Name -Neeraj Kumar Singh

1 Data Preparation:

Import Libraries

Import warnings

import warnings

warnings.filterwarnings (action = ‘ignore')

Import the libraries you will be using for analysis

import numpy as np

import pandas as pd

!pip install matplotlib

import matplotlib.pyplot as plt

import seaborn as sns

Check versions

Output -

numpy version: 1.26.4

pandas version: 2.2.2
matplotlib version: 3.10.0
seaborn version: 0.13.2

1.1 Load the dataset

Command:

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

df.info()

 # iterate through each file

```
for i in range(1,13,1):

    df = pd.read_parquet(f'2023-{i}.parquet')

    df.info()

    df.shape
```

1.1.1 Figure out how to sample and combine the files.

5% of the data from each months parquet file were randomly selected. The data was fetched from month file -> day -> hour: append sampled data -> move to next hour -> move to next day after 24 hours -> move to next month file

Command:

```
sampled_data = pd.DataFrame()

for i in range(1,13,1):

    df = pd.read_parquet(f'2023-{i}.parquet')

    dates = df['tpep_pickup_datetime'].dt.date.unique()

    for date in dates:

        df_data = df[df['tpep_pickup_datetime'].dt.date == date]

        for hour in range(24):

            # Filter data for this hour

            hour_data = df_data[df_data['tpep_pickup_datetime'].dt.hour == hour]

            # Only sample if hour_data is not empty

            if not hour_data.empty:

                # Sample 5% of the hour_data

                sampled = hour_data.sample(frac=0.05, random_state=42)

                # Append to the main sampled_data DataFrame

                sampled_data = pd.concat([sampled_data, sampled], ignore_index=True)

sampled_data
```

Output:

1. Dataframe with shape = 1896400 rows × 20 columns

2. Store the df in parquet:- *sampled_data.to_parquet('2023_sampled.parquet')*

2 Data Cleaning

Load the new data file

df1_0 = pd.read_parquet('2023_sampled.parquet')

df1_0

df.head()

df1_0.head()

df.info()

df1_0.info()

2.1 Fixing Columns

2.1.1 Fix the index and drop unnecessary columns

1. It will appropriate to change the data type for 'RatecodeID' and 'passenger_count' from 'Float' to 'Int64'.

df1_0['passenger_count'] = df1_0['passenger_count'].astype('Int64')

df1_0['RatecodeID'] = df1_0['RatecodeID'].astype('Int64')

Reset index

df1_0 = df1_0.reset_index(drop=True)

df1_0.info()

2.1.2 Combine the two airport fee columns

1. There are two airport fee columns. Upon analysis, found that -

1.1 if 'airport_fee' is not null then 'Airport_fee' is null and vice-a-versa.

1.2 we can combine both the columns for non-null values using combine_first() command.

1.2 Drop [‘airport_fee’,'Airport_fee'], as we created ‘Airport_Fee’ column using combine_first()

df1_0['Airport_Fee'] = df1_0['airport_fee'].combine_first(df1_0['Airport_fee'])

df1_0.drop(['airport_fee','Airport_fee'], axis = 1,inplace=True)

df1_0

2.1.3 Fix columns with negative (monetary) values

ANS - There are no negative values in 'fare_amount' column. check where values of fare amount are negative.

df1_neg_fare_amount = df1_0[df1_0.fare_amount < 0]

df1_neg_fare_amount

Output: we get a blank data frame

VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payme

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

Analyse RatecodeID for the negative fare amounts

ANS - There are no negative values in 'RatecodeID' column

df1_neg_RatecodeID = df1_0[df1_0.RatecodeID < 0]

df1_neg_RatecodeID

Output: we get a blank data frame

VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payme

 # Find which columns have negative values

ANS - Columns(n=counts) with negative values are - extra (n=3), mta_tax(n=73), improvement_surcharge(n=78), total_amount(n=78), congestion_surcharge(n=56), Airport_Fee(N=15)

print('Distribution of Negative Values in columns')

column_with_neg_values = (df1_0[['VendorID',

'passenger_count', 'trip_distance', 'RatecodeID',

'PULocationID', 'DOLocationID', 'payment_type', 'fare_amount', 'extra',

'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge',

'total_amount', 'congestion_surcharge', 'Airport_Fee']] < 0).sum()

column_with_neg_values

Output :

```
Distribution of Negative Values in columns
VendorID                  0
passenger_count           0
trip_distance             0
RatecodeID                0
PULocationID              0
DOLocationID              0
payment_type              0
fare_amount               0
extra                     3
mta_tax                  73
tip_amount                0
tolls_amount              0
improvement_surcharge    78
total_amount             78
congestion_surcharge     56
Airport_Fee              15
dtype: Int64
```

 # fix these negative values

Ans : Set negatives to zero (e.g., for features like extra, mta_tax, improvement_surcharge, total_amount, congestion_surcharge and Airport_Fee can't logically be negative)

Command:

df1_0.loc[df1_0['extra'] < 0, 'extra'] = 0.0

df1_0.loc[df1_0['mta_tax'] < 0, 'mta_tax'] = 0.0

df1_0.loc[df1_0['improvement_surcharge'] < 0, 'improvement_surcharge'] = 0.0

df1_0.loc[df1_0['total_amount'] < 0, 'total_amount'] = 0.0

df1_0.loc[df1_0['congestion_surcharge'] < 0, 'congestion_surcharge'] = 0.0

df1_0.loc[df1_0['Airport_Fee'] < 0, 'Airport_Fee'] = 0.0

Checking for negative values in columns post fixing it.

Command:

(df1_0[['VendorID',

'passenger_count', 'trip_distance', 'RatecodeID',

'PULocationID', 'DOLocationID', 'payment_type', 'fare_amount', 'extra',

'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge',

'total_amount', 'congestion_surcharge', 'Airport_Fee']] < 0).sum()

Output: There are no negatives left in any of the columns

```
VendorID                0
passenger_count         0
trip_distance           0
RatecodeID              0
PULocationID            0
DOLocationID            0
payment_type            0
fare_amount             0
extra                   0
mta_tax                 0
tip_amount              0
tolls_amount            0
improvement_surcharge   0
total_amount            0
congestion_surcharge    0
Airport_Fee             0
dtype: Int64
```

2.2 Handling Missing Values

2.2.1 Find the proportion of missing values in each column

Command:

print('Proportion of missing values is below:')

*round(df1_0.isnull().sum()/df1_0.shape[0]*100,2)*

Output:

```
Proportion of missing values is below:
VendorID                 0.00
tpep_pickup_datetime     0.00
tpep_dropoff_datetime    0.00
passenger_count          3.42
trip_distance            0.00
RatecodeID               3.42
store_and_fwd_flag       3.42
PULocationID             0.00
DOLocationID             0.00
payment_type             0.00
fare_amount              0.00
extra                    0.00
mta_tax                  0.00
tip_amount               0.00
tolls_amount             0.00
improvement_surcharge    0.00
total_amount             0.00
congestion_surcharge     3.42
Airport_Fee              3.42
dtype: float64
```

2.2.2 Handling missing values in passenger_count

ANS - Imputed the NaN values in 'passenger_count' with it's **Median value**

Command:

```
# Null passanger count
passenger_count_null = df1_0[df1_0.passenger_count.isnull()]
passenger_count_null
# Median value calculation
Median_passenger_count = df1_0.passenger_count.median()
# Replace Null values with Median values
df1_0['passenger_count'].fillna(Median_passenger_count, inplace = True)
df1_0
```

#Did you find zeroes in passenger_count? Handle these.

Ans: Dropping rows with passenger_count = 0

Command:

```
df1_1 = df1_0[~(df1_0.passenger_count == 0)]
df1_1
```

2.2.3 Handle missing values in RatecodeID

Ans - Since there is a RatecodeID for unknown/null values. Hence assigned value for 99 for all the null values in the RatecodeID.

