data.merge(zones, on='PULocationID', how='left')

import geopandas as gpd
merged_gdf = gpd.GeoDataFrame(merged_gdf, geometry='geometry')
```

Grouoing Data by Location and Calculating Number of Trips

```
trip_counts =
sampled_data.groupby('PULocationID').size().reset_index(name='trip_cou
nt')
print(trip_counts.head())
```

```
   PULocationID  trip_count
0             1          54
1             2           1
2             3          31
3             4        1499
4             5          12
```

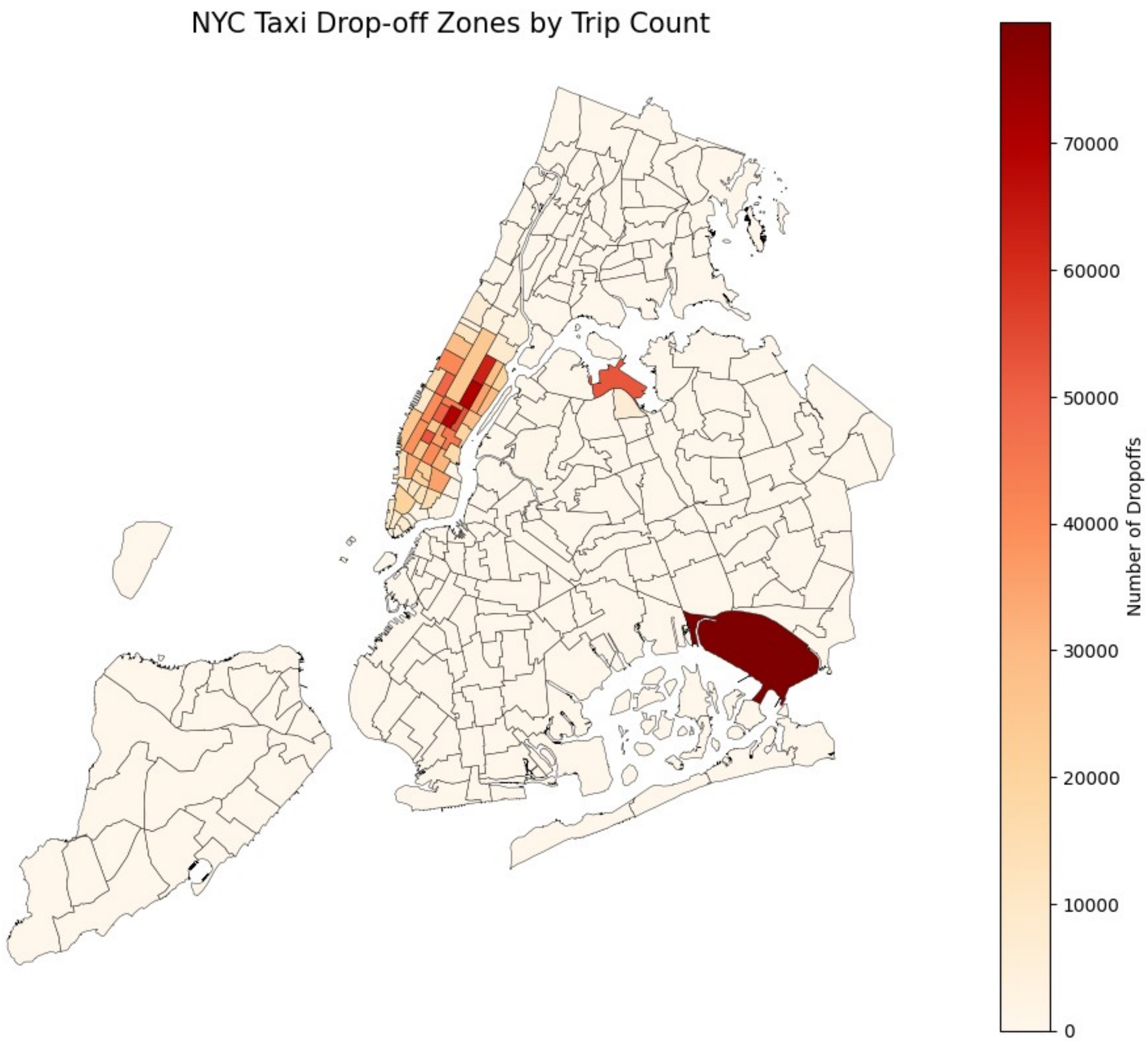
We will now merge trip counts back to the zones in Geo Data Frame. For this, we will first ensure column names match for mergin, then merge the data and finally ensure that NaNs are filled with 0

```
zones_with_trips = zones.merge(trip_counts, on='PULocationID',
how='left')
zones_with_trips['trip_count'] =
zones_with_trips['trip_count'].fillna(0)
```

Now we will create a Choropleth Map

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

zones_with_trips.plot(
    column='trip_count',
    ax=ax,
    legend=True,
    cmap='OrRd',
    legend_kwds={
        'label': "Number of Dropoffs",
        'orientation': "vertical"
    },
    edgecolor='black',
    linewidth=0.3
)
```

```
ax.set_title('NYC Taxi Drop-off Zones by Trip Count', fontsize=15)

ax.set_axis_off()

plt.show()
```



Lets now try displaying the zones DF sorted by number of trips of the Top 10 Zones.

```
zones_with_trips_sorted =
zones_with_trips.sort_values(by='trip_count', ascending=False)

print(zones_with_trips_sorted[['PULocationID', 'zone',
'trip_count']].head(10))
```

```
     PULocationID                            zone  trip_count
131           132                     JFK Airport    79607.00
160           161                  Midtown Center    69882.00
236           237           Upper East Side South    69841.00
235           236           Upper East Side North    62336.00
161           162                    Midtown East    53240.00
137           138               LaGuardia Airport    52467.00
185           186  Penn Station/Madison Sq West    52052.00
229           230       Times Sq/Theatre District    49877.00
141           142             Lincoln Square East    49201.00
169           170                     Murray Hill    44583.00
```

General Trend Analysis:

I Temporal Trends

1. Busiest Hours: Most trips occur during rush hours, particularly 8 AM–10 AM and 5 PM–7 PM, suggesting strong commuter patterns.

2. Busiest Days: Fridays and Saturdays consistently show higher trip counts, possibly due to both work and leisure travel.

3. Busiest Months: June, July, and December show peak activity—likely due to tourism and holiday travel.

II Revenue Trends

1. Daily Revenue: Revenue closely follows the trip count trends—more trips = higher daily earnings.

III Quarterly Revenue:

1. Q2 and Q4 are the strongest quarters in terms of total revenue.

2. Q1 tends to be slower, possibly due to harsh winter weather reducing demand.

IV Fare Relationships

1. Fare vs. Trip Distance: Strong positive correlation—longer distances lead to higher fares, as expected.

2. Fare vs. Trip Duration: Also positively correlated, although with greater variance due to traffic and waiting time.

3. Fare vs. Passenger Count: Minimal direct correlation—fare is calculated primarily on time and distance, not passengers.

IV Tip Behavior

1. Tip Amount vs. Trip Distance: Slight positive correlation—longer trips tend to have higher tips, though many short trips also include tips.

V Geographical Patterns

1. Busiest Zones (Pickup/Dropoff):

i. Manhattan (especially Midtown, Upper East Side, and Downtown) dominates in trip density.

ii. Airports (JFK, LaGuardia) also show high volumes, especially as destination zones.

These zones are key transport and business hubs, explaining their high activity.

EDA Insights and Strategies

Operational Efficiency

We will first try and find the routes that have the slowest speed at different tiems during a day.

To do this we will follow the following course of action:

i. Ensure datetime conversion

ii. Compute Trip Duration

iii. Filter out Zero or Negative Durations

iv. Calculate Avg Speed in Miles/Hour

v. Extract Pick-up Hour

vi. Define Each Route as Pick-Up to Drop

vii. Group by Hour and Route to Compute Mean Speed

viii. Find Slowest Route per hour

ix. Sort by Hour and finally

x. Display the Results

```
sampled_data['tpep_pickup_datetime'] =
pd.to_datetime(sampled_data['tpep_pickup_datetime'])
sampled_data['tpep_dropoff_datetime'] =
pd.to_datetime(sampled_data['tpep_dropoff_datetime'])

sampled_data['trip_duration_hours'] = (
    (sampled_data['tpep_dropoff_datetime'] -
sampled_data['tpep_pickup_datetime']).dt.total_seconds() / 3600
)


sampled_data = sampled_data[sampled_data['trip_duration_hours'] > 0]


sampled_data['avg_speed'] = sampled_data['trip_distance'] /
sampled_data['trip_duration_hours']
```

```python
sampled_data['hour'] = sampled_data['tpep_pickup_datetime'].dt.hour


sampled_data['route'] = sampled_data['PULocationID'].astype(str) + ' →
' + sampled_data['DOLocationID'].astype(str)


route_speeds = (
    sampled_data.groupby(['hour', 'route'])['avg_speed']
    .mean()
    .reset_index()
)


slowest_routes_by_hour = route_speeds.loc[
    route_speeds.groupby('hour')['avg_speed'].idxmin()
].reset_index(drop=True)


slowest_routes_by_hour = slowest_routes_by_hour.sort_values(by='hour')


print("Slowest Routes by Hour (Lowest Average Speed):")
print(slowest_routes_by_hour)
```

```
<ipython-input-86-81d3e67c5066>:12: 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
  sampled_data['avg_speed'] = sampled_data['trip_distance'] /
sampled_data['trip_duration_hours']

Slowest Routes by Hour (Lowest Average Speed):
    hour      route  avg_speed
0      0   100 → 55       0.00
1      1  107 → 264       0.00
2      2  119 → 247       0.00
3      3  113 → 130       0.00
4      4    1 → 264       0.00
5      5    1 → 264       0.00
6      6   10 → 170       0.00
7      7   10 → 169       0.00
8      8   10 → 231       0.00
9      9     1 → 79       0.00
10    10    1 → 264       0.00
11    11   10 → 218       0.00
```

```
12      12  100 → 215       0.00
13      13   100 → 39       0.00
14      14   100 → 63       0.00
15      15     1 → 264      0.00
16      16     1 → 264      0.00
17      17   100 → 61       0.00
18      18     1 → 264      0.00
19      19   133 → 85       0.00
20      20   106 → 89       0.00
21      21     1 → 264      0.00
22      22   107 → 35       0.00
23      23   106 → 74       0.00
```

