Report: Optimising NYC Taxi Operations

Include your visualisations, analysis, results, insights, and outcomes. Explain your methodology and approach to the tasks. Add your conclusions to the sections.

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

```
from google.colab import drive
drive.mount('/content/drive')
%cd /content/drive/MyDrive/Colab
```

1. Data Preparation

1.1. Loading the dataset

1.1.1. Sample the data and combine the files

read all the parquet file from PC or google drive

```
df = pd.read_parquet('2023-1.parquet')
sampled_data = df.sample(frac = 0.05 ,random_state = 42)
once file is read, please take a sample of 5% from each file.
Combine all the files into single file.
```

```
df = pd.read_parquet('2023-1.parquet')
sampled_data = df.sample(frac = 0.05 ,random_state = 42)
#sampled_data.info()
df = pd.read_parquet('2023-2.parquet')
df1= df.sample(frac = 0.05 ,random_state = 42)
sampled_data = pd.concat([sampled_data, df1])
#sampled_data.info()
df = pd.read_parquet('2023-3.parquet')
df1= df.sample(frac = 0.05 ,random_state = 42)
sampled_data = pd.concat([sampled_data, df1])
#sampled_data.info()
df = pd.read_parquet('2023-4.parquet')
df1= df.sample(frac = 0.05 ,random_state = 42)
sampled_data = pd.concat([sampled_data, df1])
```

```
#sampled_data.info()
df = pd.read_parquet('2023-5.parquet')
df1= df.sample(frac = 0.05 ,random_state = 42)
sampled_data = pd.concat([sampled_data, df1])
#sampled_data.info()
df = pd.read_parquet('2023-6.parquet')
df1= df.sample(frac = 0.05 ,random_state = 42)
sampled_data = pd.concat([sampled_data, df1])
#sampled_data.info()
df = pd.read_parquet('2023-7.parquet')
df1= df.sample(frac = 0.05 ,random_state = 42)
sampled_data = pd.concat([sampled_data, df1])
#sampled_data.info()
df = pd.read_parquet('2023-8.parquet')
df1= df.sample(frac = 0.05 ,random_state = 42)
sampled_data = pd.concat([sampled_data, df1])
#sampled_data.info()
df = pd.read_parquet('2023-9.parquet')
df1= df.sample(frac = 0.05 ,random_state = 42)
sampled_data = pd.concat([sampled_data, df1])
#sampled_data.info()
df = pd.read_parquet('2023-10.parquet')
df1= df.sample(frac = 0.05 ,random_state = 42)
sampled_data = pd.concat([sampled_data, df1])
#sampled_data.info()
df = pd.read_parquet('2023-11.parquet')
df1= df.sample(frac = 0.05 ,random_state = 42)
sampled_data = pd.concat([sampled_data, df1])
#sampled_data.info()
df = pd.read_parquet('2023-12.parquet')
df1= df.sample(frac = 0.05 ,random_state = 42)
sampled_data = pd.concat([sampled_data, df1])
sampled_data.info()
sampled_data.to_parquet('2023_sample.parquet')


write the sample data into single file

import os
#print(os.name)
#Print current working directory
print("Current directory:" , os.getcwd())
# Create a list of all the twelve files to read
file_list = os.listdir()
#print(os.listdir())
# initialise an empty dataframe
df = pd.DataFrame()
#df.info()
```

```
sampled_data = pd.DataFrame()
#sampled_data.info()
from datetime import datetime
for file_name in file_list:
    try:
        # file path for the current file
        file_path = os.path.join(os.getcwd(), file_name)
        print(file_path)
        df = pd.read_parquet(file_path)


        #df1 = df.sample(frac = 0.05 ,random_state = 42)
        df1 = df.sample(n=30000,random_state = 42)
        sampled_data = pd.concat([sampled_data, df1])
        #sampled_data.info()
    except Exception as e:
        print(f"Error reading file {file_name}: {e}")

sampled_data.info()
sampled_data.to_parquet('2023_sample.parquet')
```

2. Data Cleaning

2.1. Fixing Columns

2.1.1. Fix the index

read the sample file and reset the index

```
df = pd.read_parquet('2023_sample.parquet')
df.head()
#df.info()
df.reset_index(inplace=True)
df.head()
del df['index']
```

Combine the two airport_fee columns

Fill the na fields with zero value and combine the fileds into single.

```
df['airport_fee'] = df['airport_fee'].fillna(0)
```

```
df['Airport_fee'] = df['Airport_fee'].fillna(0)
df['Airport_Fees_Combined'] = df['airport_fee'] + df['Airport_fee']
del df['airport_fee']
del df['Airport_fee']
df.head()
```

2.2. Handling Missing Values

2.2.1. Find the proportion of missing values in each column

```
df['store_and_fwd_flag'].info()

def find_columns_with_negative_values(df):
    negative_value_columns = []
    for col in df.columns:
        if pd.api.types.is_numeric_dtype(df[col]):
          if (df[col] < 0).any():
            negative_value_columns.append(col)
    return negative_value_columns

columns_with_negatives = find_columns_with_negative_values(df)
#print(columns_with_negatives)
df['extra'] = df['extra'].mask(df['extra'] < 0).fillna(0)
df['mta_tax'] = df['mta_tax'].mask(df['mta_tax'] < 0).fillna(0)
df['improvement_surcharge'] =
df['improvement_surcharge'].mask(df['improvement_surcharge'] <
0).fillna(0)
df['total_amount'] = df['total_amount'].mask(df['total_amount'] <
0).fillna(0)
df['congestion_surcharge'] =
df['congestion_surcharge'].mask(df['congestion_surcharge'] <
0).fillna(0)
df['Airport_Fees_Combined'] =
df['Airport_Fees_Combined'].mask(df['Airport_Fees_Combined'] <
0).fillna(0)
columns_with_negatives = find_columns_with_negative_values(df)
print(columns_with_negatives)
df.head()
(df['passenger_count'].isna()).sum()
(df['total_amount'].isna()).sum()
nan_count = df.isna().sum()
print("Count NaN values of column wise:\n", nan_count)
percentage_nan=(df.isna().sum() / len(df)) * 100
print("percentage_nan:\n", percentage_nan)
nan_count = df.isna().sum(axis = 1)
```

```
print("Count NaN values of all rows:\n", nan_count)
df.loc[:, df.isna().any()]


print("NAN of congestion_surcharge count",
(df['congestion_surcharge'].isna()).sum())


nan_count = df.isna().sum()
print("Count NaN values of column wise:\n", nan_count)

print("NAN of store_and_fwd_flag count",
(df['store_and_fwd_flag'].isna()).sum())

df['store_and_fwd_flag'] = df['store_and_fwd_flag'].fillna('N')
print("NAN of store_and_fwd_flag  count",
(df['store_and_fwd_flag'].isna()).sum())
```