Command:

```
df1_1.loc[df1_1.RatecodeID.isnull(),'RatecodeID' ] = 99
```

2.2.4 Impute NaN in congestion_surcharge

Ans: Impute NaN value in congestion_surcharge with Median value

Command:

```
median_congestion_surcharge = df1_1.congestion_surcharge.median()
df1_1.congestion_surcharge.fillna(median_congestion_surcharge, inplace = True)
df1_1
```

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

Ans: Yes there are two columns with missing values - Airport_Fee and store_and_fwd_flag

'''

Handle missing values in Airport_Fee columns -

Observation 1 - Prior to assigning 99 to Null values in 'RatecodeID', if we filter the RatecodeID = 99, we will find that airport_fee is 0.0 for throughout. This means for Null/Unknown in 'RatecodeID', the value in airport_fee is 0.0 since 'RatecodeID' = 99 stands for 'Null/unknown' values.

Observation 2 - While handling Null values in 'RatecodeID' we had assigned a value of 99 to it. The corresponding airport_fee is null for the same set of rows.

Solution for handling missing values in Airport_Fee - Extrapolating the Observation 1 onto Observation 2. We can assign 0.0 for all the 'RatecodeID' = 99.

'''

Command:

df1_1.Airport_Fee.fillna(0.0, inplace = True)

'''

Handle missing values in 'store_and_fwd_flag' columns -

Observation 1 - Prior to assigning 99 to Null values in 'RatecodeID', if we filter the RatecodeID = 99, we will find that mode()[0] of store_and_fwd_flag is 'N'.

Observation 2 - While handling Null values in 'RatecodeID' we had assigned a value of 99 to it. The corresponding store_and_fwd_flag is null for the same set of rows.

Solution for handling missing values in store_and_fwd_flag - Extrapolating the Observation 1 onto Observation 2. We can assign 'N' for all the 'RatecodeID' = 99.

'''

Command:

mode_store_and_fwd_flag = df1_1[df1_1.RatecodeID == 99].store_and_fwd_flag.mode()[0]

df1_1.loc[df1_1.RatecodeID == 99,'store_and_fwd_flag'] = mode_store_and_fwd_flag

Now, There is no null value left.

Command:

df1_1.isnull().sum()

Output:

```
VendorID                0
tpep_pickup_datetime    0
tpep_dropoff_datetime   0
passenger_count         0
trip_distance           0
RatecodeID              0
store_and_fwd_flag      0
PULocationID            0
DOLocationID            0
payment_type            0
fare_amount             0
extra                   0
mta_tax                 0
tip_amount              0
tolls_amount            0
improvement_surcharge   0
total_amount            0
congestion_surcharge    0
Airport_Fee             0
dtype: int64
```

2.3 Handling Outliers

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

Command:

Describe the data and check if there are any potential outliers present

df1_1.describe()

Check for potential out of place values in various columns

columns = ['passenger_count', 'trip_distance','fare_amount', 'extra','mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge','total_amount', 'congestion_surcharge', 'Airport_Fee']

outlier_summary = {}

for col in columns:

Q1 = df1_1[col].quantile(0.25)

Q3 = df1_1[col].quantile(0.75)

lower_bound = Q1 - 1.5(Q3-Q1)*

upper_bound = Q3 + 1.5(Q3+Q1)*

outlier_count = df1_1[(df1_1[col] < lower_bound) | (df1_1[col] > upper_bound)].shape[0]

outlier_summary[col] = {'Lower_Bound' : lower_bound,'Upper_Bound':upper_bound, 'Outlier_count':outlier_count }

#outlier_summary

df1_1_outlier = pd.DataFrame(outlier_summary).T

df1_1_outlier

Output:

Outlier_analysis

	Lower_Bound	Upper_Bound	Outlier_count
passenger_count	1.0	4.0	39752.0
trip_distance	-2.49	10.1	158712.0
fare_amount	-10.170000000000000	69.65	96566.0
extra	-3.75	6.25	32941.0
mta_tax	0.5	2.0	17598.0
tip_amount	-4.175000000000000	12.625	83470.0
tolls_amount	0.0	0.0	152213.0
improvement_surcharge	1.0	4.0	2195.0
total_amount	-6.600000000000000	101.44	24529.0
congestion_surcharge	2.5	10.0	138790.0
Airport_Fee	0.0	0.0	159254.0

2.3.1 Based on the above analysis, it seems that some of the outliers are present due to errors in registering the trips. Fix the outliers.

Outlier handling 1 - First, let us remove 7+ passenger counts as there are very less instances.

Command:

df1_2 = df1_1[~(df1_1.passenger_count > 6)]

Outlier handling 2 - Removing entries where trip_distance is nearly 0 and fare_amount is more than upper bound (69.650) from above table

Command:

df1_3 = df1_2[~((df1_2.trip_distance == 0) & (df1_2.fare_amount > 69.650))]

Outlier handling 3 - Removing entries where trip_distance and fare_amount are 0 but the pickup and dropoff zones are different (both distance and fare should not be zero for different zones)

Command:

df1_4 = df1_3[~((df1_3.trip_distance == 0) & (df1_3.fare_amount == 0) & (df1_3.PULocationID != df1_3.DOLocationID))]

Outlier handling 4 - Removing entries where trip_distance is more than 250 miles.

Command:

df1_5 = df1_4[~(df1_4.trip_distance > 250)]

Outlier handling 5 - Removing entries where pickup_datetime and dropoff_datetime is same but PULocationID and DOLocationID are different

Command:

df1_6 = df1_5[~((df1_5.tpep_pickup_datetime == df1_5.tpep_dropoff_datetime) & (df1_5.PULocationID!= df1_5.DOLocationID))]

Do any columns need standardising?

Standardisation 1 - VendorID should be 1, 2 as per data dictionary. Hence, removed VendorId other than 1,2

Command:

df1_7 = df1_6[df1_6.VendorID.isin([1,2])]

Standardisation 2 - RatecodeID should be 1,2,3,4,5,6 as per data dictionary. Hence, remove any other values

Command:

df1_8 = df1_7[df1_7.RatecodeID.isin([1,2,3,4,5,6])]

Standardisation 3 - payment_type should be 1,2,3,4,5,6 as per data dicstionary. Hence, remove any other values

df1_9 = df1_8[df1_8.payment_type.isin([1,2,3,4,5,6])].reset_index(drop=True)

3 Exploratory Data Analysis

3.1 General EDA: Finding Patterns and Trends

3.1.1 Categorise the variables into Numerical or Categorical.

Ans: The parameters are mapped below

- VendorID: Categorical
- tpep_pickup_datetime: datetime (Can be considered as Numerical)
- tpep_dropoff_datetime: datetime (Can be considered as Numerical)
- passenger_count: Numerical
- trip_distance: Numerical
- RatecodeID: Categorical
- PULocationID: Categorical
- DOLocationID: Categorical

- payment_type: Categorical
- pickup_hour: Categorical
- trip_duration: Numerical

The following monetary parameters belong in the same category, is it categorical or numerical?