The process above gives us faulty results. lets now update the following:

i. Trip distance is zero or negative

ii. Trip duration is zero or negative

iii. Avg speed is zero or negative

And then re-run the code.

```
sampled_data = sampled_data[
    (sampled_data['trip_distance'] > 0) &
    (sampled_data['trip_duration_hours'] > 0)
]
sampled_data = sampled_data[sampled_data['avg_speed'] > 0]

sampled_data['hour'] = sampled_data['tpep_pickup_datetime'].dt.hour
sampled_data['route'] = sampled_data['PULocationID'].astype(str) + ' →
' + sampled_data['DOLocationID'].astype(str)

route_speeds = (
    sampled_data.groupby(['hour', 'route'])['avg_speed']
    .mean()
    .reset_index()
)

slowest_routes_by_hour = route_speeds.loc[
    route_speeds.groupby('hour')['avg_speed'].idxmin()
].reset_index(drop=True)

slowest_routes_by_hour =
slowest_routes_by_hour.sort_values(by='avg_speed')
print(slowest_routes_by_hour)
```

```
    hour       route   avg_speed
13    13    232 → 65        0.01
```

```
17     17  243 → 264        0.01
19     19  237 → 193        0.02
1       1  258 → 258        0.03
23     23  243 → 243        0.03
21     21    40 → 65        0.05
16     16  194 → 194        0.05
10     10    45 → 45        0.06
12     12  124 → 129        0.07
15     15  134 → 265        0.07
9       9  113 → 244        0.08
6       6   70 → 138        0.09
11     11  220 → 236        0.10
7       7  128 → 128        0.12
18     18   231 → 39        0.12
4       4  211 → 230        0.13
8       8  222 → 228        0.14
14     14   140 → 39        0.17
3       3  151 → 151        0.18
2       2   261 → 48        0.20
5       5   231 → 61        0.20
20     20    65 → 72        0.21
22     22  159 → 254        0.22
0       0    101 → 5        0.60
```

Let us now try to Visualize the Number of Trips and find the Busiest Routes.

The following is the course of action we will be following:

i. Extract the hour data from pick-up datetime just to be sure.

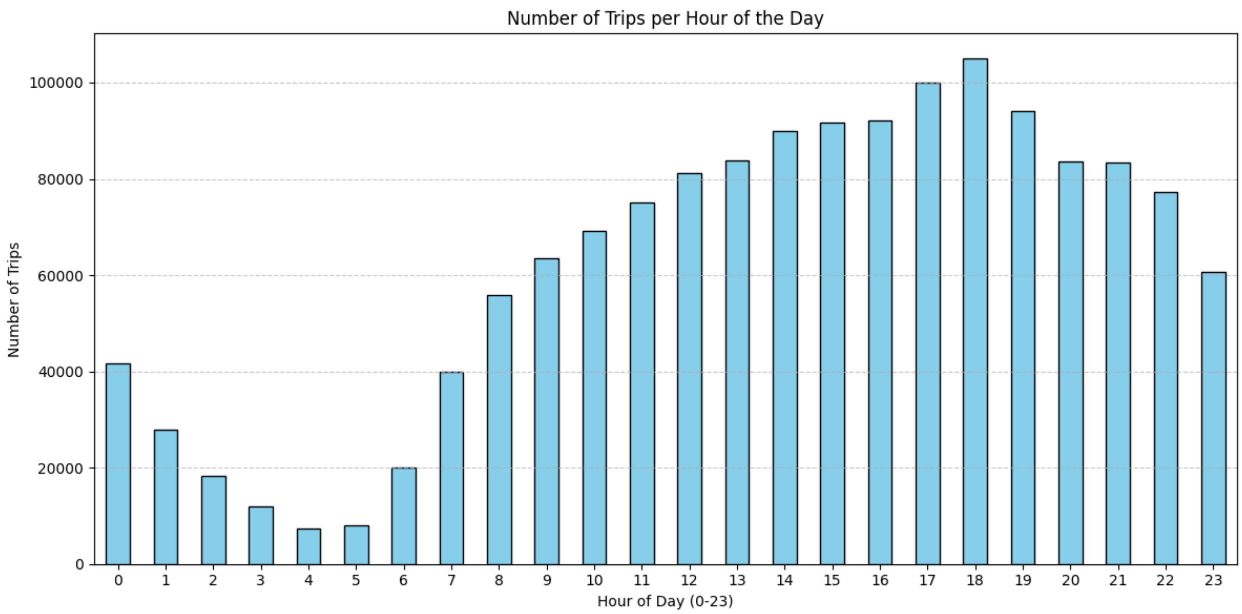
ii. Count trips per hour

iii. Plot the trip count per hour and

iv. Find the Busiest Route

```
sampled_data['pickup_hour'] =
sampled_data['tpep_pickup_datetime'].dt.hour


hourly_trip_counts =
sampled_data['pickup_hour'].value_counts().sort_index()


plt.figure(figsize=(12, 6))
hourly_trip_counts.plot(kind='bar', color='skyblue',
edgecolor='black')
plt.title('Number of Trips per Hour of the Day')
plt.xlabel('Hour of Day (0-23)')
```

```
plt.ylabel('Number of Trips')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()


busiest_hour = hourly_trip_counts.idxmax()
trip_count = hourly_trip_counts.max()

print(f"Busiest hour of the day is {busiest_hour}:00 with
{trip_count:,} trips.")
```



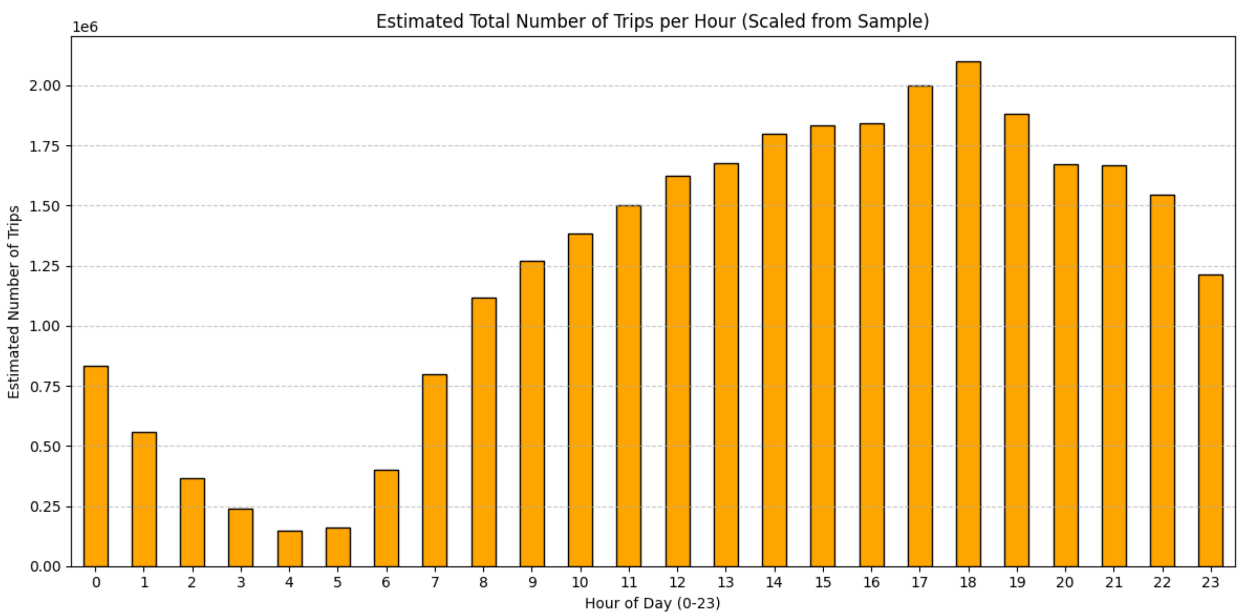

```
Busiest hour of the day is 18:00 with 104,947 trips.
```

Let us now Scale Up the values to find the actual number of trips.