2.2.2. Handling missing values in passenger_count

```
print("NAN of passenger count", (df['passenger_count'].isna()).sum())
median_value = df['passenger_count'].median()
print("median_value:\n", median_value)
df['passenger_count'] = df['passenger_count'].fillna(median_value)
print("NAN of passenger count", (df['passenger_count'].isna()).sum())
np.count_nonzero(df['passenger_count']==0)
print("zero passenger count",
np.count_nonzero(df['passenger_count']==0))
df['passenger_count']= np.where(df['passenger_count']==0, median_value,
df['passenger_count'])
np.count_nonzero(df['passenger_count']==0)
print("zero passenger count",
np.count_nonzero(df['passenger_count']==0))
```

```
df = df[df.passenger_count <= 6]
df['passenger_count'].describe()
```

	passenger_count
count	389905.000000
mean	1.374137
std	0.868763
min	1.000000
25%	1.000000
50%	1.000000
75%	1.000000
max	6.000000

dtype: float64

2.2.3 Handle missing values in RatecodeID

```
print("NAN of RatecodeID count", (df['RatecodeID'].isna()).sum())
median_value = df['RatecodeID'].median()
print("median_value:\n", median_value)
df['RatecodeID'] = df['RatecodeID'].fillna(median_value)
print("NAN of RatecodeID count", (df['RatecodeID'].isna()).sum())
```

2.2.3. Impute NaN in congestion_surcharge

```
print("NAN of congestion_surcharge count",
(df['congestion_surcharge'].isna()).sum())
```

2.3. Handling Outliers and Standardising Values

2.3.1. Check outliers in payment type, trip distance and tip amount columns

```
df['trip_distance'].value_counts()
```

trip_distance	count
0.00	7620
0.90	5299
1.00	5175
1.10	5103
0.80	5024
...	...
73.04	1
24.08	1
22.96	1
31.77	1
23.64	1

3073 rows × 1 columns

dtype: int64

```
df = df[df.trip_distance >0]

df['trip_distance'].describe()
df['trip_distance'].value_counts()
```

```
df['payment_type'].value_counts()
```

payment_type	count
1	1480783
2	309072
4	12304
3	6768

dtype: int64

trip_distance	count
0.90	5299
1.00	5175
1.10	5103
0.80	5024
1.20	4995
...	...
22.96	1
25.61	1
36.06	1
32.53	1
31.26	1

3072 rows × 1 columns

dtype: int64

3. Exploratory Data Analysis

3.1. General EDA: Finding Patterns and Trends

3.1.1. Classify variables into categorical and numerical

```
def category_var(df):
num_colm= df.select_dtypes(include=np.number).columns.tolist()
 cat_colm= df.select_dtypes(exclude=np.number).columns.tolist()

return num_colm, cat_colm

num_colm, cat_colm = category_var (df)

print("Numerical Columns:", num_colm)

print("Categorical Columns:", cat_colm)
```

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

```
df['tpep_pickup_day']=df['tpep_pickup_datetime'].dt.day_name()
```

```
df['tpep_dropoff_day']=df['tpep_dropoff_datetime'].dt.day_name()
df['tpep_pickup_day'].value_counts()
df['tpep_dropoff_day'].value_counts()
```

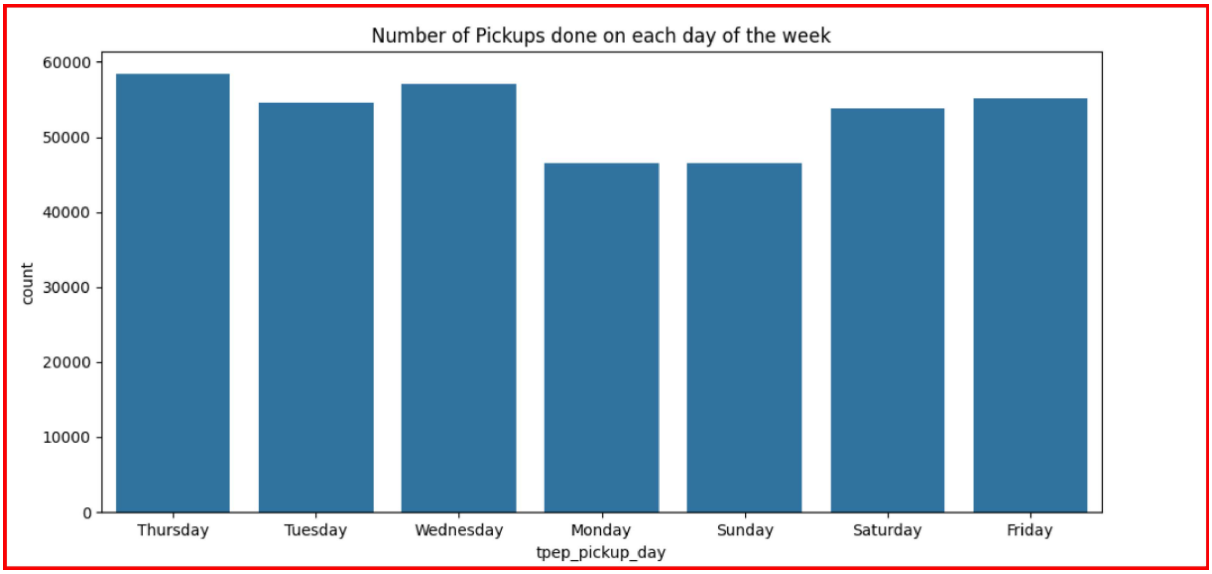
tpep_dropoff_day	count
Thursday	58273
Wednesday	57104
Friday	54963
Tuesday	54414
Saturday	53858
Sunday	46863
Monday	46609

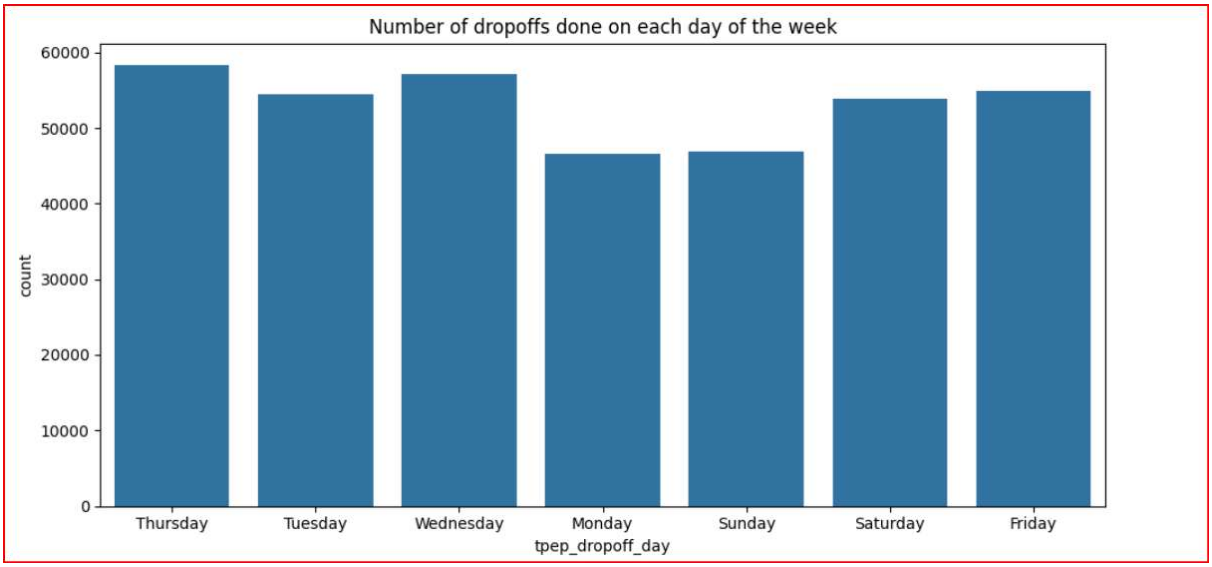
dtype: int64

tpep_pickup_day	count
Thursday	283464
Wednesday	278488
Friday	268369
Tuesday	262990
Saturday	261828
Sunday	227753
Monday	226035