Ans - These parameters are numerical values

- fare_amount
- extra
- mta_tax
- tip_amount
- tolls_amount
- improvement_surcharge
- total_amount
- congestion_surcharge
- airport_fee

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

 # Find and show the hourly trends in taxi pickups

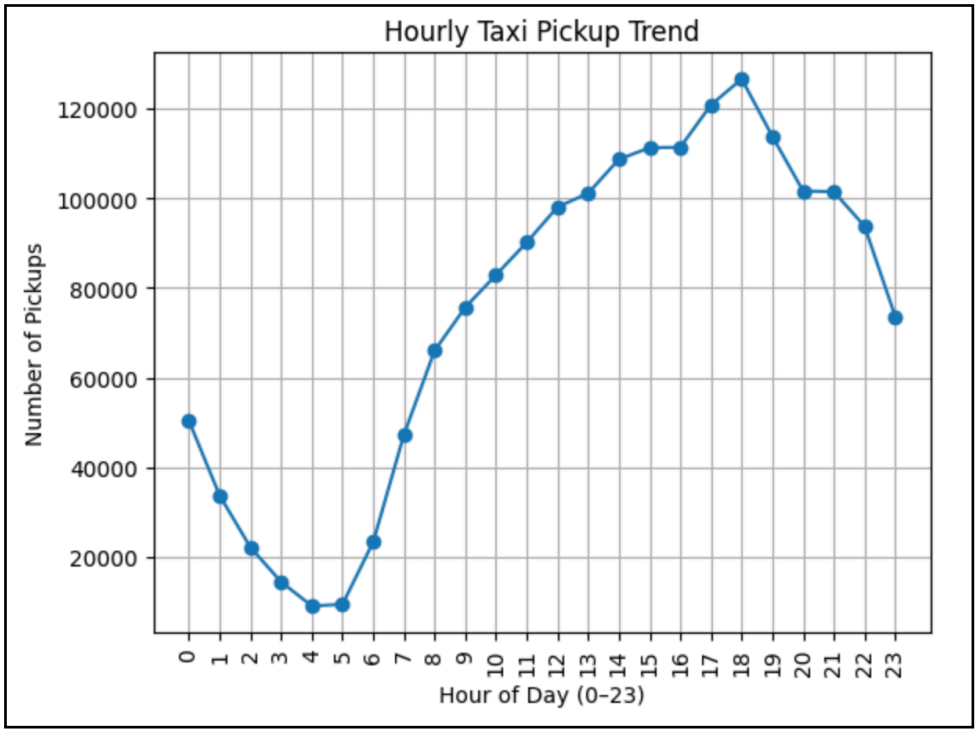
Ans

Command:

```
hourly_pickup_counts = df1_9['tpep_pickup_datetime'].dt.hour.value_counts().reset_index()

hourly_pickup_counts.rename(columns={'tpep_pickup_datetime':'Hourly_Pickup','count':'counts'}, inplace = True)

hourly_pickup_counts.sort_values('Hourly_Pickup', inplace = True)

hourly_pickup_counts


plt.plot(hourly_pickup_counts.Hourly_Pickup, hourly_pickup_counts.counts, marker='o')

plt.title('Hourly Taxi Pickup Trend')

plt.xlabel('Hour of Day (0–23)')

plt.ylabel('Number of Pickups')

plt.xticks(range(0, 24))

plt.xticks(rotation = 90)

plt.grid(True)

plt.show()
```

Output: There is lesser pickups in the morning hours and as the day progresses the pickup counts increases with peak at 6:00 pm in the evening and then gradually falls down as night hour sets in. This makes a complete sense, which say that people may be travelling at different time but the return hours peak at 6:00pm.

Hourly_Pickup	counts
0	50350
1	33718
2	22156
3	14452
4	9079
5	9439
6	23335
7	47268
8	66021
9	75700
10	82891
11	90166
12	98037
13	101116
14	108681
15	111274
16	111364
17	120825
18	126598
19	113793
20	101616
21	101462
22	93776
23	73656



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

Ans:

Command -

DayOfWeek_pickup_counts = df1_9['tpep_pickup_datetime'].dt.day_name().value_counts().reindex(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

print('Daily trends in taxi pickups (days of the week)',DayOfWeek_pickup_counts,'\n\n')

plt.plot(DayOfWeek_pickup_counts.index, DayOfWeek_pickup_counts.values, marker='o')

plt.title('Day of the week - Taxi Pickup Trend')

plt.xlabel('Day of the week')

plt.ylabel('Number of Pickups')

plt.xticks(rotation = 45)

plt.ylim(0, 300000)

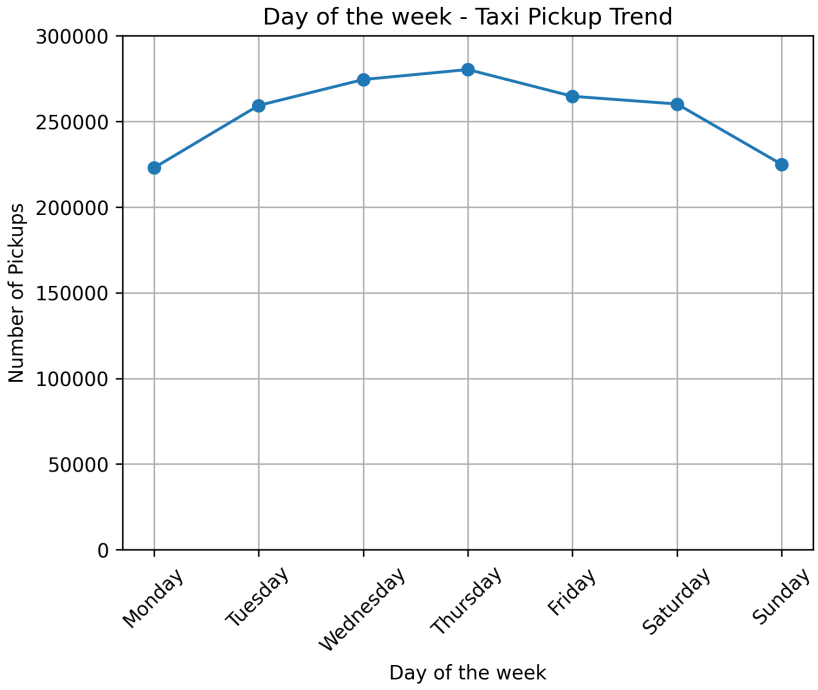
plt.grid(True)

plt.savefig("DayOfWeek_pickup_counts.png", bbox_inches='tight', dpi=300)

plt.show()

Output: Day of week analysis for pickup number show that it gradually increases from Monday and peaks on Thursday and gradually come down towards the end of the week.

```
Daily trends in taxi pickups (days of the week)
Monday       222904
Tuesday      259327
Wednesday    274462
Thursday     280318
Friday       264680
Saturday     260178
Sunday       224904
Name: count, dtype: int64
```



 # Show the monthly trends in pickups

Ans:

Command:

monthly_pickup_counts = df1_9['tpep_pickup_datetime'].dt.month.value_counts().sort_index()

print('Monthly trends in taxi pickups',monthly_pickup_counts,'\n\n')

plt.plot(monthly_pickup_counts.index, monthly_pickup_counts.values, marker='o')

plt.title('Monthly Taxi Pickup Trend')

plt.xlabel('Month of year (1-12)')

plt.ylabel('Number of Pickups')

plt.xticks(range(0, 13))

plt.ylim(0, 200000)

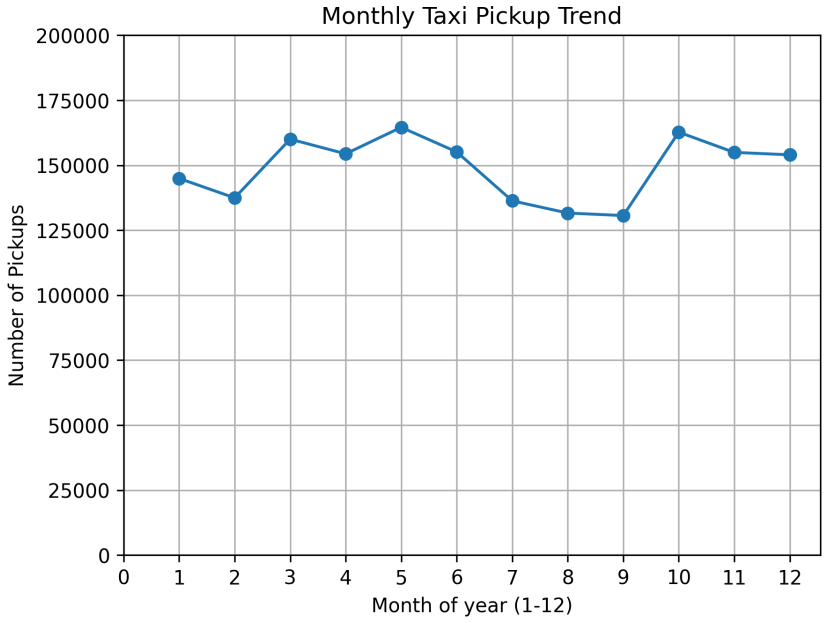
plt.grid(True)

plt.savefig("monthly_pickup_counts.png", bbox_inches='tight', dpi=300)

plt.show()

Output*: Peak months are March, April, May, October, November, December. There is dipping pickup after May till September and December to February.*

```
Monthly trends in taxi pickups
1     144927
2     137447
3     160038
4     154394
5     164592
6     155126
7     136310
8     131607
9     130642
10    162729
11    154942
12    154019
Name: count, dtype: int64
```



Take a look at the financial parameters like fare_amount, tip_amount, total_amount, and also trip_distance. Do these contain zero/negative values?