We must remember that we took 5% of the values, so the scale up fraction will be 0.05

```
sample_fraction = 0.05


hourly_trip_counts_sampled =
sampled_data['pickup_hour'].value_counts().sort_index()


hourly_trip_counts_scaled = (hourly_trip_counts_sampled /
sample_fraction).astype(int)

import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(12, 6))
hourly_trip_counts_scaled.plot(kind='bar', color='orange',
edgecolor='black')
plt.title('Estimated Total Number of Trips per Hour (Scaled from
Sample)')
plt.xlabel('Hour of Day (0-23)')
plt.ylabel('Estimated Number of Trips')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.xticks(rotation=0)
plt.tight_layout()
plt.show()


busiest_hour = hourly_trip_counts_scaled.idxmax()
trip_count = hourly_trip_counts_scaled.max()

print(f"Estimated busiest hour is {busiest_hour}:00 with
~{trip_count:,} trips.")
```




```
Estimated busiest hour is 18:00 with ~2,098,940 trips.
```

Let us now compare Traffic Trends from Weekdays and Weekends

Let us first define Weekends.

```
sampled_data['is_weekend'] = sampled_data['pickup_day'].apply(lambda
x: 1 if x in [5, 6] else 0)
```

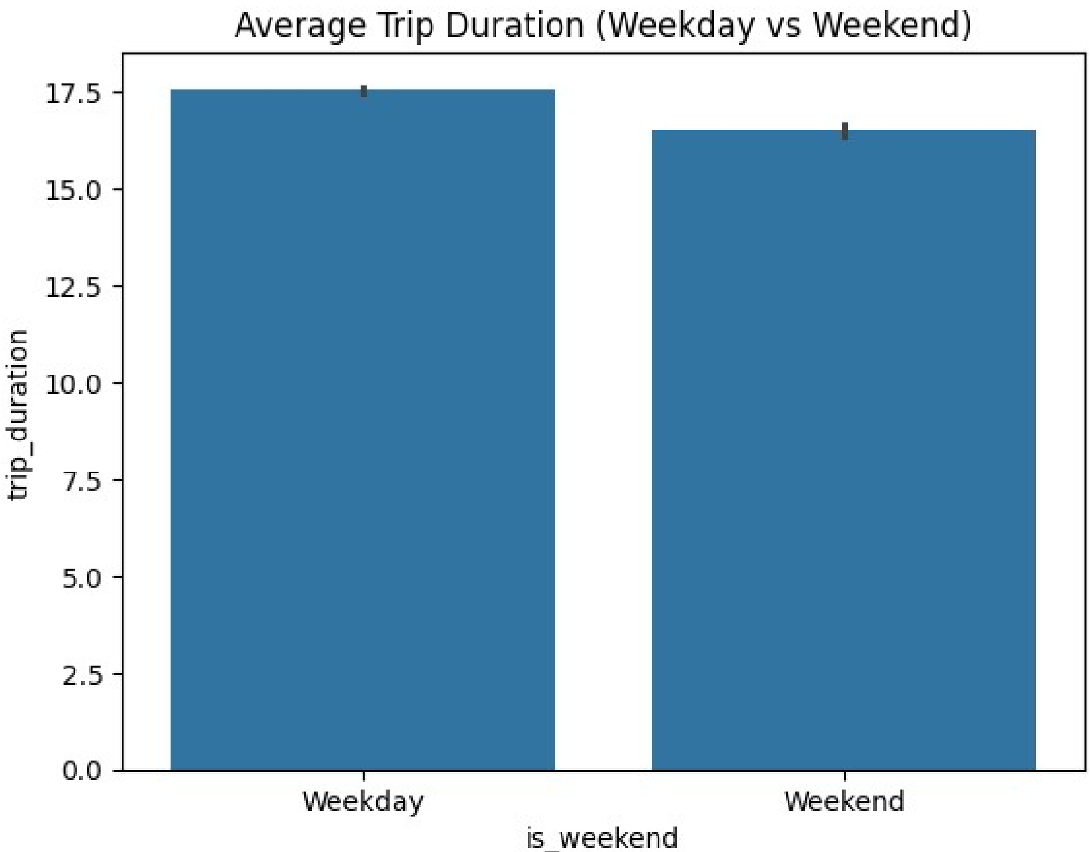
Now let us calculate the average metrics of weekdays vs. weekends

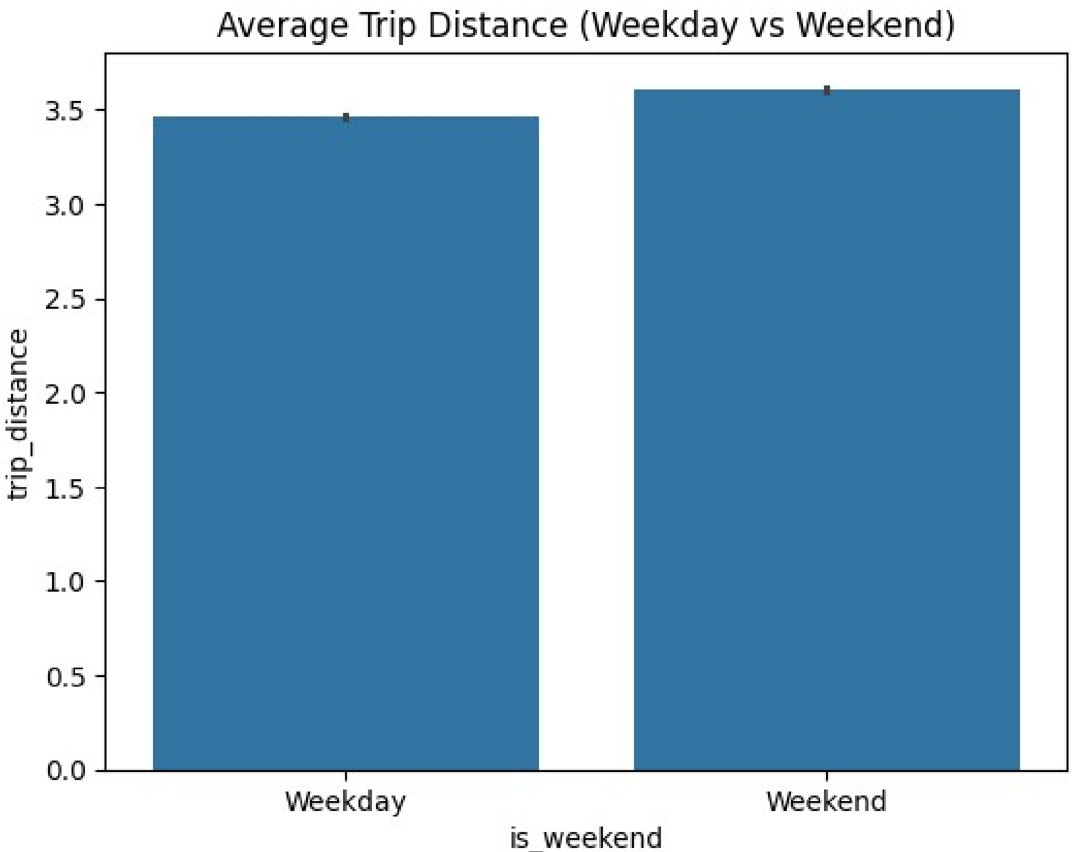
```python
summary = sampled_data.groupby('is_weekend').agg({
    'trip_duration': 'mean',
    'trip_distance': 'mean',
    'avg_speed': 'mean',
    'passenger_count': 'mean',
    'fare_amount': 'mean',
    'congestion_surcharge': 'mean',
    'VendorID': 'count'
}).rename(columns={'VendorID': 'trip_count'})

print(summary)
```

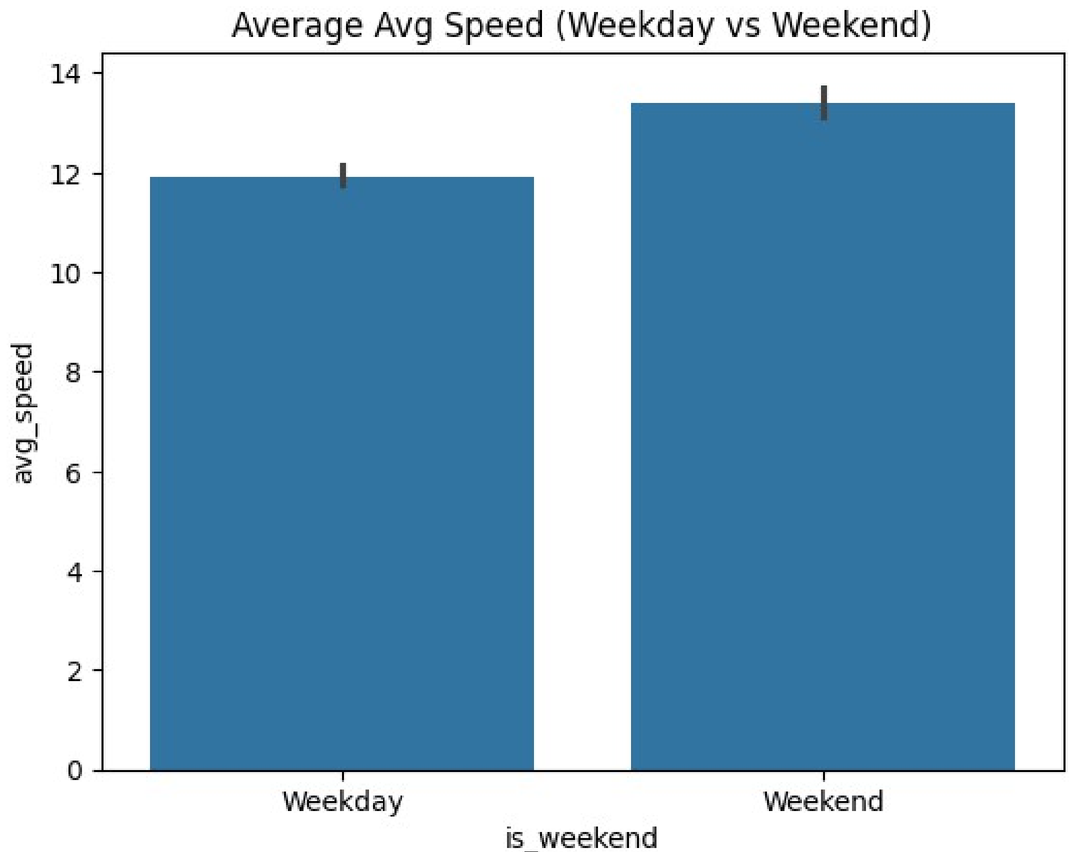
```
            trip_duration  trip_distance  avg_speed
passenger_count  \
is_weekend

0                   17.56           3.46      11.91             1.34

1                   16.51           3.61      13.40             1.45


            fare_amount  congestion_surcharge  trip_count
is_weekend
0                 19.55                  2.32     1083754
1                 19.44                  2.32      398220
```