dtype: int64

```
figure,ax=plt.subplots(nrows=2,ncols=1,figsize=(10,10))
sns.countplot(x='tpep_pickup_day',data=df,ax=ax[0])
ax[0].set_title('Number of Pickups done on each day of the week')
sns.countplot(x='tpep_dropoff_day',data=df,ax=ax[1])
ax[1].set_title('Number of dropoffs done on each day of the week')
plt.tight_layout()
```






```
def time_of_day(x):
    if x in range(6,12):
        return 'Morning'
    elif x in range(12,16):
        return 'Afternoon'
    elif x in range(16,22):
        return 'Evening'
    else:
        return 'Late night'
df['pickup_timeofday']=df['pickup_hour'].apply(time_of_day)
df['dropoff_timeofday']=df['dropoff_hour'].apply(time_of_day)
```

dropoff_month	count
5	167091
10	164462
3	162366
6	156873
4	156723
11	156544
12	155901
1	146724
2	138963
7	137841
8	133355
9	132084

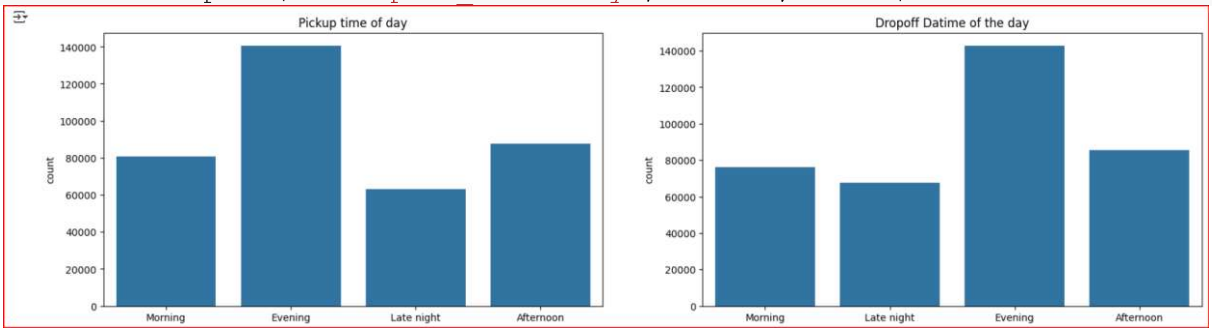
dtype: int64

pickup_month	count
5	167092
10	164427
3	162429
6	156898
4	156683
11	156585
12	155846
1	146727
2	138954
7	137800
8	133386
9	132100

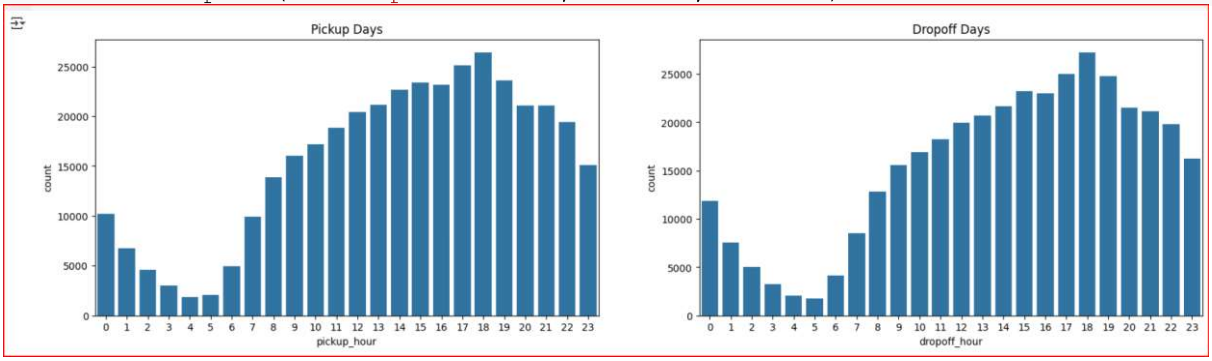
dtype: int64

```
figure,(ax1,ax2)=plt.subplots(ncols=2,figsize=(20,5))
```

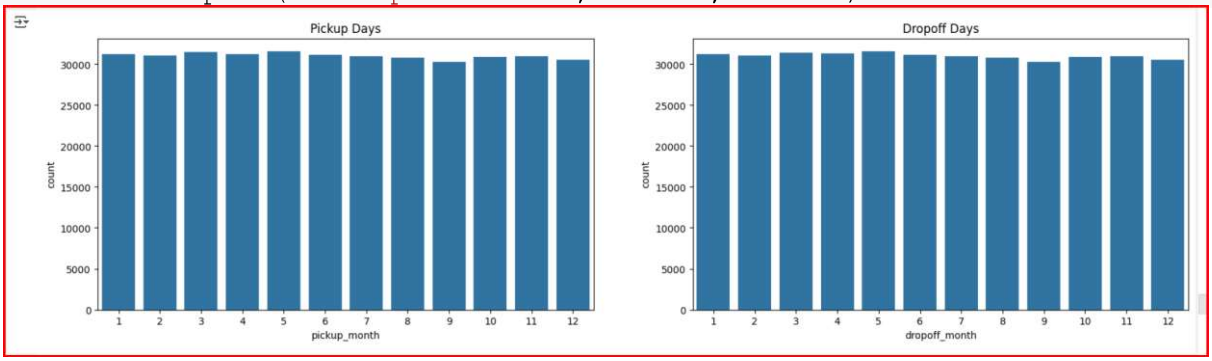
```
ax1.set_title('Pickup time of day')
ax=sns.countplot(x="pickup_timeofday",data=df,ax=ax1)

ax2.set_title('Dropoff Datime of the day')
ax=sns.countplot(x="dropoff_timeofday",data=df,ax=ax2)
```




```
figure,(ax9,ax10)=plt.subplots(ncols=2,figsize=(20,5))

ax9.set_title('Pickup hour')
ax=sns.countplot(x="pickup_hour",data=df,ax=ax9)

ax10.set_title('Dropoff hour')
ax=sns.countplot(x="dropoff_hour",data=df,ax=ax10)
```




```
figure,(ax9,ax10)=plt.subplots(ncols=2,figsize=(20,5))

ax9.set_title('Pickup Days')
ax=sns.countplot(x="pickup_month",data=df,ax=ax9)

ax10.set_title('Dropoff Days')
ax=sns.countplot(x="dropoff_month",data=df,ax=ax10)
```



3.1.2. Filter out the zero/negative values in fares, distance and tips

```
df['trip_distance'].value_counts(sort=True)
(df['trip_distance'] == 0).sum().sum()
df = df[df.trip_distance >0]
df['trip_distance'].value_counts()
df['trip_distance'].describe()
```

	trip_distance
count	372084.000000
mean	3.499246
std	4.575915
min	0.010000
25%	1.080000
50%	1.800000
75%	3.400000
max	170.300000

dtype: float64

trip_distance	count
0.90	25232
1.00	24847
1.10	24400
0.80	24320
1.20	23863
...	...
112.97	1
41.01	1
69.45	1
39.98	1
33.94	1