Commands:

*((df1_9[['fare_amount', 'tip_amount', 'total_amount','trip_distance']] <= 0).sum()/ df1_9.shape[0])*100*

Output: Below are the Percentage of values in columns with Zero value. There are no negative values.

```
fare_amount       0.029159
tip_amount       21.810213
total_amount      0.016678
trip_distance     0.918919
dtype: float64
```

3.1.3 Filter out the zero values from the above columns.

Command:

 # Removing tip_amount containing zero. There are no negative values.

df1_10 = df1_9[~(df1_9.tip_amount <= 0)]

 # Removing fare_amount containing zero. There are no negative values

df1_11 = df1_10[~(df1_10.fare_amount <= 0)]

 # Removing the distance = 0 in cases where pickup and drop is in the different zone

df1_12 = df1_11[~((df1_11.trip_distance <= 0) & (df1_11.PULocationID != df1_11.DOLocationID))]

 # Checking for zero's

*((df1_12[['fare_amount', 'tip_amount', 'total_amount','trip_distance']] <= 0).sum()/ df1_12.shape[0])*100*

Output: The trip_distance contains zero if the pick and drop location is same.

```
fare_amount       0.000000
tip_amount        0.000000
total_amount      0.000000
trip_distance     0.238422
dtype: float64
```

3.1.4 Analyse the monthly revenue (total_amount) trend

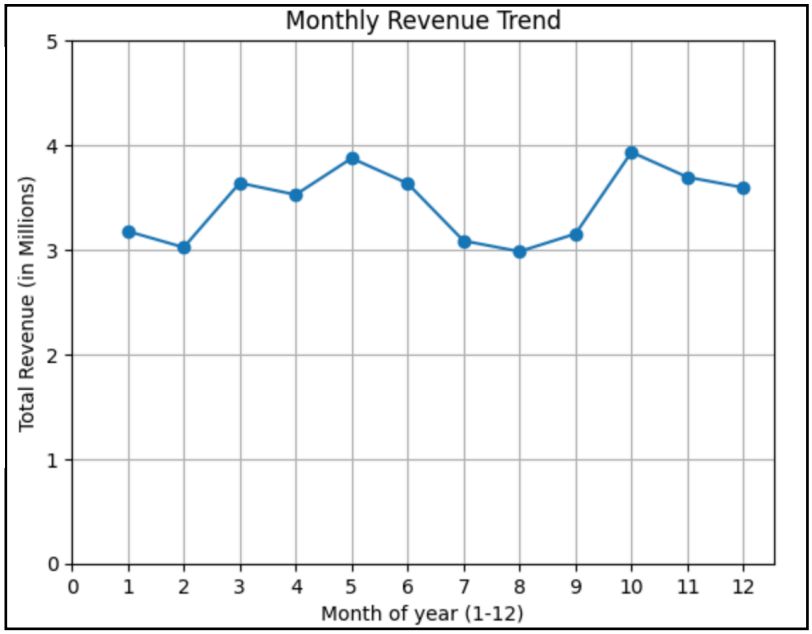
Ans:

Command:

```
# Group data by month and analyse monthly revenue
df1_12['Months'] = df1_12['tpep_pickup_datetime'].dt.month
Monthly_revenue = df1_12.groupby(['Months'])[['total_amount']].agg('sum')
print('Monthly revenue (total_amount) ----> ',Monthly_revenue,'\n\n\n')
plt.plot(Monthly_revenue.index, Monthly_revenue.values/1000000, marker='o')
plt.title('Monthly Revenue Trend')
plt.xlabel('Month of year (1-12)')
plt.ylabel('Total Revenue (in Millions)')
plt.xticks(range(0, 13))
plt.ylim(0, 5)
plt.grid(True)
plt.savefig("Monthly_revenue.png", bbox_inches='tight', dpi=300)
plt.show()
```

Output: Monthly Revenue follows the trend of Monthly pickups as revenues is directly proportional to pickups/rides. There is dipping trend in revenue after May till August and December to February. The peak of months in terms of revenue are in following order(low to high) - April, December, March, November, May and October.

```
Monthly revenue (total_amount) ----->
Months
1          3176659.30
2          3022834.73
3          3638147.39
4          3526293.47
5          3875052.03
6          3634389.16
7          3085302.84
8          2983359.57
9          3152203.04
10         3932682.02
11         3694035.05
12         3594431.35
```



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

Command:

```
# Calculate proportion of each quarter

df1_12['Quarter'] = df1_12['tpep_pickup_datetime'].dt.quarter

#print('Proportion of revenue per quarter ----> ')

quarterly_revenue = df1_12.groupby(['Quarter'])[['total_amount']].agg('sum')

quarterly_revenue['Proportion (%)'] = round((quarterly_revenue['total_amount']/
quarterly_revenue['total_amount'].sum())*100,2)

quarterly_revenue
```

Output: Order of quarterly revenue is Q4 > Q2 > Q1 > Q3

Quarter	total_amount	Proportion (%)
1	9837641.42	23.81
2	11035734.66	26.71
3	9220865.45	22.32
4	11221148.42	27.16

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

Ans:-

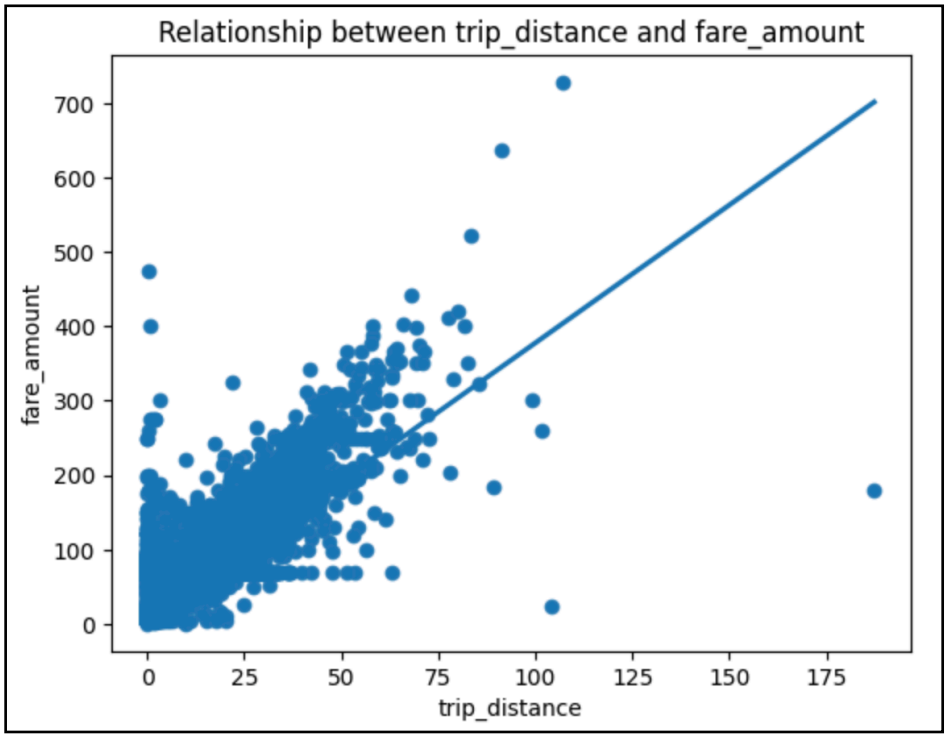
Command:

```
## Dropping rows with trip_distance = 0

df1_13 = df1_12[~(df1_12.trip_distance == 0)]

sns.scatterplot(x = df1_13['trip_distance'], y = df1_13['fare_amount'])

sns.regplot(x = df1_13['trip_distance'], y = df1_13['fare_amount'])

plt.title('Relationship between trip_distance and fare_amount')

plt.show()
```

correlation = df1_13['trip_distance'].corr(df1_13['fare_amount'])

print(f'Correlation between trip_distance and fare_amount: {correlation:.2f}\n\n')

Output: Correlation between trip_distance and fare_amount: 0.95.There is strong positive relationship between fare_amount and trip_distance.