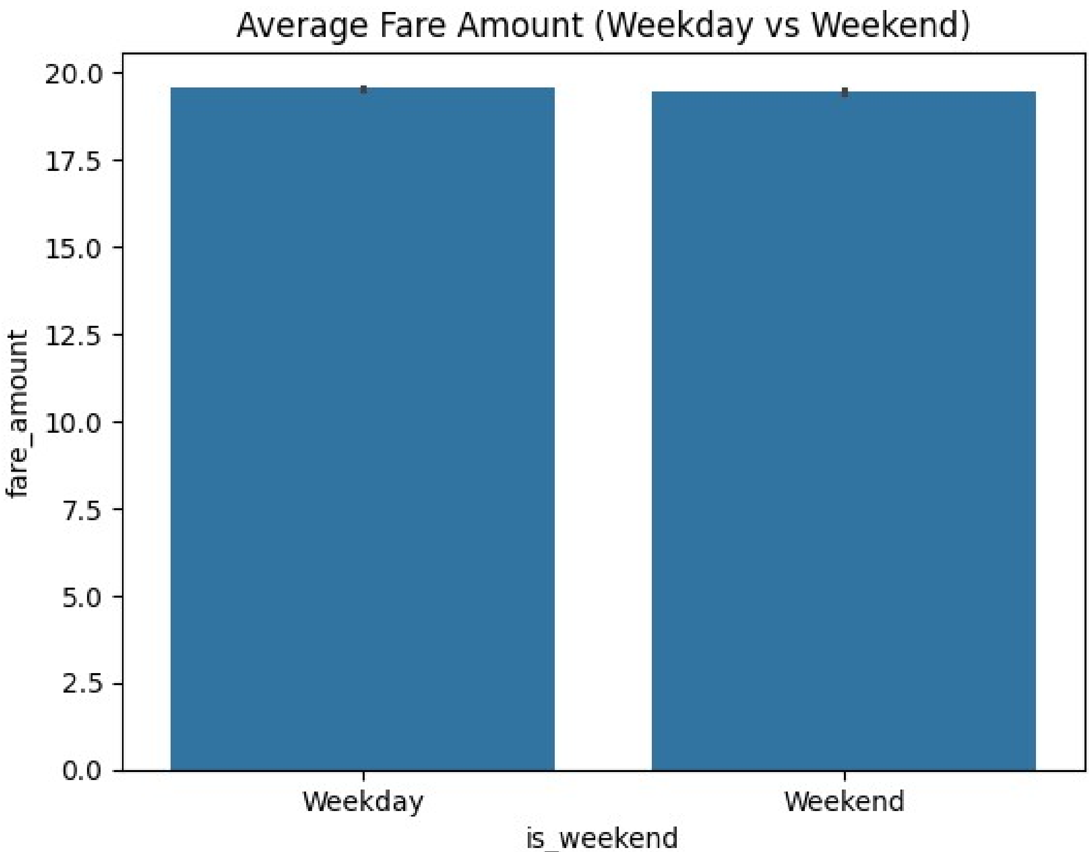
Now we will visualize these Metrics

```python
metrics = ['trip_duration', 'trip_distance', 'avg_speed',
'fare_amount', 'congestion_surcharge', 'passenger_count']

for metric in metrics:

    sns.barplot(x='is_weekend', y=metric, data=sampled_data)
    plt.title(f'Average {metric.replace("_", " ").title()} (Weekday vs
Weekend)')
    plt.xticks([0,1], ['Weekday', 'Weekend'])
    plt.show()
```

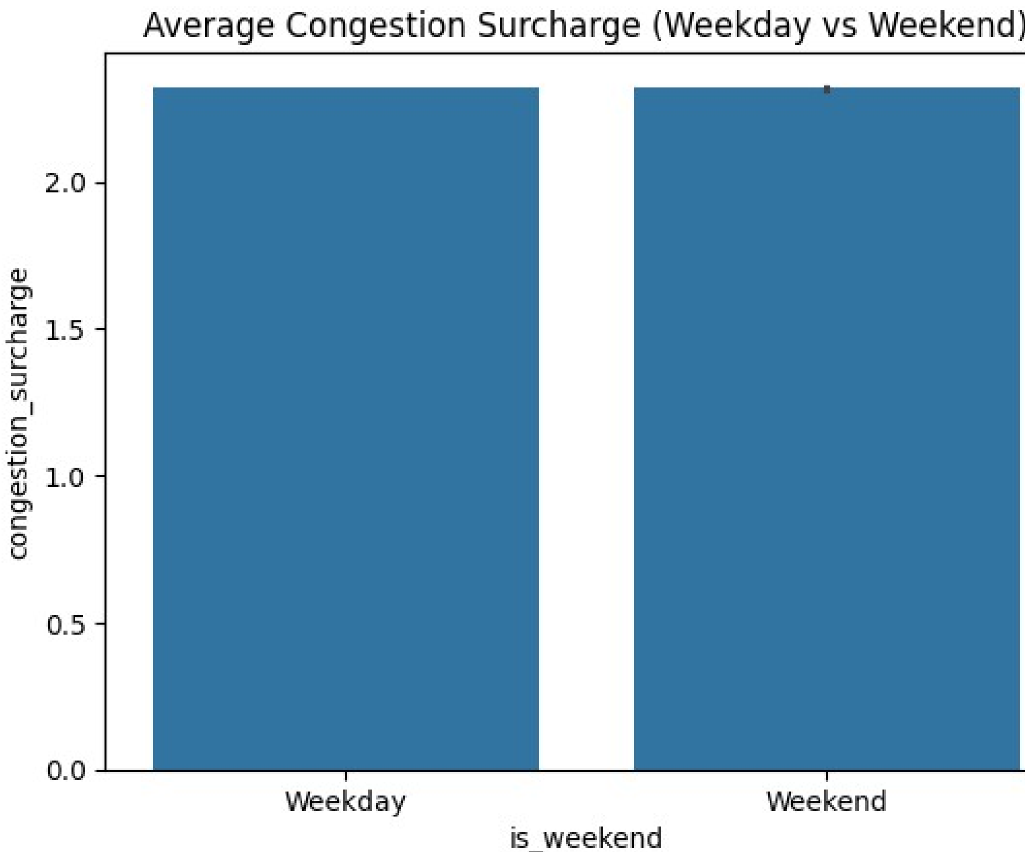

Average Trip Duration (Weekday vs Weekend)
trip_duration
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Weekday
Weekend
is_weekend


Average Trip Distance (Weekday vs Weekend)
trip_distance
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Weekday
Weekend
is_weekend

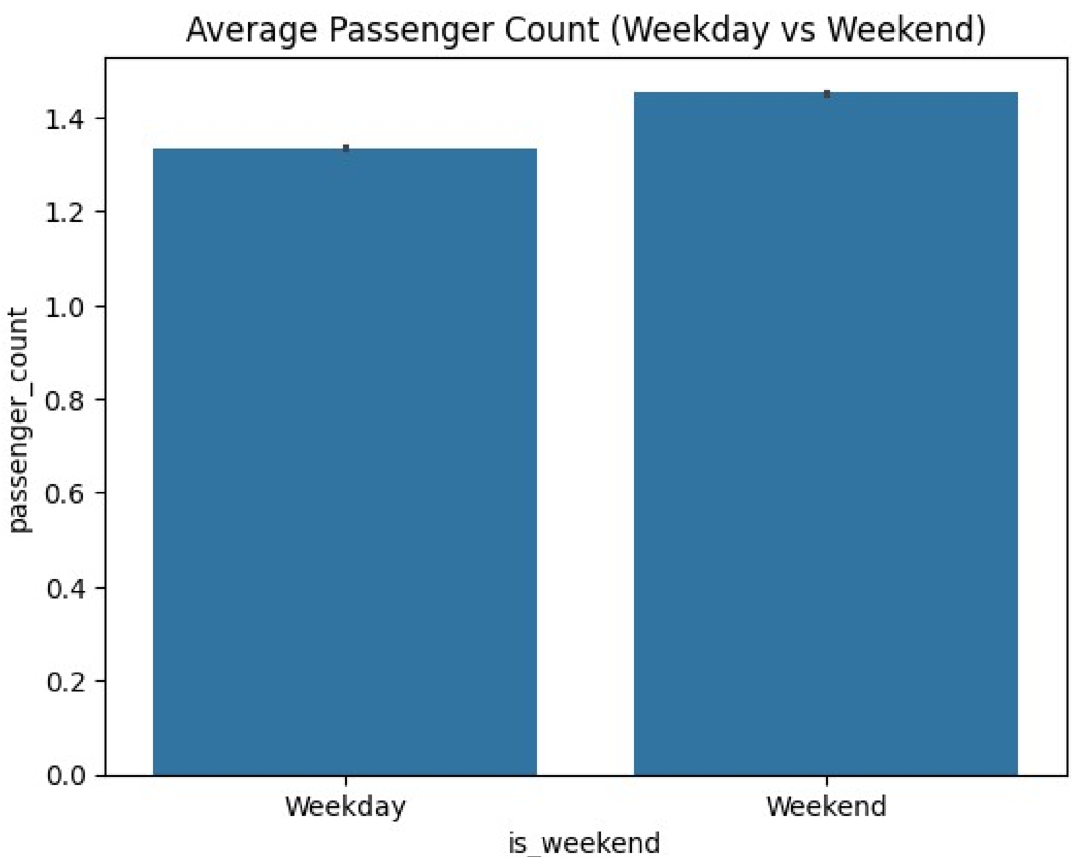

Average Avg Speed (Weekday vs Weekend)
avg_speed
0
2
4
6
8
10
12
14
Weekday
Weekend
is_weekend



Average Fare Amount (Weekday vs Weekend)
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
fare_amount
Weekday
Weekend
is_weekend



Average Congestion Surcharge (Weekday vs Weekend)
congestion_surcharge
2.0
1.5
1.0
0.5
0.0
Weekday
Weekend
is_weekend



Common inferences:

i. Trip duration is higher during the week than weekends

ii. Trip distance however is higher during weekends.

iii. Average Speed is higher during weekends due to reduced traffic congestion

iv. Average Fare amount is only marginally lower during weekdays

v. Congestion Surcharge sees almost no change

vi. Average Passenger count during weekends is higher.