4136 rows × 1 columns

```
df['fare_amount'].value_counts(sort=True)
(df['fare_amount'] == 0).sum().sum()
```

```
df = df[df.fare_amount >0]
df['fare_amount'].value_counts()
#df['fare_amount'].describe()
```

fare_amount	count
8.60	82749
9.30	82435
10.00	81132
7.90	80521
10.70	77183
...	...
86.94	1
16.47	1
85.89	1
153.00	1
61.00	1

1474 rows × 1 columns

dtype: int64

	fare_amount
count	1.808927e+06
mean	1.968509e+01
std	1.812132e+01
min	1.000000e-02
25%	9.300000e+00
50%	1.350000e+01
75%	2.190000e+01
max	2.194700e+03

dtype: float64

```
df['tip_amount'].value_counts(sort=True)
(df['tip_amount'] == 0).sum().sum()
df = df[df.tip_amount >0]
df['tip_amount'].value_counts()
df['tip_amount'].describe()
```

	tip_amount
count	1.414422e+06
mean	4.593138e+00
std	4.016619e+00
min	1.000000e-02
25%	2.380000e+00
50%	3.400000e+00
75%	5.000000e+00
max	3.000000e+02

dtype: float64

tip_amount	count
2.00	92256
1.00	73300
3.00	48712
5.00	28124
2.80	21683
...	...
28.59	1
30.74	1
32.31	1
28.03	1
35.98	1

3807 rows × 1 columns

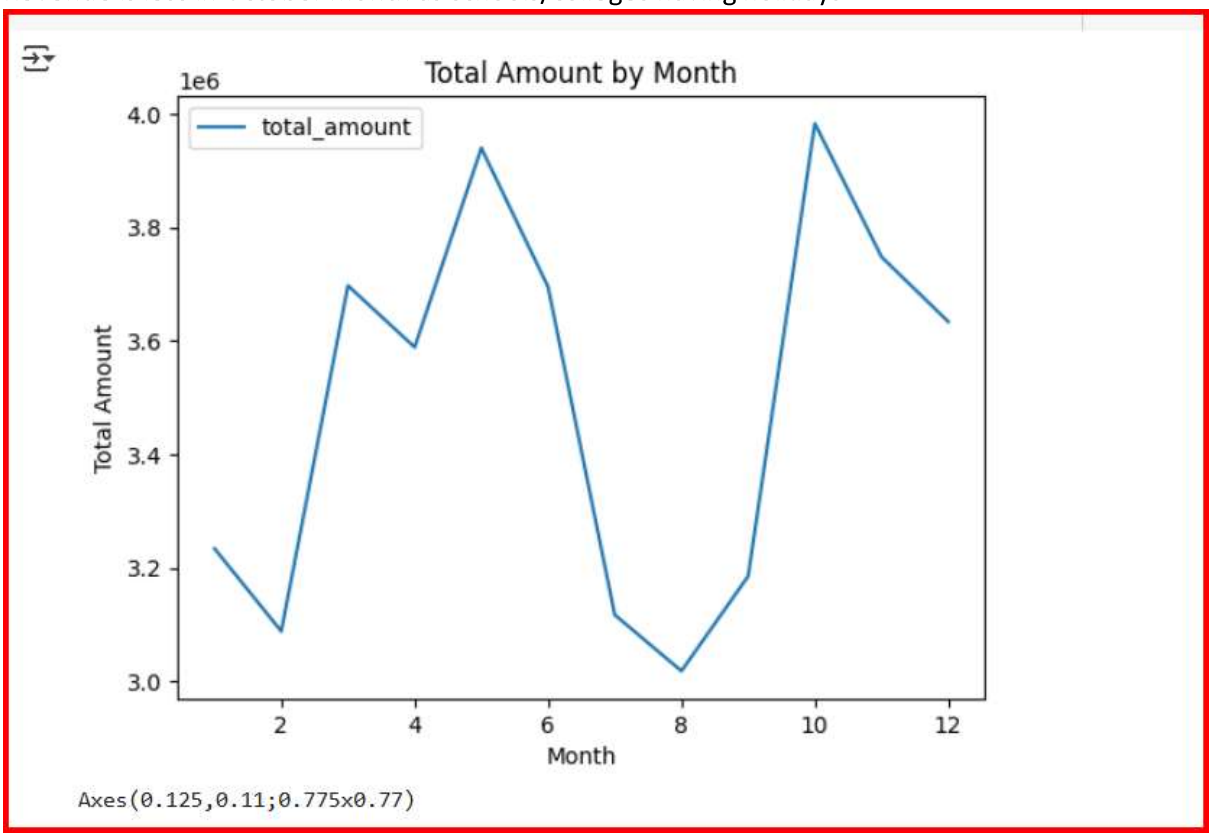
dtype: int64

3.1.3. Analyse the monthly revenue trends

```
grouped_data = df.groupby('pickup_month').agg({'total_amount':
'sum'}).plot()
grouped_data.set_xlabel('Month')
grouped_data.set_ylabel('Total Amount')
grouped_data.set_title('Total Amount by Month')
plt.show()
```

	pickup_month	total_amount
0	1	3234203.96
1	2	3088512.87
2	3	3697270.14
3	4	3588857.24
4	5	3939624.77
5	6	3695156.06
6	7	3117520.45
7	8	3018598.10
8	9	3185606.27
9	10	3982905.14
10	11	3747829.24
11	12	3633889.59