3.1.7 Find and visualise the correlation between:

1. **fare_amount and trip duration (pickup time to dropoff time)**

Command:

df1_13['tpep_dropoff_datetime'] = pd.to_datetime(df1_13['tpep_dropoff_datetime'], format='%Y-%m-%d %H:%M:%S')
df1_13['tpep_pickup_datetime'] = pd.to_datetime(df1_13['tpep_pickup_datetime'], format='%Y-%m-%d %H:%M:%S')
df1_13['trip_duration'] = (df1_13['tpep_dropoff_datetime'] - df1_13['tpep_pickup_datetime'])

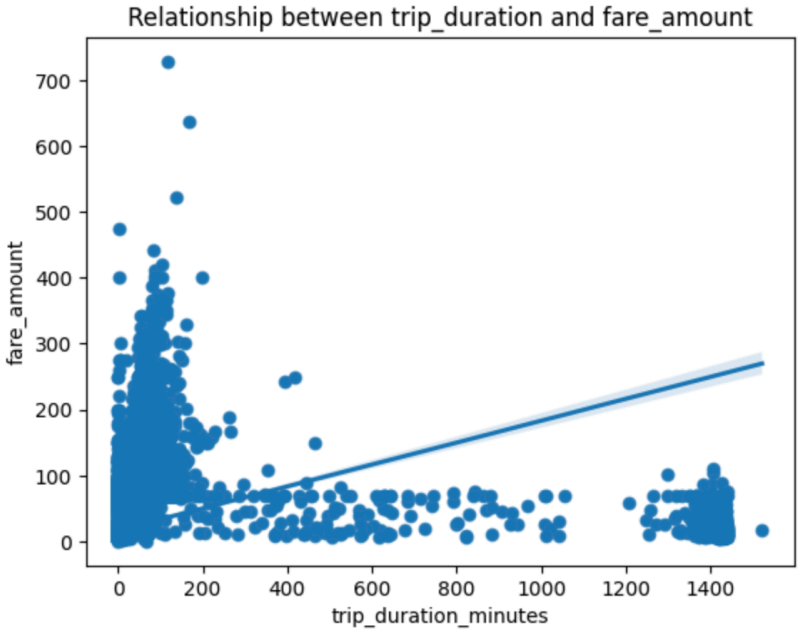
df1_13['trip_duration_minutes'] = df1_13['trip_duration'].dt.total_seconds()/60
np.result_type(np.dtype(df1_13['trip_duration_minutes']), np.complex128)

 # droping duration with negative values

df1_13 = df1_13[~(df1_13.trip_duration_minutes < 0)]

```
sns.scatterplot(x = df1_13['trip_duration_minutes'], y = df1_13['fare_amount'])
sns.regplot(x = df1_13['trip_duration_minutes'], y = df1_13['fare_amount'])
plt.show()

correlation = df1_13['trip_duration_minutes'].corr(df1_13['fare_amount'])
print(f'Correlation between trip_duration_minutes and fare_amount: {correlation:.2f}')
```

Output: Correlation between trip_duration and fare_amount: 0.33. There is weak positive relationship between fare_amount and trip duration

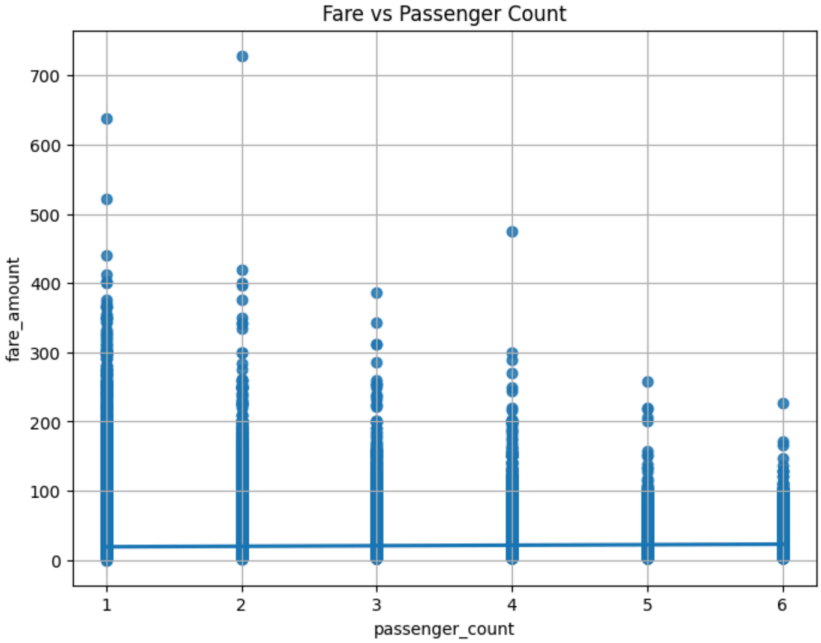


2. **fare_amount and passenger_count**

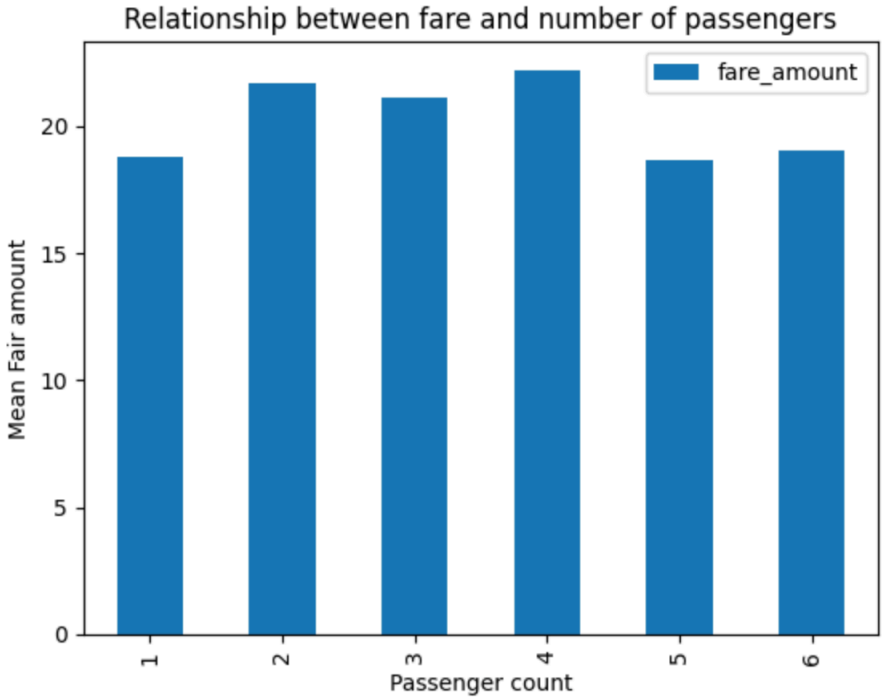
Command:

```
plt.figure(figsize=(8, 6))
plt.scatter(df1_13['passenger_count'], df1_13['fare_amount'], alpha=0.3)
plt.title('Fare vs Passenger Count')
plt.grid(True)
plt.show()

correlation_3 = df1_13['passenger_count'].corr(df1_13['fare_amount'])
print(f'Correlation between trip_duration and fare_amount: {correlation_3:.2f}\n\n')

## Scatter plot does not give a proper idea. hence ploting the mean fare value per passenger_counts.
df1_13.groupby(['passenger_count'])[['fare_amount']].agg('mean').plot.bar()
plt.title('Relationship between fare and number of passengers')
plt.xlabel('Passenger count')
plt.ylabel('Mean Fair amount')
plt.show()
```

Output: Correlation between trip_duration and fare_amount: 0.04. There is no relationship between fare and number of passengers



Scatter plot does not give a proper idea on correlation. Hence plotting the mean fare value per passenger_counts.