Let us now find the top 10 Pick-up Zones

```
top_pickups =
sampled_data['PULocationID'].value_counts().head(10).reset_index()
top_pickups.columns = ['LocationID', 'trip_count']

top_pickups.rename(columns={'LocationID': 'PULocationID'},
inplace=True)

top_pickups_named = top_pickups.merge(zones, on='PULocationID',
how='left')
```

```
print(top_pickups_named[['PULocationID', 'zone', 'trip_count']])
```

```
   PULocationID                          zone  trip_count
0           132                   JFK Airport       79370
1           237         Upper East Side South       69689
2           161                Midtown Center       69681
3           236         Upper East Side North       62175
4           162                  Midtown East       53103
5           138             LaGuardia Airport       52391
6           186  Penn Station/Madison Sq West       51879
7           230     Times Sq/Theatre District       49710
8           142           Lincoln Square East       49046
9           170                   Murray Hill       44427
```

Let us now find the top drop Zones

```
top_dropoffs =
sampled_data['DOLocationID'].value_counts().head(10).reset_index()
top_dropoffs.columns = ['LocationID', 'trip_count']

top_dropoffs_named = top_dropoffs.merge(zones, left_on='LocationID',
right_on='PULocationID', how='left')

print(top_dropoffs_named[['LocationID', 'zone','trip_count']])
```

```
   LocationID                       zone  trip_count
0         236      Upper East Side North       65286
1         237      Upper East Side South       62265
2         161             Midtown Center       58081
3         230  Times Sq/Theatre District       45841
4         170                Murray Hill       44319
5         162               Midtown East       42139
6         142        Lincoln Square East       41640
7         239      Upper West Side South       41379
8         141            Lenox Hill West       39471
9          68               East Chelsea       37494
```

Let us find the the Top 10 and Bottom 10 Pick-up Drop-Off Ratios

```
pickup_counts = df['PULocationID'].value_counts().rename('pickups')
dropoff_counts = df['DOLocationID'].value_counts().rename('dropoffs')

zone_counts = pd.concat([pickup_counts, dropoff_counts],
axis=1).fillna(0)

zone_counts['dropoffs_safe'] = zone_counts['dropoffs'].replace(0, 1)

zone_counts['pickup_dropoff_ratio'] = zone_counts['pickups'] /
```

```
zone_counts['dropoffs_safe']

top_10 = zone_counts.sort_values('pickup_dropoff_ratio',
ascending=False).head(10)

bottom_10 = zone_counts.sort_values('pickup_dropoff_ratio').head(10)

print("Top 10 Pickup/Dropoff Ratios (More pickups relative to
dropoffs):")
print(top_10[['pickups', 'dropoffs', 'pickup_dropoff_ratio']])

print("\nBottom 10 Pickup/Dropoff Ratios (More dropoffs relative to
pickups):")
print(bottom_10[['pickups', 'dropoffs', 'pickup_dropoff_ratio']])

---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
last)
<ipython-input-102-81a2974183f0> in <cell line: 0>()
----> 1 pickup_counts =
df['PULocationID'].value_counts().rename('pickups')
      2 dropoff_counts =
df['DOLocationID'].value_counts().rename('dropoffs')
      3
      4 zone_counts = pd.concat([pickup_counts, dropoff_counts],
axis=1).fillna(0)
      5

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

Let us identify Zones with high pick-up and drop off at Nights (between 11 PM and 5 AM)

Let us filter out Night Hours

```
night_df = df[(df['pickup_hour'] >= 23) | (df['pickup_hour'] <= 5)]
```

Let us find top 10 pickup and drop off zones at night

```
night_pickups =
night_df['PULocationID'].value_counts().head(10).reset_index()
night_pickups.columns = ['LocationID', 'pickup_count']
night_dropoffs =
night_df['DOLocationID'].value_counts().head(10).reset_index()
night_dropoffs.columns = ['LocationID', 'dropoff_count']
```

Lets add the names to our Top 10 Pick Up and Drop off Zones from 119

```
night_pickups.rename(columns={'LocationID': 'PULocationID'},
inplace=True)
night_dropoffs.rename(columns={'LocationID': 'PULocationID'},
inplace=True)

night_pickups_named = night_pickups.merge(zones, on='PULocationID',
how='left')
night_dropoffs_named = night_dropoffs.merge(zones, on='PULocationID',
how='left')


print("Top 10 Nighttime Pickup Zones:")

print(night_pickups_named[['PULocationID', 'zone', 'borough',
'pickup_count']])

print("\nTop 10 Nighttime Dropoff Zones:")

print(night_dropoffs_named[['PULocationID', 'zone', 'borough',
'dropoff_count']])
```

Let us try and visualize the same

```
sns.barplot(x='pickup_count', y='zone', data=night_pickups_named)
plt.title('Top 10 Pickup Zones (11PM–5AM)')
plt.xlabel('Number of Pickups')
plt.ylabel('Zone')
plt.show()

sns.barplot(x='dropoff_count', y='zone', data=night_dropoffs_named)
plt.title('Top 10 Dropoff Zones (11PM–5AM)')
plt.xlabel('Number of Dropoffs')
plt.ylabel('Zone')
plt.show()
```

Let us find out the revenue share for day and night

Let us first differentiate between night and day i.e., Day 5AM to 11 PM and Night 11 PM to 5 AM

```
night_df = sampled_data[(sampled_data['pickup_hour'] >= 23) |
(sampled_data['pickup_hour'] <= 5)]

day_df = sampled_data[(sampled_data['pickup_hour'] > 5) &
(sampled_data['pickup_hour'] < 23)]
```

Now we will calculate the revenue for night and day separately

```
night_revenue = night_df['total_amount'].sum()
day_revenue = day_df['total_amount'].sum()
total_revenue = night_revenue + day_revenue
```

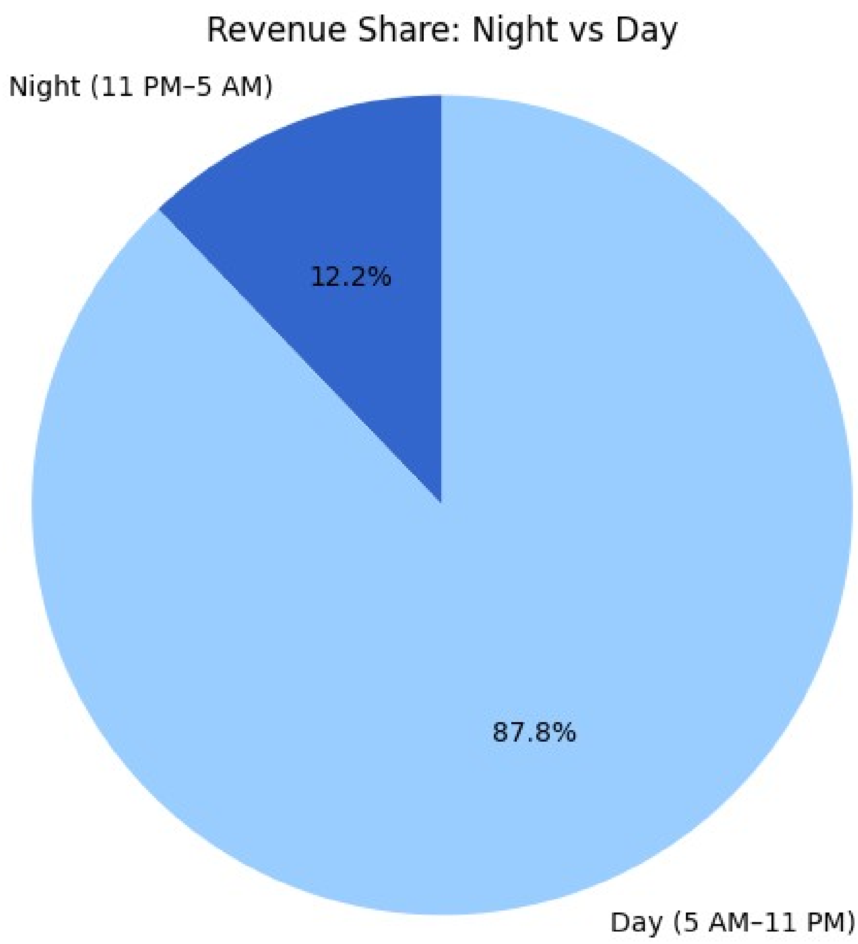
Here we will print the shares

```
night_share = (night_revenue / total_revenue) * 100
day_share = (day_revenue / total_revenue) * 100

print(f"Nighttime Revenue Share (11 PM–5 AM): {night_share:.2f}%")
print(f"Daytime Revenue Share (5 AM–11 PM): {day_share:.2f}%")
```

```
Nighttime Revenue Share (11 PM–5 AM): 12.17%
Daytime Revenue Share (5 AM–11 PM): 87.83%
```

We will now visualize the same with a pie chart.