Revenue is less in October month as schools/colleges having holidays

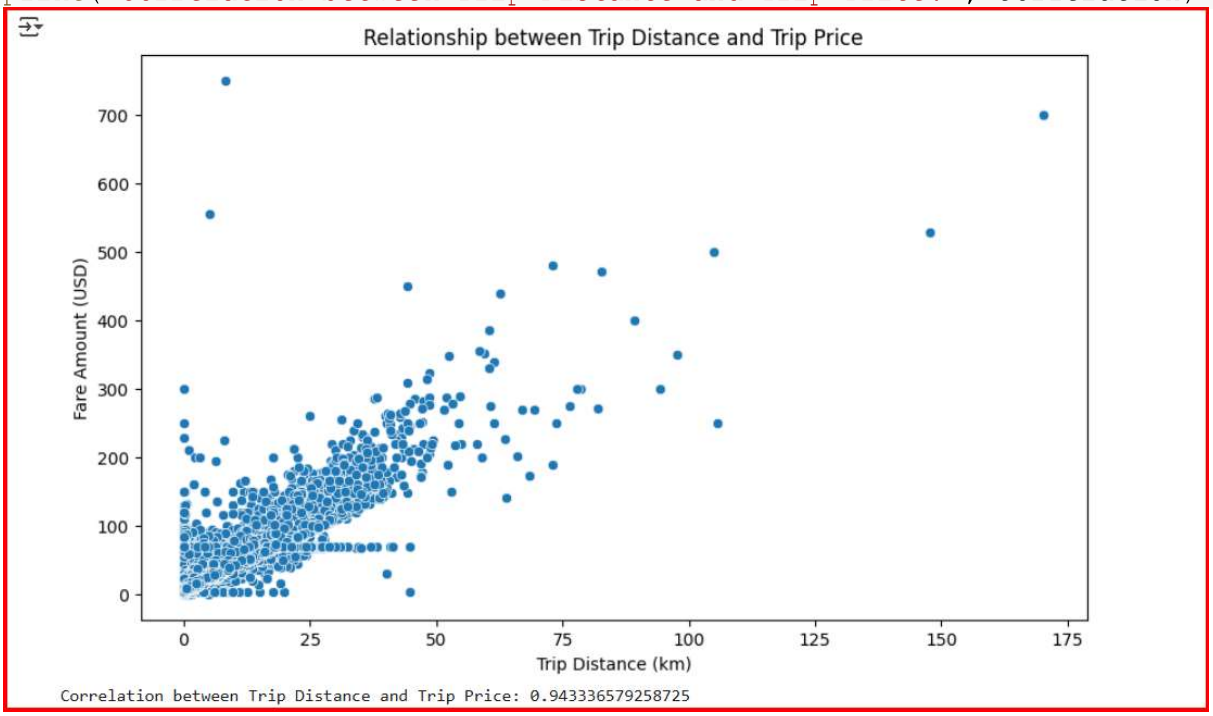


3.1.4. Find the proportion of each quarter's revenue in the yearly revenue

3.1.5. Analyse and visualise the relationship between distance and fare amount

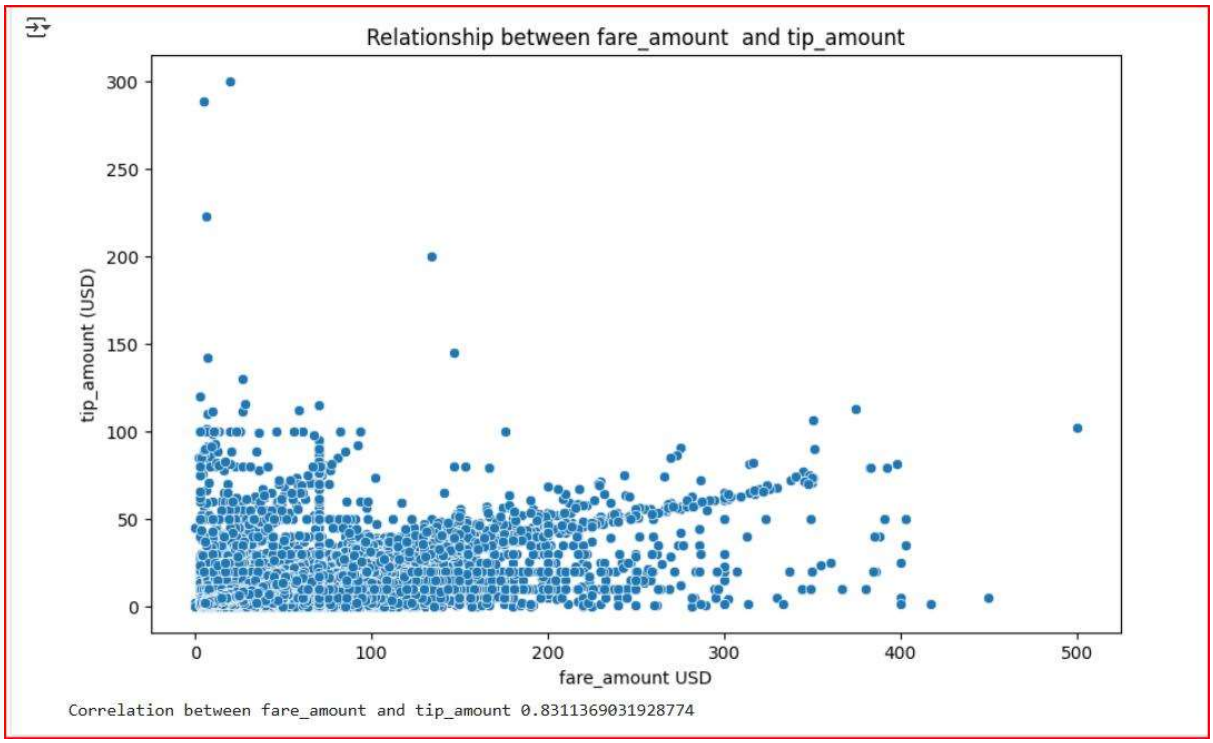
Analyse and visualise the relationship between distance and fare amount

```
#visualization of relationship between distance and fare amount

plt.figure(figsize=(10,6))
sns.scatterplot(x='trip_distance', y='fare_amount', data= df)
plt.title('Relationship between Trip Distance and Trip Price')
plt.xlabel('Trip Distance (km)')
plt.ylabel('Fare Amount (USD)')
plt.show()

#calculate the correlation between distance and fare
correlation = df['trip_distance'].corr(df['fare_amount'])
print("Correlation between Trip Distance and Trip Price:", correlation)
```



3.1.6. Analyse the relationship between fare/tips and trips/passengers

```
#visualization of relationship between fare_amount and tip_amount
amount
plt.figure(figsize=(10,6))
sns.scatterplot(x='fare_amount', y='tip_amount', data= df)
plt.title('Relationship between fare_amount  and tip_amount')
plt.xlabel('fare_amount USD')
plt.ylabel('tip_amount (USD)')
plt.show()

#calculate the correlation between distance and fare
correlation = df['fare_amount'].corr(df['tip_amount'])
print("Correlation between fare_amount and tip_amount", correlation)
```



3.1.7. Analyse the distribution of different payment types

```
grouped_data = df.groupby('payment_type').agg({'total_amount':
'sum'}).reset_index()
print(grouped_data)
```

	payment_type	total_amount
0	1	41928482.13
1	2	480.64
2	3	254.84
3	4	756.22

```
df['payment_type'].value_counts()
```