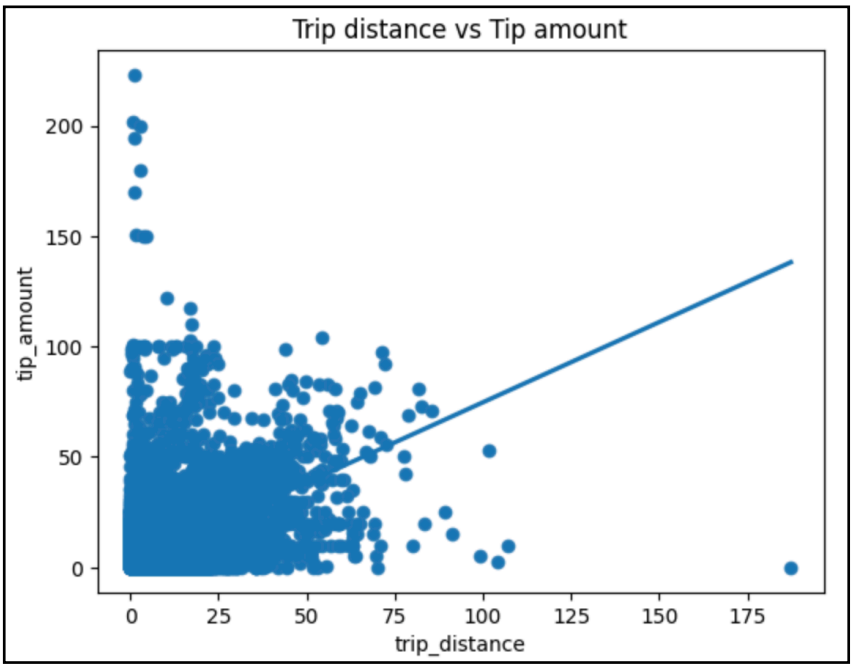
Conclusion - There is no relationship between fare and number of passengers

3. **tip_amount and trip_distance**

Command:

```
sns.scatterplot(x = df1_13['trip_distance'], y = df1_13['tip_amount'])
sns.regplot(x = df1_13['trip_distance'], y = df1_13['tip_amount'])
plt.title('Trip distance vs Tip amount')
plt.show()

correlation_4 = df1_13['trip_distance'].corr(df1_13['tip_amount'])
print(f'Correlation between trip_distance and tip_amount: {correlation_4:.2f}\n\n')
```

Output: Correlation between trip_distance and tip_amount: 0.80. There is strong positive relationship between tip_amount and trip_distance



3.1.8 Analyse the distribution of different payment types (payment_type)

Command:

```
payment_ditribution = pd.DataFrame
df1_13['payment_type'].value_counts()
df_counts = df1_13['payment_type'].value_counts().reset_index()
# Rename columns (optional but helpful)
df_counts.columns = ['payment_type', 'count']
df_counts
```

Output: Almost payment is through Credit card (payment_type = 1)

	payment_type	count
0	1	1391187
1	2	21
2	4	16
3	3	6

3.1.9 Load the shapefile and display it.

Command:

import geopandas as gpd

 # Read the shapefile using geopandas

#zones = # read the .shp file using gpd

zones =gpd.read_file('/Users/neerajkumarsingh/EDA_Assignment/taxi_zones/taxi_zones.shp')

zones.head()

Output:

	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...

Command:

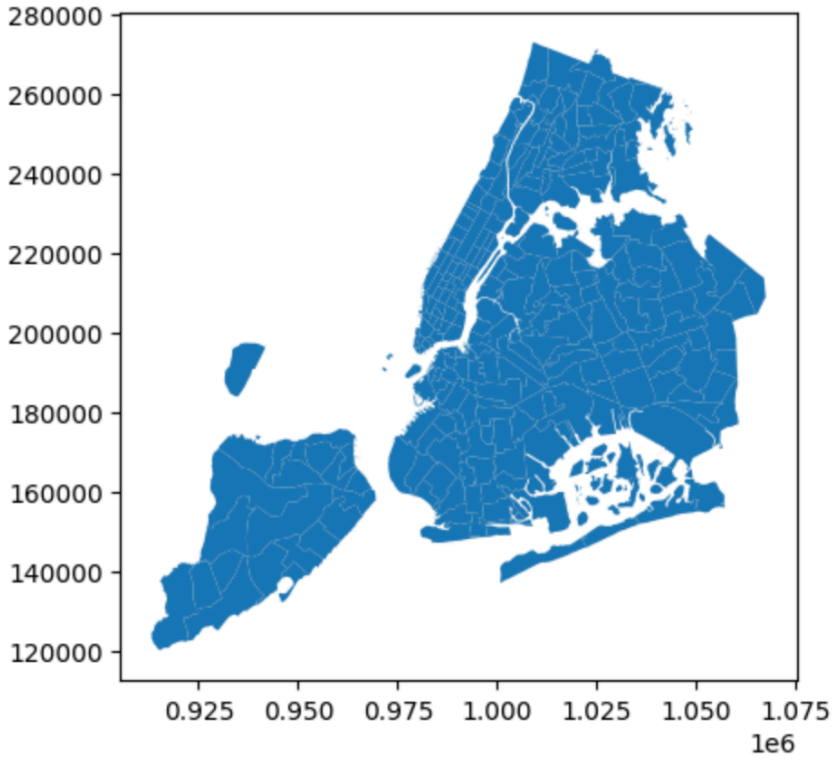
print(zones.info())

zones.plot()

Output:

```
<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
```

```
<Axes: >
```



3.1.10 Merge the zones data into trip data using the locationID and PULocationID columns.

Command:

 # Merge zones and trip records using locationID and PULocationID

pd.set_option('display.max_columns', 35)

df1_14 = pd.merge(df1_13,zones, how='inner', left_on = 'PULocationID', right_on = 'LocationID')

df1_14

Output:

Merged Schema with zone information. Shape - 1378974 rows × 30 columns

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

Command:

Did Value count per location on the data frame. This will give same result as grouping the data.

Trip_per_locationId = df1_14.LocationID.value_counts().reset_index()

Trip_per_locationId.columns = ['LocationID','trip_count']

Trip_per_locationId

Output: Dataframe with number of trips(trip_count) per LocationID

	LocationID	trip_count
0	237	68714
1	161	66652
2	132	63064
3	236	62557
4	162	51884
...	...	...
218	227	1
219	251	1
220	81	1
221	46	1
222	11	1

223 rows × 2 columns

3.1.12 Now, use the grouped data to add number of trips to the GeoDataFrame.

Command:

 # Merge trip counts back to the zones GeoDataFrame

zones['LocationID'] = zones['LocationID'].astype(int)

zones_with_trip_count = pd.merge(zones,Trip_per_locationId, how='left', on = 'LocationID')

zones_with_trip_count['trip_count'].fillna(0, inplace = True)

zones_with_trip_count

Output: Merged the number of trip(trip_count) with zones. Zones data frame was kept intact. If there were no trip_count for Zone, added a ‘0’ value for them using fillna() command.