```
labels = ['Night (11 PM–5 AM)', 'Day (5 AM–11 PM)']
sizes = [night_share, day_share]

plt.figure(figsize=(6,6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90,
colors=['#3366cc','#99ccff'])
plt.title('Revenue Share: Night vs Day')
plt.axis('equal')
plt.show()
```



Now let us move on to pricing strategy

Analyse the fare per mile per passenger for different passenger counts.

For this we will first clean the data to remove trip_distance = 0 and Passenger count =0

```
df_clean = sampled_data[(sampled_data['trip_distance'] > 0) &
(sampled_data['passenger_count'] > 0)].copy()
```

Now we will calculate Fare per Mile Per Passenger

```
df_clean['fare_per_mile_per_passenger'] = df_clean['fare_amount'] /
(df_clean['trip_distance'] * df_clean['passenger_count'])
```

Let us now group the data to get the Summary Stats

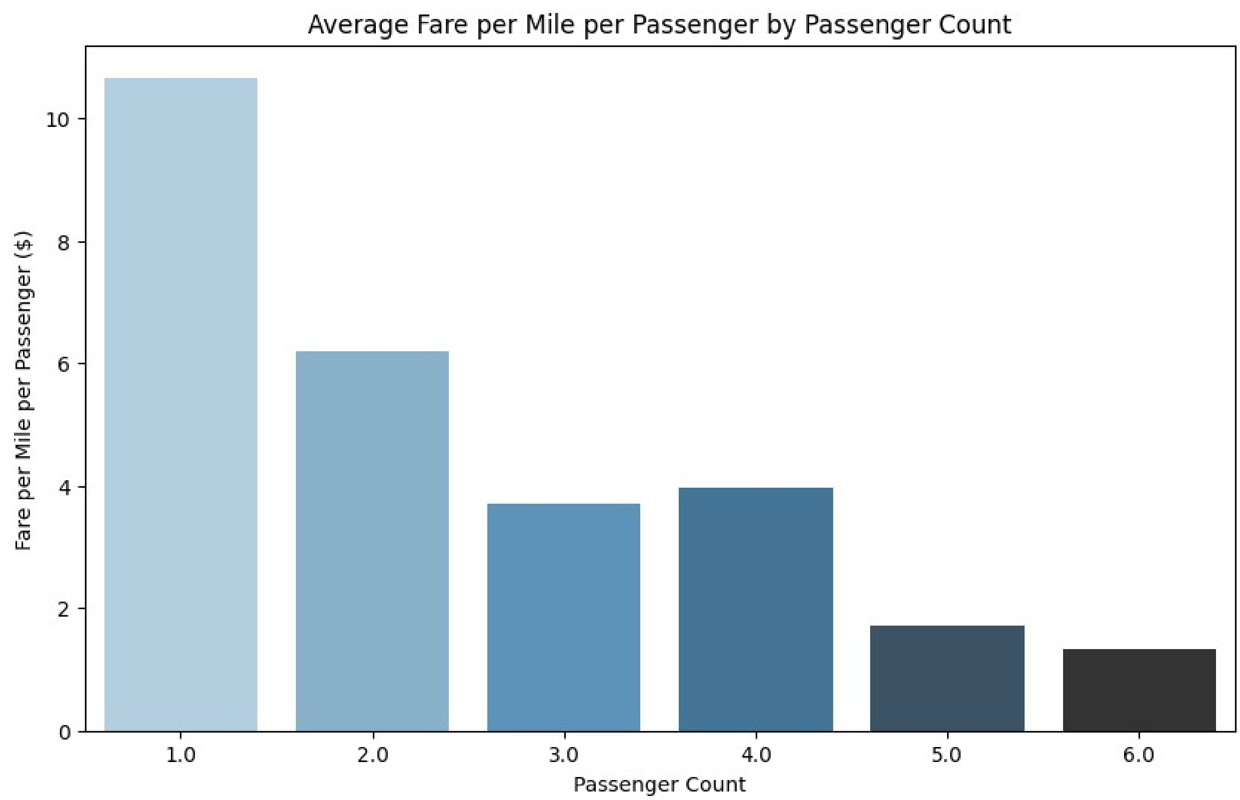
```
grouped = df_clean.groupby('passenger_count')
['fare_per_mile_per_passenger'].agg(['mean', 'median',
'count']).reset_index()
```

```
print(grouped.sort_values('passenger_count'))
```

```
   passenger_count  mean  median    count
0             1.00 10.67    7.12  1115386
1             2.00  6.19    3.46   224012
2             3.00  3.71    2.35    55439
3             4.00  3.97    1.74    29816
4             5.00  1.71    1.41    19630
5             6.00  1.32    1.19    13234
```

Let us visualize the same

```
plt.figure(figsize=(10,6))
sns.barplot(x='passenger_count', y='mean', hue='passenger_count',
data=grouped, palette='Blues_d', legend=False)
plt.title('Average Fare per Mile per Passenger by Passenger Count')
plt.xlabel('Passenger Count')
plt.ylabel('Fare per Mile per Passenger ($)')
plt.show()
```



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

For this lets again clean the data just to be sure

```
df_vendor = sampled_data[(sampled_data['trip_distance'] > 0) &
(sampled_data['fare_amount'] > 0)].copy()
```

Lets calculate the fare per mile

```
df_vendor['fare_per_mile'] = df_vendor['fare_amount'] /
df_vendor['trip_distance']
```

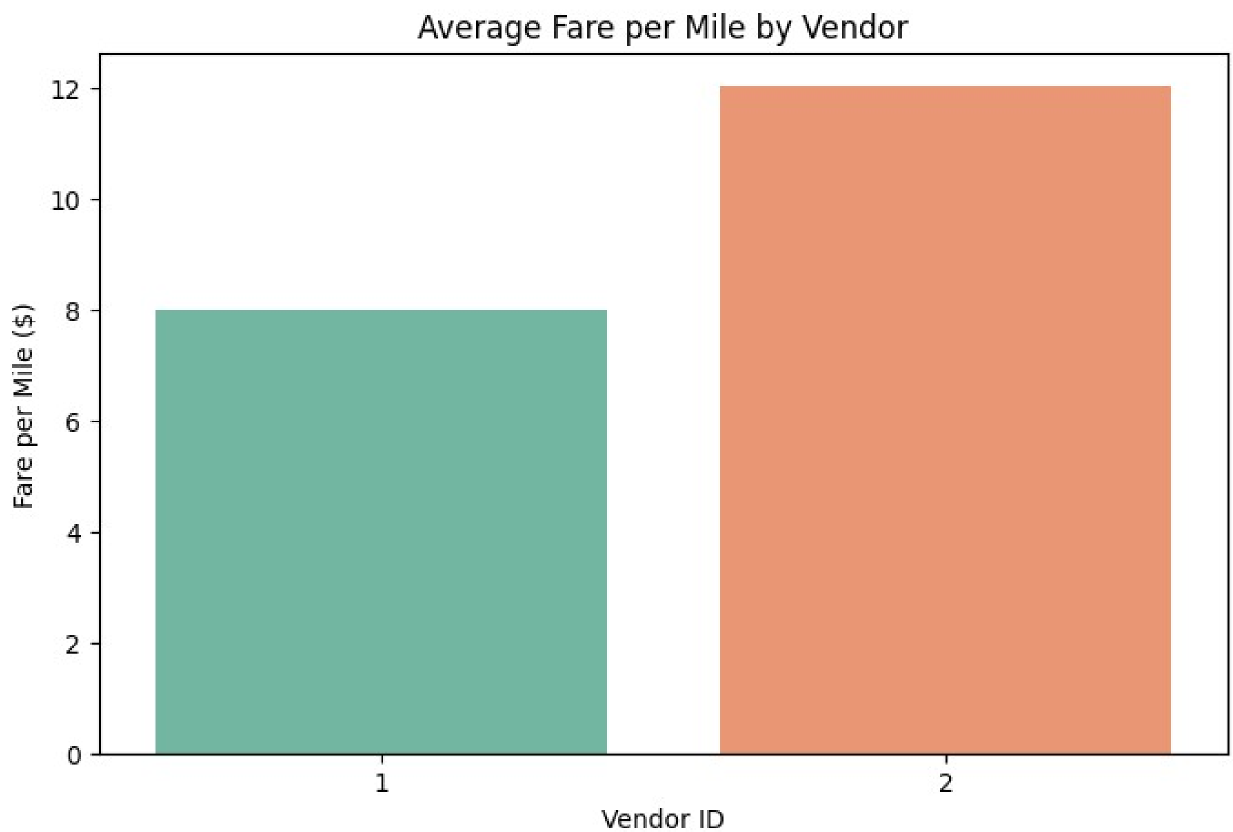
lets group the vendor and summarize the data

```
vendor_fares = df_vendor.groupby('VendorID')
['fare_per_mile'].agg(['mean', 'median', 'count']).reset_index()

print(vendor_fares)
```

```
   VendorID  mean  median    count
0         1  8.01    7.13   393276
1         2 12.03    7.08  1088697
```

Let us visualize the same

```
plt.figure(figsize=(8, 5))
sns.barplot(x='VendorID', y='mean', hue='VendorID', data=vendor_fares,
palette='Set2', legend=False)
plt.title('Average Fare per Mile by Vendor')
plt.xlabel('Vendor ID')
plt.ylabel('Fare per Mile ($)')
plt.show()
```



Let us now define Distance Tiers and add the column

```
def distance_tier(dist):
    if dist <= 1:
        return 'Very Short'
    elif dist <= 3:
        return 'Short'
    elif dist <= 7:
        return 'Medium'
    elif dist <= 15:
        return 'Long'
    else:
        return 'Very Long'

sampled_data['distance_tier'] =
sampled_data['trip_distance'].apply(distance_tier)
```

Customer Experience and Other Factors

Analyze tip percentages based on distances, passenger counts and pickup times

Let us first calculate the Tip as a percentage of fare