payment_type	count
1	1414379
2	20
4	18
3	5

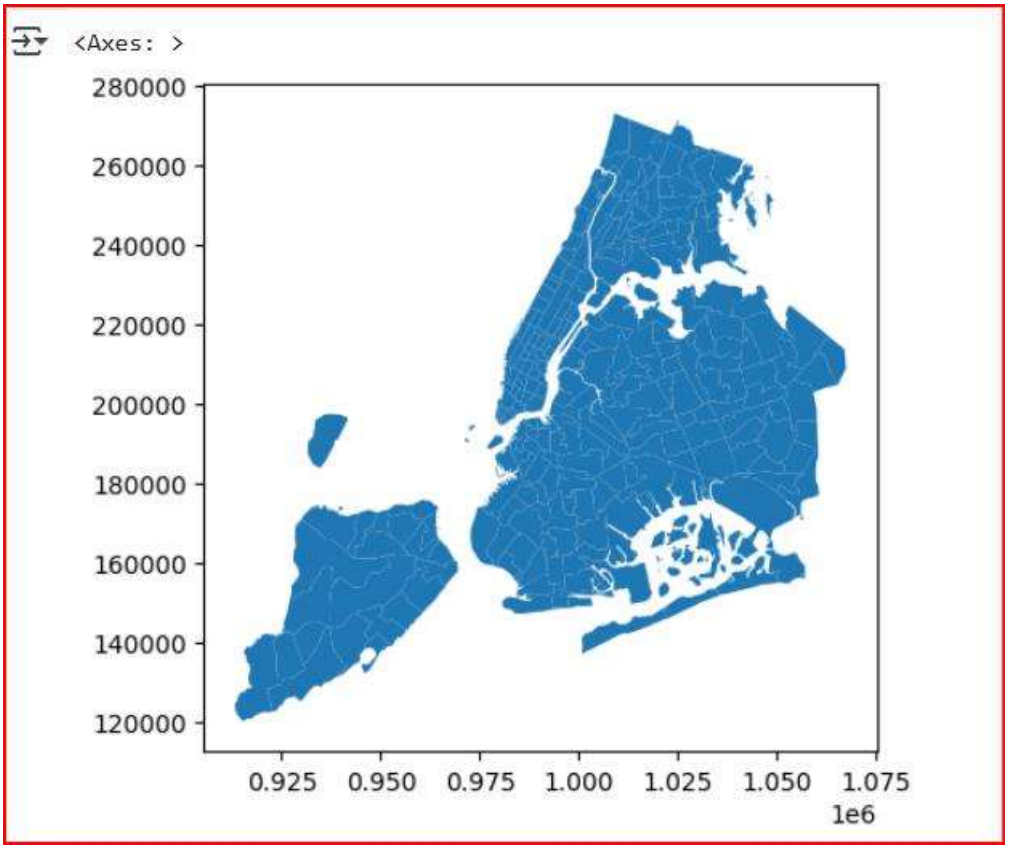
dtype: int64

	payment_type
count	1.414422e+06
mean	1.000059e+00
std	1.195036e-02
min	1.000000e+00
25%	1.000000e+00
50%	1.000000e+00
75%	1.000000e+00
max	4.000000e+00

dtype: float64

3.1.8. Load the taxi zones shapefile and display it

```
# Import necessary modules
import geopandas as gpd
# Read file using gpd.read_file()
gdf = gpd.read_file('taxi_zones.shp')
gdf.plot()
```




```
gdf.head()
```

gdf.head()

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

```
gdf.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
```

3.1.9. Merge the zone data with trips data

```
merged_df = pd.merge(df, gdf, left_on='PULocationID',
right_on='LocationID')
merged_df.head()
```

3.1.10. Find the number of trips for each zone/location ID

```
grouped_data = merged_df.groupby('PULocationID')
# Calculate the sum of 'PULocationID' for each category
sum_values = grouped_data['PULocationID'].sum()
print("\nSum of values by category:\n", sum_values)
# Applying multiple aggregations
aggregated_data = grouped_data.agg({'PULocationID': ['mean',
'sum', 'count']})
print("\nAggregated data:\n", aggregated_data)
```

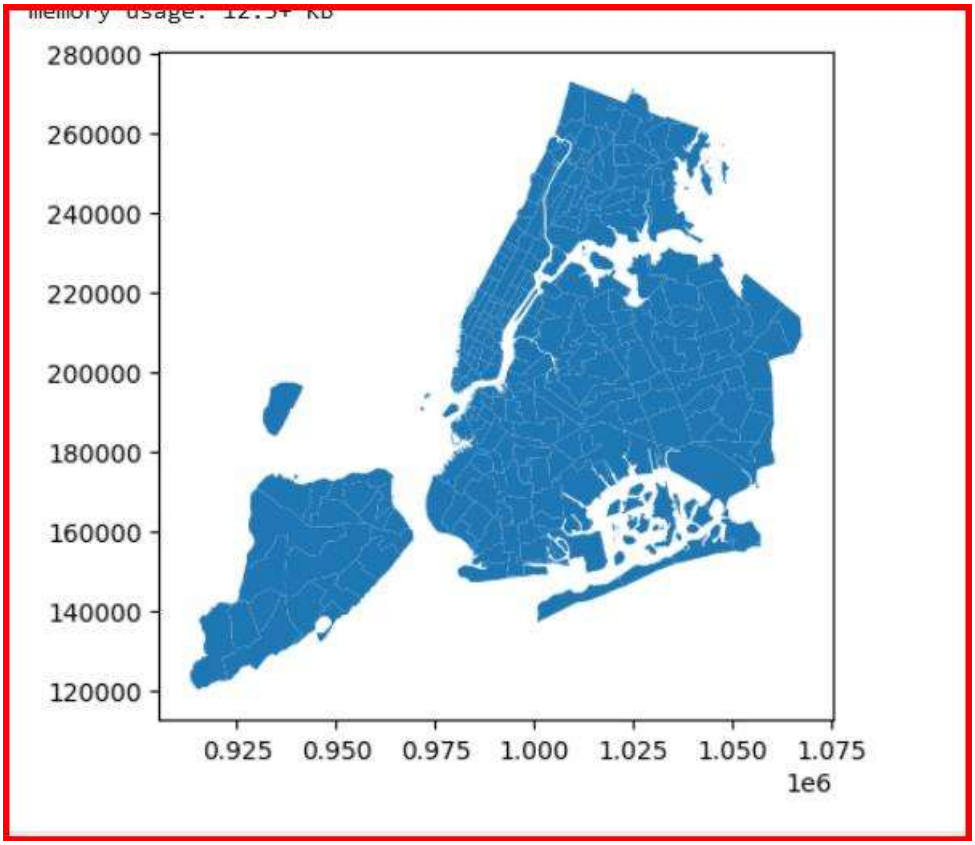
```
Sum of values by category:
 PULocationID
1           28
2            8
3            9
4         5844
7         1540
         ...
259        777
260      29640
261    1724166
262    5035640
263    7300617
Name: PULocationID, Length: 224, dtype: int64

Aggregated data:
              PULocationID
                      mean      sum  count
PULocationID
1                      1.0       28     28
2                      2.0        8      4
3                      3.0        9      3
4                      4.0     5844   1461
7                      7.0     1540    220
...                    ...      ...    ...
259                  259.0      777      3
260                  260.0    29640    114
261                  261.0  1724166   6606
262                  262.0  5035640  19220
263                  263.0  7300617  27759

[224 rows x 3 columns]
```

3.1.11. Add the number of trips for each zone to the zones dataframe

```
grouped_data = merged_df.groupby('zone')
print(grouped_data)
aggregated_data = grouped_data.agg({'zone': ['count']})
print("\nAggregated data:\n", aggregated_data)
```

```
Aggregated data:
                                    zone
                                   count
                        zone
                Allerton/Pelham Gardens      3
                Alphabet City             1461
                Astoria                    220
                Astoria Park                 7
                Auburndale                   3
                ...                        ...
                Woodlawn/Wakefield           3
                Woodside                   114
                World Trade Center        6606
                Yorkville East           19220
                Yorkville West           27759
```