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry	trip_count
0	1	0.116357	0.000782	Newark Airport	1	EWR	POLYGON ((933100.918 192536.086, 933091.011 19...	35.0
1	2	0.433470	0.004866	Jamaica Bay	2	Queens	MULTIPOLYGON (((1033269.244 172126.008, 103343...	2.0
2	3	0.084341	0.000314	Allerton/Pelham Gardens	3	Bronx	POLYGON ((1026308.77 256767.698, 1026495.593 2...	0.0
3	4	0.043567	0.000112	Alphabet City	4	Manhattan	POLYGON ((992073.467 203714.076, 992068.667 20...	1379.0
4	5	0.092146	0.000498	Arden Heights	5	Staten Island	POLYGON ((935843.31 144283.336, 936046.565 144...	0.0
...	...	...	...	...	...	...	...	...
258	259	0.126750	0.000395	Woodlawn/Wakefield	259	Bronx	POLYGON ((1025414.782 270986.139, 1025138.624 ...	0.0
259	260	0.133514	0.000422	Woodside	260	Queens	POLYGON ((1011466.966 216463.005, 1011545.889 ...	103.0
260	261	0.027120	0.000034	World Trade Center	261	Manhattan	POLYGON ((980555.204 196138.486, 980570.792 19...	6831.0
261	262	0.049064	0.000122	Yorkville East	262	Manhattan	MULTIPOLYGON (((999804.795 224498.527, 999824....	19026.0
262	263	0.037017	0.000066	Yorkville West	263	Manhattan	POLYGON ((997493.323 220912.386, 997355.264 22...	27471.0

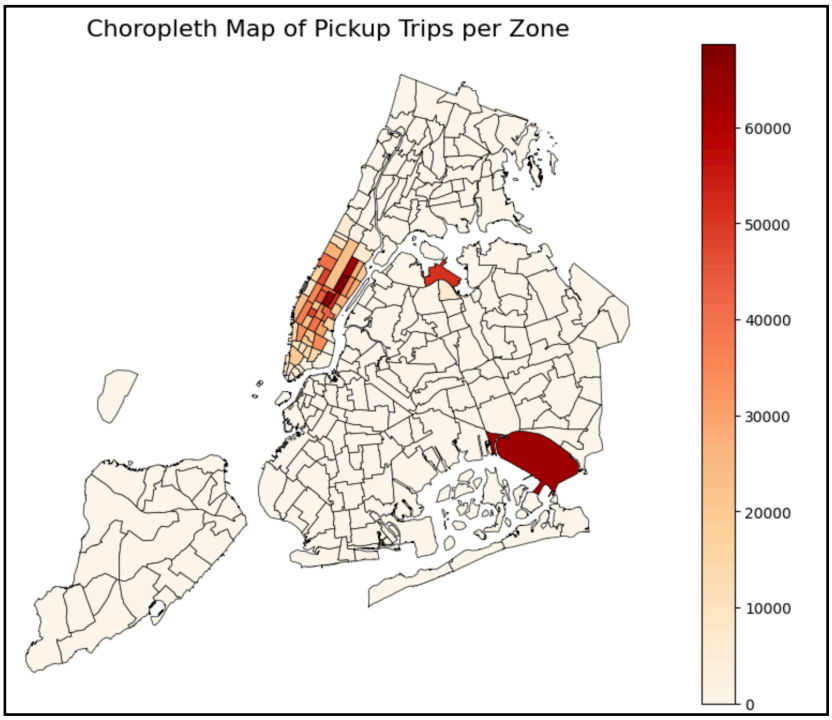
263 rows × 8 columns

3.1.13 Plot a color-coded map showing zone-wise trips

Command:

```
fig, ax = plt.subplots(1,1,figsize=(12, 10))
zones_with_trip_count.plot(column='trip_count',
    cmap='OrRd',        # Color scale: Orange-Red
    linewidth=0.5,
    edgecolor='black',
    legend=True,
    ax=ax)
ax.set_title('Choropleth Map of Pickup Trips per Zone', fontsize=16)
ax.axis('off')
plt.show()
```

Output:



 # can you try displaying the zones DF sorted by the number of trips?

Command:

zones_with_trip_count_sorted = zones_with_trip_count.sort_values(['trip_count'], ascending = False)

zones_with_trip_count_sorted

Output: Zones DF sorted by the number of trips (trip_count) in descending order.

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry	trip_count
236	237	0.042213	0.000096	Upper East Side South	237	Manhattan	POLYGON ((993633.442 216961.016, 993507.232 21...	68714.0
160	161	0.035804	0.000072	Midtown Center	161	Manhattan	POLYGON ((991081.026 214453.698, 990952.644 21...	66652.0
131	132	0.245479	0.002038	JFK Airport	132	Queens	MULTIPOLYGON (((1032791.001 181085.006, 103283...	63064.0
235	236	0.044252	0.000103	Upper East Side North	236	Manhattan	POLYGON ((995940.048 221122.92, 995812.322 220...	62557.0
161	162	0.035270	0.000048	Midtown East	162	Manhattan	POLYGON ((992224.354 214415.293, 992096.999 21...	51884.0
...	...	...	...	...	...	...	...	...
148	149	0.083681	0.000271	Madison	149	Brooklyn	POLYGON ((999782.783 162246.843, 999853.961 16...	0.0
83	84	0.233624	0.002074	Eltingville/Annadale/Prince's Bay	84	Staten Island	POLYGON ((939754.454 131548.91, 939802.804 131...	0.0
146	147	0.058765	0.000106	Longwood	147	Bronx	POLYGON ((1013266.371 240896.958, 1013354.947 ...	0.0
85	86	0.134245	0.000623	Far Rockaway	86	Queens	POLYGON ((1049025.346 163141.969, 1049041.462 ...	0.0
104	105	0.077425	0.000369	Governor's Island/Ellis Island/Liberty Island	103	Manhattan	POLYGON ((979605.759 191880.575, 979978.435 19...	0.0

263 rows × 8 columns

3.2 Detailed EDA: Insights and Strategies

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

Command:

#Find Pickup hour

df1_14['Pickup_hour'] = df['tpep_pickup_datetime'].dt.hour

 # Get trip duration in hour

df1_14['trip_duration_hour'] = df1_14['trip_duration_minutes']/60

 # Group the data based 'PULocationID', 'DOLocationID', 'Pickup_hour' to find average duration and average distance.

df1_14_grouped = df1_14.groupby(['PULocationID', 'DOLocationID', 'Pickup_hour']).agg(

avg_duration=('trip_duration_hour', 'mean'),

avg_distance=('trip_distance', 'mean'),

).reset_index()

 # find the average speed

df1_14_grouped['avg_speed'] = df1_14_grouped['avg_distance'] / df1_14_grouped['avg_duration']

 # Find the 20 slowest route based on 'PULocationID', 'DOLocationID', 'Pickup_hour'

slow_routes = df1_14_grouped.sort_values(by=['avg_speed'])

slow_routes

slow_routes.head(20)

Output: Sorting the value based avg_speed in ascending order provides the information on slow routs. Displaying the 20 slowest routes based on pickup hours.

	PULocationID	DOLocationID	Pickup_hour	avg_duration	avg_distance	avg_speed
99757	243	264	16.0	23.159167	0.180000	0.007772
7602	45	45	17.0	0.840556	0.050000	0.059484
80616	224	113	12.0	12.000278	0.850000	0.070832
16241	70	138	14.0	17.376111	1.490000	0.085750
42779	137	233	3.0	6.099861	0.780000	0.127872
9851	48	184	15.0	1.094444	0.140000	0.127919
78636	211	52	6.0	23.848889	3.170000	0.132920
5942	43	24	4.0	11.993889	1.680000	0.140071
31284	113	66	19.0	23.809722	3.430000	0.144059
76977	193	193	9.0	0.066667	0.010000	0.150000
3095	13	264	15.0	0.126667	0.020000	0.157895
69426	164	112	10.0	23.621111	3.750000	0.158756
7611	45	48	11.0	23.553889	4.190000	0.177890
47792	140	140	6.0	2.008009	0.413333	0.205842
8495	48	12	2.0	23.396389	4.960000	0.211999
77658	209	125	10.0	6.099236	1.365000	0.223799
4928	41	41	18.0	3.058194	0.702500	0.229711
55595	144	217	20.0	12.024028	2.800000	0.232867
4094	25	79	18.0	23.256111	5.420000	0.233057
78138	209	232	23.0	6.067708	1.420000	0.234026

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

Command:

 # Visualise the number of trips per hour and find the busiest hour

trip_per_hour = df1_14.groupby(['Pickup_hour']).agg(count_of_trip = ('tpep_pickup_datetime', 'count')).reset_index()