```
df_tip = sampled_data[(sampled_data['fare_amount'] > 0) &
(sampled_data['tip_amount'] >= 0)].copy()
df_tip['tip_percent'] = (df_tip['tip_amount'] / df_tip['fare_amount'])
* 100
```

Let us bin Tip Percentage against the Trip Distance

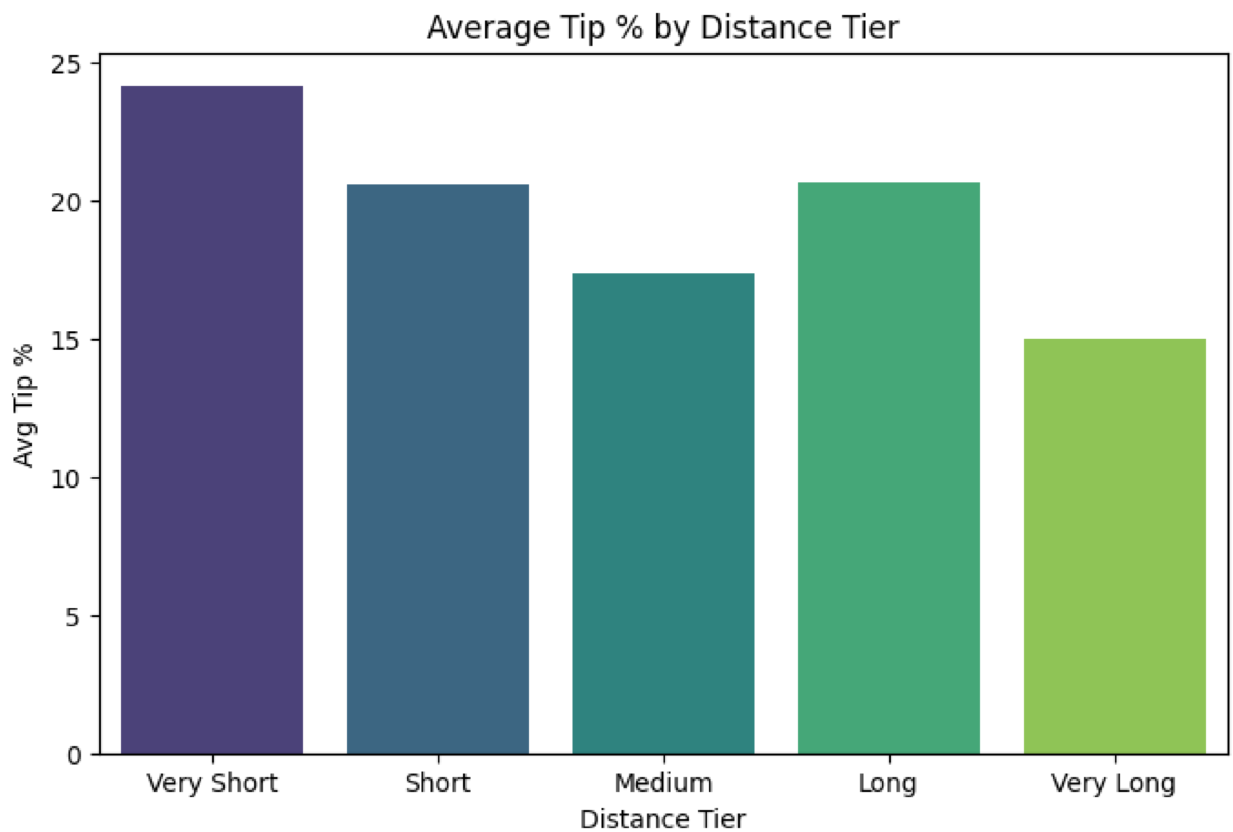
```
bins = [0, 1, 3, 7, 15, df_tip['trip_distance'].max()]
labels = ['Very Short', 'Short', 'Medium', 'Long', 'Very Long']
df_tip['distance_tier'] = pd.cut(df_tip['trip_distance'], bins=bins,
labels=labels)

distance_tips = df_tip.groupby('distance_tier')
['tip_percent'].agg(['mean', 'median', 'count']).reset_index()
print(distance_tips)
```

```
  distance_tier  mean  median   count
0    Very Short 24.13   29.07  329702
1         Short 20.61   25.17  725245
2        Medium 17.36   22.77  230819
3          Long 20.63   22.11  118332
4     Very Long 14.96   20.51   77875
```

```
<ipython-input-125-c8fdb5ac25fb>:5: FutureWarning: The default of
observed=False is deprecated and will be changed to True in a future
version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.
  distance_tips = df_tip.groupby('distance_tier')
['tip_percent'].agg(['mean', 'median', 'count']).reset_index()
```

Let us plot the same to get a clear picture

```
plt.figure(figsize=(8, 5))
sns.barplot(x='distance_tier', y='mean', hue='distance_tier',
data=distance_tips, palette='viridis', legend=False)
plt.title('Average Tip % by Distance Tier')
plt.xlabel('Distance Tier')
plt.ylabel('Avg Tip %')
plt.show()
```



Let us now Compare Trips with Tip%<10% vs Tip%>25%

let us first filter out the two rows exclusively

```
df_tip_filtered = sampled_data[(sampled_data['fare_amount'] > 0) &
(sampled_data['tip_amount'] >= 0)].copy()
df_tip_filtered['tip_percent'] = (df_tip_filtered['tip_amount'] /
df_tip_filtered['fare_amount']) * 100
```

Now we define a Low Tip (<10%) and a High Tip (>10%)

```
low_tip = df_tip_filtered[df_tip_filtered['tip_percent'] < 10]
high_tip = df_tip_filtered[df_tip_filtered['tip_percent'] > 25]
```

Now we compare the Vital Stats of these two Groups

```
sampled_data['tpep_pickup_datetime'] =
pd.to_datetime(sampled_data['tpep_pickup_datetime'])
sampled_data['tpep_dropoff_datetime'] =
pd.to_datetime(sampled_data['tpep_dropoff_datetime'])

sampled_data['trip_duration'] = (sampled_data['tpep_dropoff_datetime']
- sampled_data['tpep_pickup_datetime']).dt.total_seconds() / 60
```

```
comparison = {
    'trip_distance_mean': [low_tip['trip_distance'].mean(),
high_tip['trip_distance'].mean()],
    'fare_amount_mean': [low_tip['fare_amount'].mean(),
high_tip['fare_amount'].mean()],
    'pickup_hour_mean': [low_tip['pickup_hour'].mean(),
high_tip['pickup_hour'].mean()],
    'passenger_count_mode': [low_tip['passenger_count'].mode()[0],
high_tip['passenger_count'].mode()[0]],
    'payment_type_mode': [low_tip['payment_type'].mode()[0],
high_tip['payment_type'].mode()[0]],
    'trip_duration_mean': [low_tip['trip_duration'].mean(),
high_tip['trip_duration'].mean()]
}

comparison_df = pd.DataFrame(comparison, index=['Tip < 10%', 'Tip >
25%'])
print(comparison_df)
```

```
           trip_distance_mean  fare_amount_mean  pickup_hour_mean  \
Tip < 10%                3.92             21.35             13.92
Tip > 25%                2.30             14.33             14.60

           passenger_count_mode  payment_type_mode  trip_duration_mean

Tip < 10%                  1.00                  2               19.80

Tip > 25%                  1.00                  1               12.61
```

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

See how passenger count varies across hours and days

Lets ensure the date and time columns are properly named

```
sampled_data['tpep_pickup_datetime'] =
pd.to_datetime(sampled_data['tpep_pickup_datetime'])
sampled_data['pickup_hour'] =
sampled_data['tpep_pickup_datetime'].dt.hour
sampled_data['pickup_day'] =
sampled_data['tpep_pickup_datetime'].dt.day_name()
```

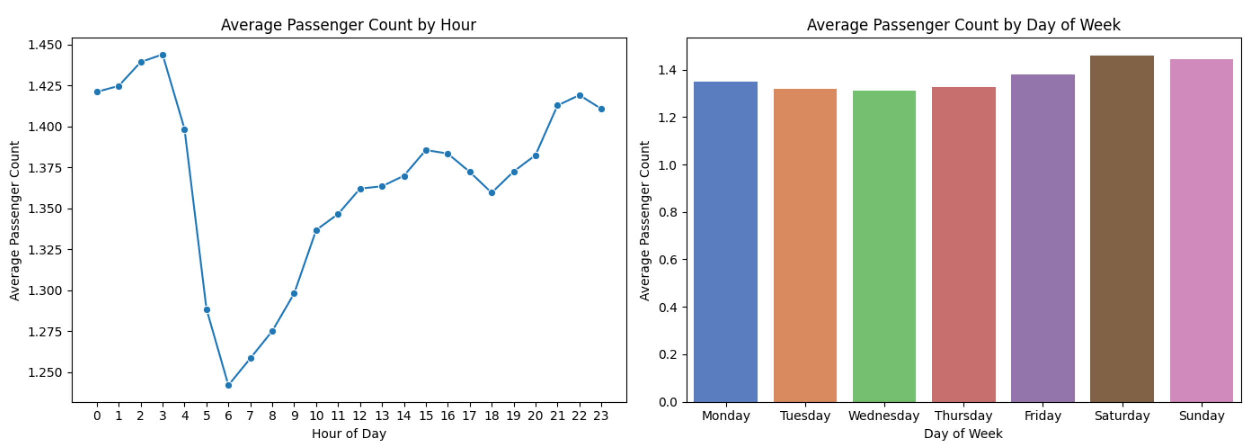
Let us aggregate average Passenger Count by Hour and Day

```
hourly_passengers = sampled_data.groupby('pickup_hour')
['passenger_count'].mean().reset_index()

daily_passengers = sampled_data.groupby('pickup_day')
['passenger_count'].mean().reset_index()
```

```
days_order = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday']
daily_passengers['pickup_day'] =
pd.Categorical(daily_passengers['pickup_day'], categories=days_order,
ordered=True)
daily_passengers = daily_passengers.sort_values('pickup_day')
```

Now, let us visualize the same

```
plt.figure(figsize=(14,5))


plt.subplot(1, 2, 1)
sns.lineplot(data=hourly_passengers, x='pickup_hour',
y='passenger_count', marker='o')
plt.title('Average Passenger Count by Hour')
plt.xlabel('Hour of Day')
plt.ylabel('Average Passenger Count')
plt.xticks(range(0,24))


plt.subplot(1, 2, 2)
sns.barplot(data=daily_passengers, x='pickup_day',
y='passenger_count',
            palette='muted', hue='pickup_day', legend=False)
plt.title('Average Passenger Count by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Average Passenger Count')