Plot a map of the zones showing number of trips



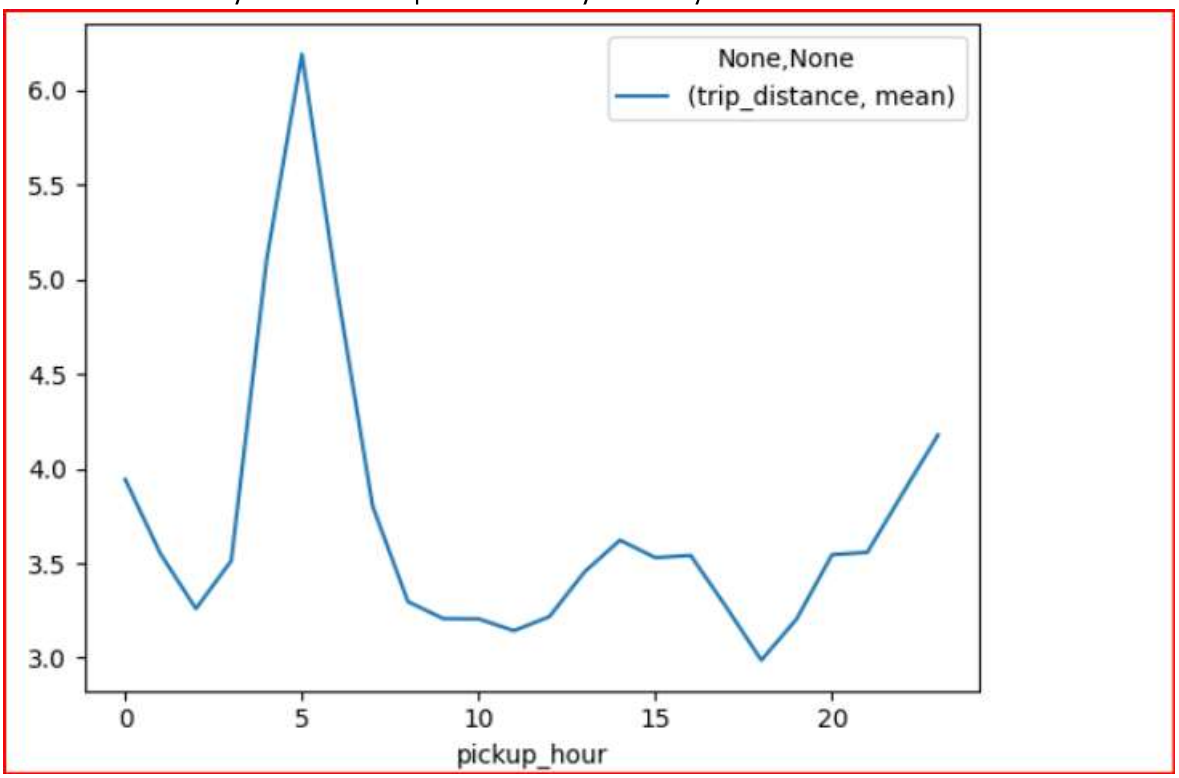
3.1.12. Conclude with results

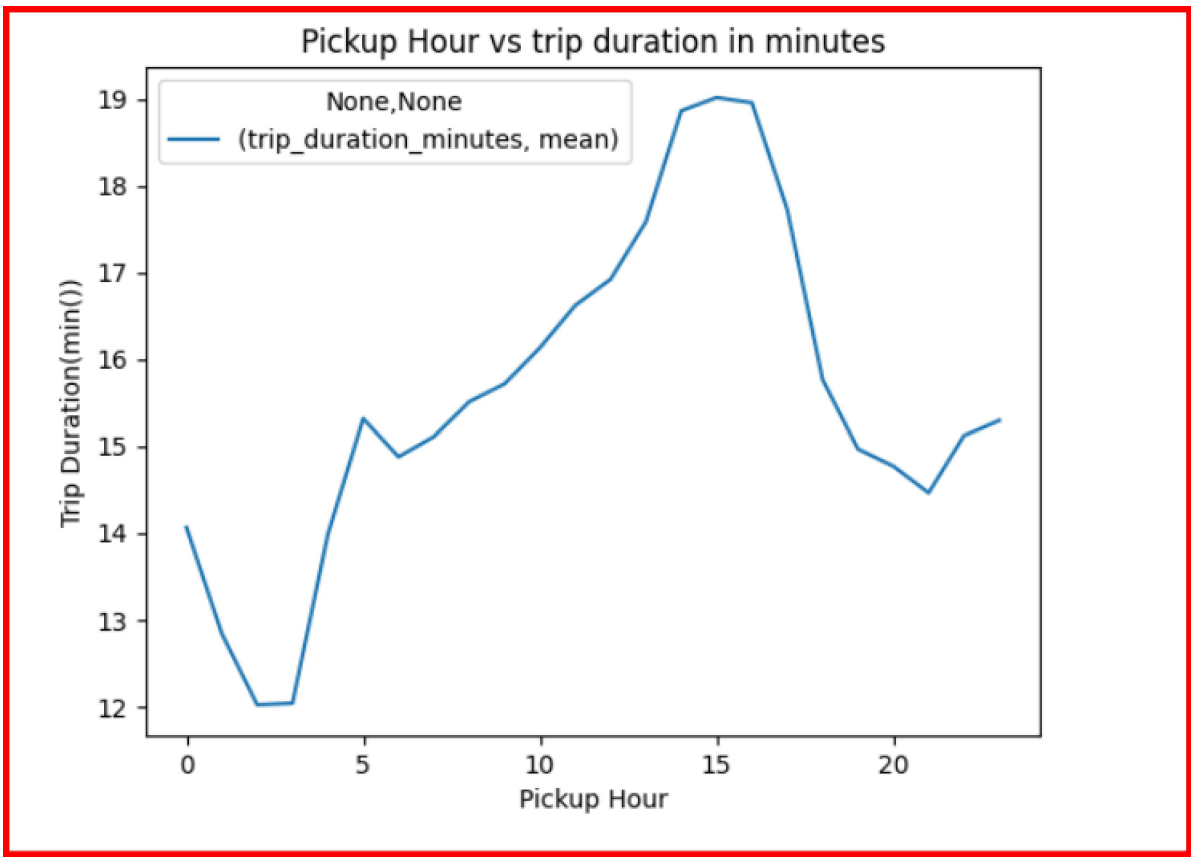
3.2. Detailed EDA: Insights and Strategies

Identify slow routes by comparing average speeds on different routes

pickup_hour	trip_duration_minutes	speed
	mean	
0	14.062299	16.821847
1	12.838178	16.582829
2	12.019479	16.272432
3	12.036857	17.511098
4	13.982085	21.895485
5	15.314818	24.254295
6	14.875103	19.931202
7	15.107110	15.098872
8	15.508413	12.753775
9	15.713393	12.245531
10	16.129457	11.924084
11	16.617494	11.348411
12	16.918162	11.409750
13	17.580852	11.789551
14	18.860145	11.521406
15	19.012393	11.138210
16	18.953147	11.209854
17	17.720478	11.071487
18	15.762744	11.371498
19	14.964605	12.852814
20	14.768493	14.401867
21	14.461220	14.760413
22	15.118595	15.356078
23	15.293776	16.390146

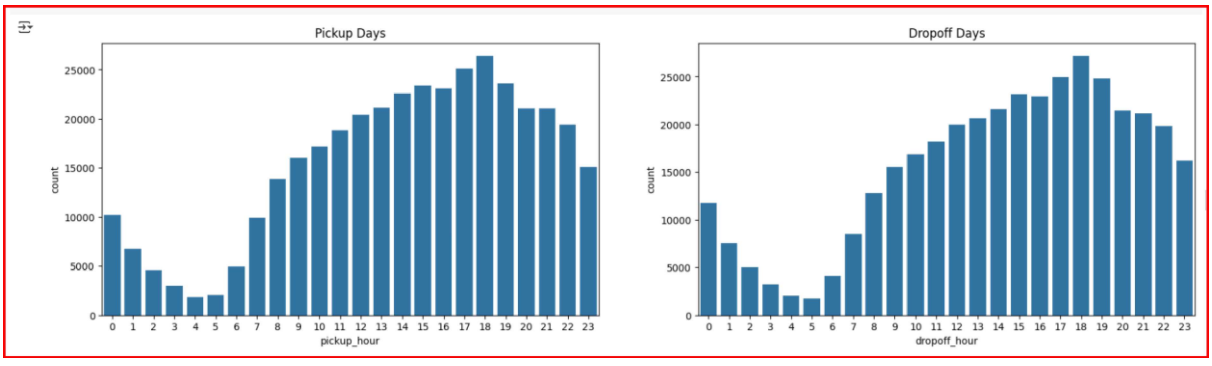
Calculate the hourly number of trips and identify the busy hours