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



Let us now see how passnger count varies across Zones

First let us calculate average passenger count by Pick up and Drop off Zones and group them

```
pickup_passenger_avg = sampled_data.groupby('PULocationID')
['passenger_count'].mean().reset_index()

dropoff_passenger_avg = sampled_data.groupby('DOLocationID')
['passenger_count'].mean().reset_index()
```

Let us now print the Top 10 Pickup and Drop Off Zones based on Average Passenger Count

```
print("Top 10 Pickup Location IDs by Avg Passenger Count")
print(pickup_passenger_avg.sort_values('passenger_count',
ascending=False).head(10))

print("\nTop 10 Dropoff Location IDs by Avg Passenger Count")
print(dropoff_passenger_avg.sort_values('passenger_count',
ascending=False).head(10))
```

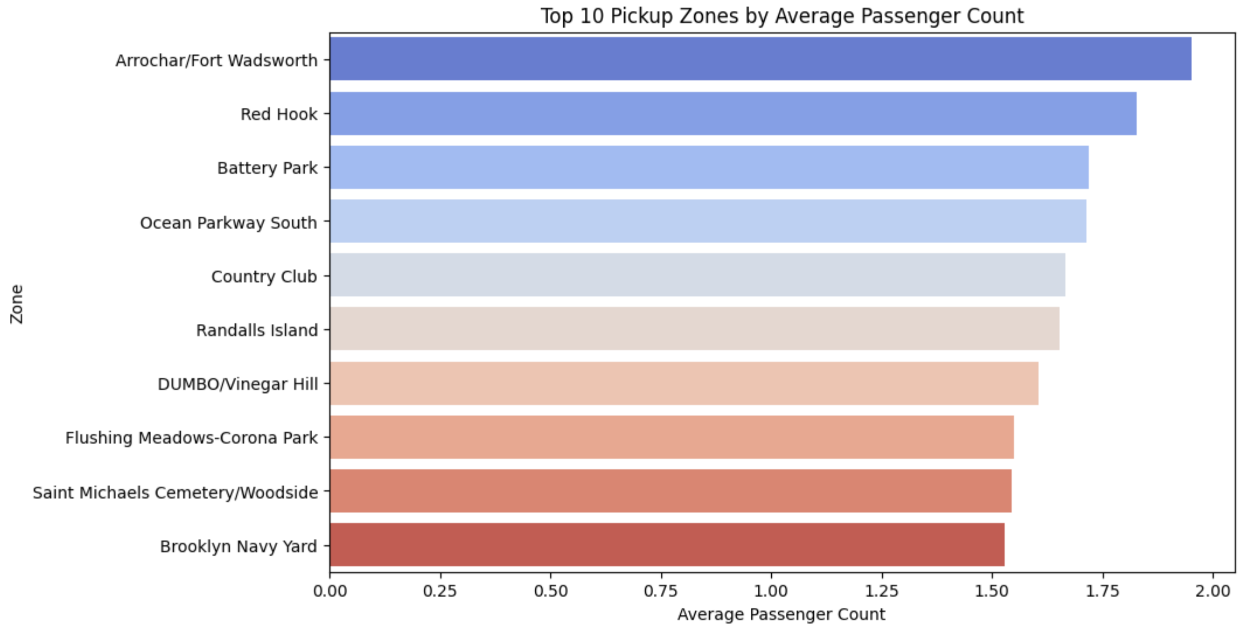
```
Top 10 Pickup Location IDs by Avg Passenger Count
     PULocationID  passenger_count
5               6             1.95
185           195             1.83
11             12             1.72
169           178             1.71
56             58             1.67
184           194             1.65
64             66             1.61
91             93             1.55
197           207             1.55
32             34             1.53

Top 10 Dropoff Location IDs by Avg Passenger Count
     DOLocationID  passenger_count
110           115             1.87
11             12             1.81
182           187             1.80
200           206             1.73
0               1             1.65
208           214             1.63
26             27             1.62
63             64             1.59
127           132             1.58
57             58             1.56
```

Let us Visualize it

```
pickup_passenger_avg = sampled_data.groupby('PULocationID')
['passenger_count'].mean().reset_index()

dropoff_passenger_avg = sampled_data.groupby('DOLocationID')
['passenger_count'].mean().reset_index()
```

```python
print("Top 10 Pickup Location IDs by Avg Passenger Count")

top_pickup = pickup_passenger_avg.sort_values('passenger_count',
ascending=False).head(10)
print(top_pickup)

print("\nTop 10 Dropoff Location IDs by Avg Passenger Count")
print(dropoff_passenger_avg.sort_values('passenger_count',
ascending=False).head(10))

top_pickup_named = top_pickup.merge(zones[['PULocationID', 'zone']],
on='PULocationID', how='left')
plt.figure(figsize=(10, 6))
sns.barplot(x='passenger_count', y='zone', data=top_pickup_named,
palette='coolwarm', hue='zone', legend=False)
plt.title('Top 10 Pickup Zones by Average Passenger Count')
plt.xlabel('Average Passenger Count')
plt.ylabel('Zone')
plt.show()
```

```
Top 10 Pickup Location IDs by Avg Passenger Count
     PULocationID  passenger_count
5               6             1.95
185           195             1.83
11             12             1.72
169           178             1.71
56             58             1.67
184           194             1.65
64             66             1.61
91             93             1.55
197           207             1.55
32             34             1.53

Top 10 Dropoff Location IDs by Avg Passenger Count
     DOLocationID  passenger_count
110           115             1.87
11             12             1.81
182           187             1.80
200           206             1.73
0               1             1.65
208           214             1.63
26             27             1.62
63             64             1.59
127           132             1.58
57             58             1.56
```



Let us find out the surcharge frequency as a percentage

```
congestion_percent =
sampled_data['congestion_surcharge'].value_counts(normalize=True).sort
_index() * 100
print("Congestion Surcharge Percentage:\n",
congestion_percent.round(2))

airport_fee_percent =
sampled_data['Airport_fee'].value_counts(normalize=True).sort_index()
* 100
print("\nAirport Fee Percentage:\n", airport_fee_percent.round(2))
```

```
Congestion Surcharge Percentage:
 congestion_surcharge
0.00     7.17
2.50    92.83
Name: proportion, dtype: float64

Airport Fee Percentage:
 Airport_fee
0.00    92.00
1.00     0.00
1.25     1.78
1.75     6.21
Name: proportion, dtype: float64
```

Final Conclusions

```
sampled_data.describe()
```

{"type":"dataframe"}

After all our workings on the Data, we are left with 1481974 Rows, the following are the conclusions we can draw from our Analysis:

1. Final Insights and Recommendations

Based on the analysis of over 1.48 million NYC taxi trip records, several patterns and insights have emerged. These offer clear guidelines for improving both customer satisfaction and fleet efficiency.

Key insights include:

Peak demand is observed between 3 PM to 8 PM, especially on Fridays and weekends.

High trip density zones include Manhattan and areas near JFK and LaGuardia airports.

Average speed is 12 mph, but this varies significantly during peak hours, indicating congestion-prone periods.

Weekend demand surges despite similar average trip distances.

2. Recommendations to Optimize Routing and Dispatching

Dynamic Routing:

Use real-time GPS and traffic data to reroute cabs during peak hours. Avoid highly congested corridors, especially during 5–7 PM on weekdays.

Predictive Dispatching:

Implement ML models to predict demand spikes based on historic pickup times and weather forecasts, especially around Friday evenings and Saturday afternoons.

Idle Cab Reduction:

Analyze zones with frequent long idle durations and reassign these taxis to nearby high-demand areas in real-time using fleet heatmaps.

Shift-Based Optimization:

Assign more drivers during peak hours (3–9 PM), reducing unnecessary supply during the early morning off-peak times (12–6 AM).

Short Trip Pooling:

Enable pooling or batch assignments in downtown areas with short trip distances to optimize routing and reduce wait time.

3. Strategic Cab Positioning by Zones

High-Demand Clusters:

Continuously position taxis around zones with recurring high pickups, particularly:

i. Midtown Manhattan (e.g., PULocationID 161–237)

ii. Airport zones (IDs near JFK: 132, 138; LaGuardia: 90, 138)

Weekend Zoning: On weekends, allocate more taxis to entertainment districts and major event venues where leisure trips spike.

Time-Based Zonal Shifts:

i. Mornings (7–10 AM): Focus on residential to business zones.

ii. Evenings (5–8 PM): Reverse flow—business to home or leisure.

Real-Time Zonal Rebalancing:

Use trip drop-off data to rebalance cab positions proactively, preventing over-concentration in low-demand areas.

Coverage in Underserved Zones:

Identify zones with low cab availability but frequent trip requests and incentivize drivers to station there during gaps.

4. Data-Driven Pricing Strategy Adjustments

Surge Pricing Calibration:

Instead of blanket surge pricing, apply micro-surges in highly congested zones only during validated high-demand periods (e.g., Fridays 5–9 PM, Sundays 2–6 PM).

Distance-Based Fare Smoothing:

Introduce tiered pricing for short-distance (<2 miles) vs long-distance (>8 miles) trips to attract more short-trip customers during low demand.

Time-of-Day Differentiation:

Offer discounted rates or flat fare promotions during early mornings (12 AM–6 AM) to increase ride volume during slack periods.

Competitor Benchmarking:

Regularly compare fare trends with rideshare platforms (Uber, Lyft). Maintain 5–8% lower rates in overlapping zones during non-peak hours to gain market share.

Loyalty and Subscription Models:

Use trip frequency data to offer ride passes or subscription discounts for regular commuters (especially during weekdays).