3.2.1. Scale up the number of trips from above to find the actual number of trips

Compare hourly traffic on weekdays and weekends



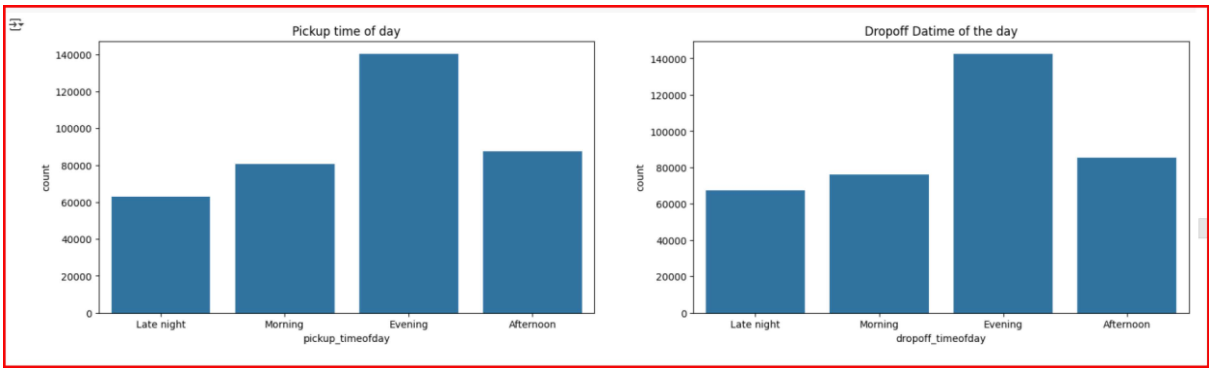
3.2.2. Identify the top 10 zones with high hourly pickups and drops

3.2.3. Find the ratio of pickups and dropoffs in each zone

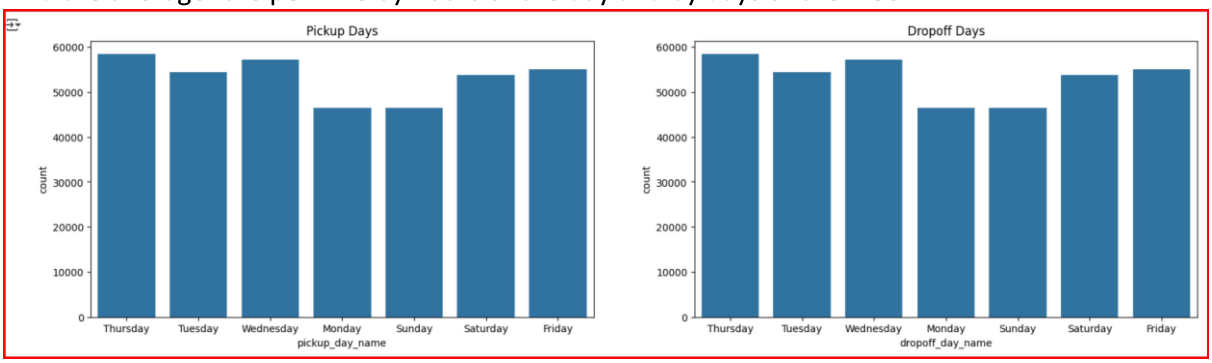
Identify the top zones with high traffic during night hours

3.2.4. Find the revenue share for nighttime and daytime hours

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



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

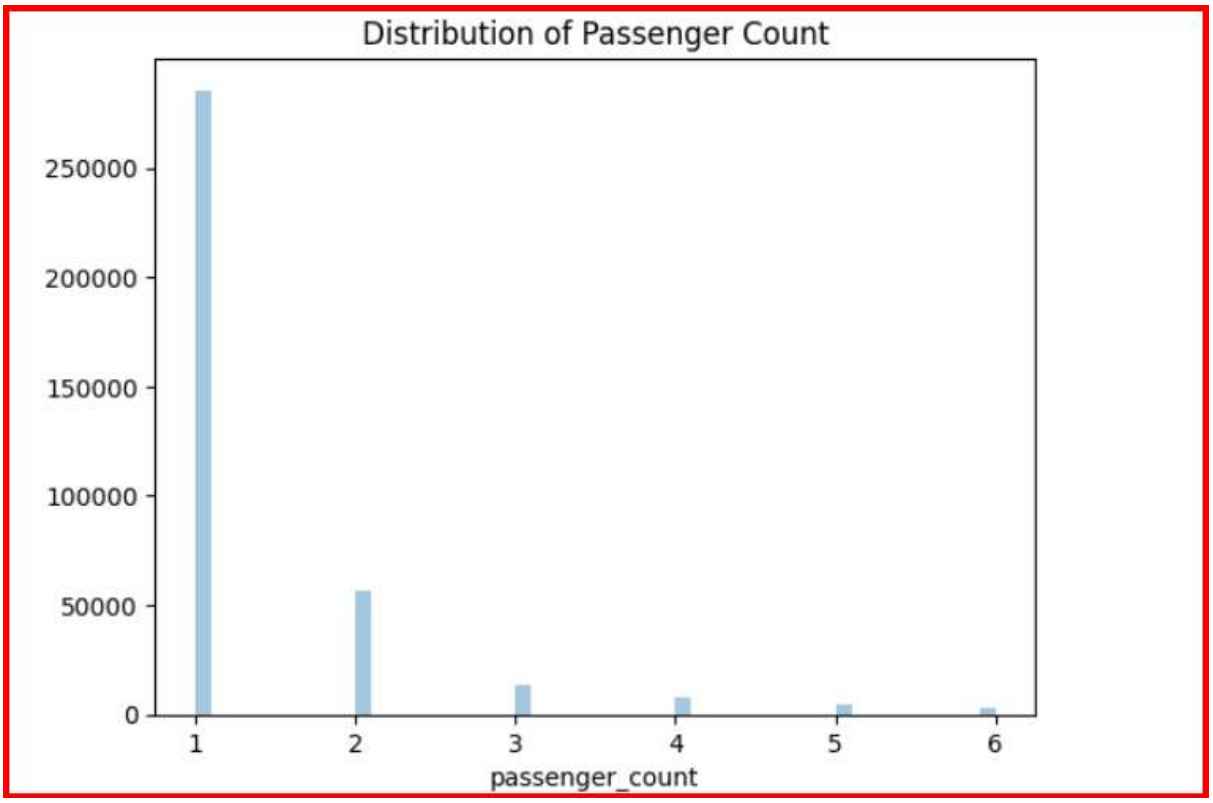


3.2.5. Analyse the average fare per mile for the different vendors

3.2.6. Compare the fare rates of different vendors in a distance-tiered fashion

3.2.7. Analyse the tip percentages

Analyse the trends in passenger count



3.2.8. Analyse the variation of passenger counts across zones

3.2.9. Analyse the pickup/dropoff zones or times when extra charges are applied more frequently.

4. Conclusions

4.1. Final Insights and Recommendations

4.1.1. Recommendations to optimize routing and dispatching based on demand patterns and operational inefficiencies.

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

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