Busiest_hour = trip_per_hour.sort_values(by = 'count_of_trip', ascending = False).head(1).iloc[0,0]

Busiest_hour_trip_count = trip_per_hour.sort_values(by = 'count_of_trip', ascending = False).head(1).iloc[0,1]

print(f'\033[1mThe busiest hour of the day is {Busiest_hour} with {Busiest_hour_trip_count} trips\033[0m')

plt.plot(trip_per_hour.Pickup_hour, trip_per_hour.count_of_trip, marker='o')

plt.title('Number of trips per hour')

plt.xlabel('Pickup hour (0-23)')

plt.ylabel('Number of trips')

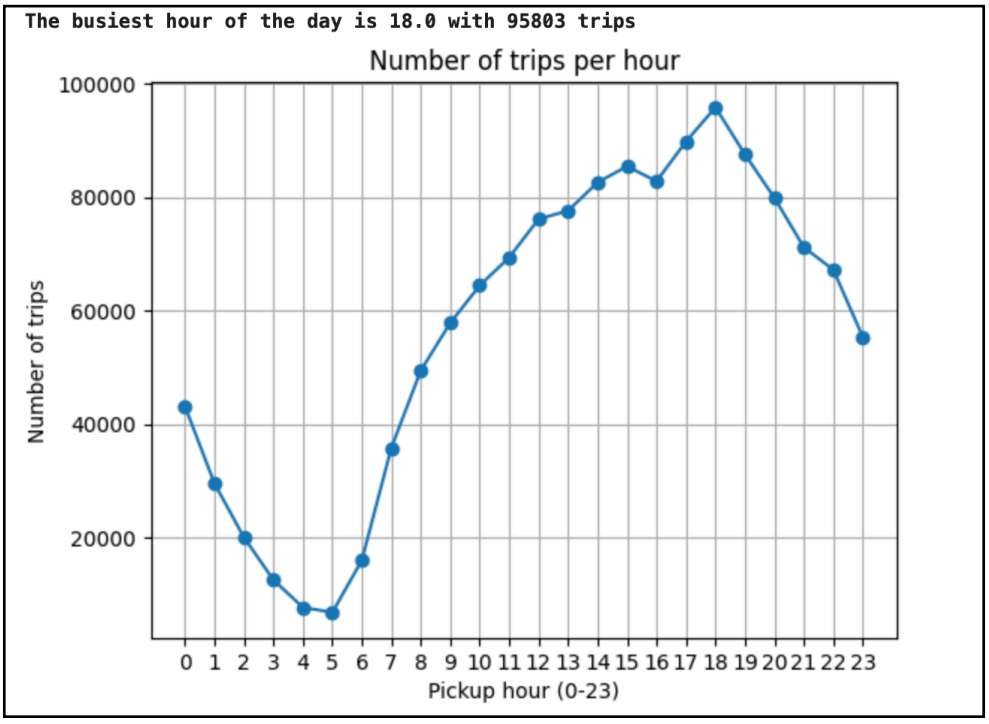
plt.xticks(range(0, 24))

#plt.ylim(0, 4500000)

plt.grid(True)

plt.show()

Output: The busiest hour of the day is 18.0 hour (6:00 pm) with 95803 trips.



3.2.3 Find the actual number of trips in the five busiest hours

Command:

```
# Scale up the number of trips

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

sample_fraction = 0.05

trip_per_hour['count_of_trip_scaled_up'] = trip_per_hour['count_of_trip']/sample_fraction

Busiest_hour = trip_per_hour.sort_values(by = 'count_of_trip_scaled_up', ascending =
False ).head(5)

print('\033[1mBelow are the 5 busiest hours and their respective scaled up number of
trips\033[0m')

Busiest_hour[[‘Pickup_hour','count_of_trip_scaled_up']]
```

Output: Below are the 5 busiest hours and their respective scaled up number of trips

Pickup_hour	count_of_trip_scaled_up
18.0	1916060.0
17.0	1795560.0
19.0	1750580.0
15.0	1708120.0
16.0	1655760.0

3.2.4 Compare hourly traffic pattern on weekdays. Also compare for weekend.

Command:

```
# Compare traffic trends for the week days and weekends

df1_14['days_of_week'] = df1_14['tpep_pickup_datetime'].dt.day_name()

# First find the horly trend for every day

hourly_trend_per_day_of_week  =
df1_14.groupby(['Pickup_hour','days_of_week']).agg(count_of_trip=('tpep_pickup_datetime',
'count')).reset_index()

## Per Day hourly traffic

Monday =
hourly_trend_per_day_of_week[hourly_trend_per_day_of_week.days_of_week.isin(['Monday'])]

Tuesday =
hourly_trend_per_day_of_week[hourly_trend_per_day_of_week.days_of_week.isin(['Tuesday'])]

Wednesday =
hourly_trend_per_day_of_week[hourly_trend_per_day_of_week.days_of_week.isin(['Wednesday']
)]
```

```
Thursday =
hourly_trend_per_day_of_week[hourly_trend_per_day_of_week.days_of_week.isin(['Thursday'])]

Friday =
hourly_trend_per_day_of_week[hourly_trend_per_day_of_week.days_of_week.isin(['Friday'])]

Saturday =
hourly_trend_per_day_of_week[hourly_trend_per_day_of_week.days_of_week.isin(['Saturday'])]

Sunday =
hourly_trend_per_day_of_week[hourly_trend_per_day_of_week.days_of_week.isin(['Sunday'])]


## Weekdays hourly traffic

hourly_trend_for_week_day =
hourly_trend_per_day_of_week[hourly_trend_per_day_of_week.days_of_week.isin(['Monday',
'Tuesday','Wednesday','Thursday','Friday'])]


## Weekend hourly traffic

hourly_trend_for_weekend =
hourly_trend_per_day_of_week[hourly_trend_per_day_of_week.days_of_week.isin(['Saturday',
'Sunday'])]


## Avg Weekdays hourly traffic

Avg_hourly_trend_for_week_day  =
hourly_trend_for_week_day.groupby(['Pickup_hour']).agg(Avg_count_of_trip=('count_of_trip',
'mean')).reset_index()


## Avg Weekend hourly traffic

Avg_hourly_trend_for_weekend  =
hourly_trend_for_weekend.groupby(['Pickup_hour']).agg(Avg_count_of_trip=('count_of_trip',
'mean')).reset_index()

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

# Daily trends

plt.plot(Monday.Pickup_hour, Monday.count_of_trip, label='Monday', color='red')

plt.plot(Tuesday.Pickup_hour, Tuesday.count_of_trip, label='Tuesday', color='blue')

plt.plot(Wednesday.Pickup_hour, Wednesday.count_of_trip, label='Wednesday', color='green')

plt.plot(Thursday.Pickup_hour, Thursday.count_of_trip, label='Thursday', color='orange')

plt.plot(Friday.Pickup_hour, Friday.count_of_trip, label='Friday', color='purple')

plt.plot(Saturday.Pickup_hour, Saturday.count_of_trip, label='Saturday', color='brown')

plt.plot(Sunday.Pickup_hour, Sunday.count_of_trip, label='Sunday', color='pink')
```

```
# Plot formatting
plt.title('Hourly Trip Trends by Day of Week')
plt.xlabel('Hour of Day')
plt.ylabel('Number of Trips')
plt.xticks(range(0, 24))
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 6))
# Average trends
plt.plot(Avg_hourly_trend_for_week_day.Pickup_hour,
    Avg_hourly_trend_for_week_day.Avg_count_of_trip,
    label='Weekday Avg', color='red', linewidth=2, linestyle='--')

plt.plot(Avg_hourly_trend_for_weekend.Pickup_hour,
    Avg_hourly_trend_for_weekend.Avg_count_of_trip,
    label='Weekend Avg', color='black', linewidth=2, linestyle='--')
# Plot formatting
plt.title('Averaged Hourly Trip Trends Weekdays vs Weekends')
plt.xlabel('Hour of Day')
plt.ylabel('Number of Trips')
plt.xticks(range(0, 24))
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

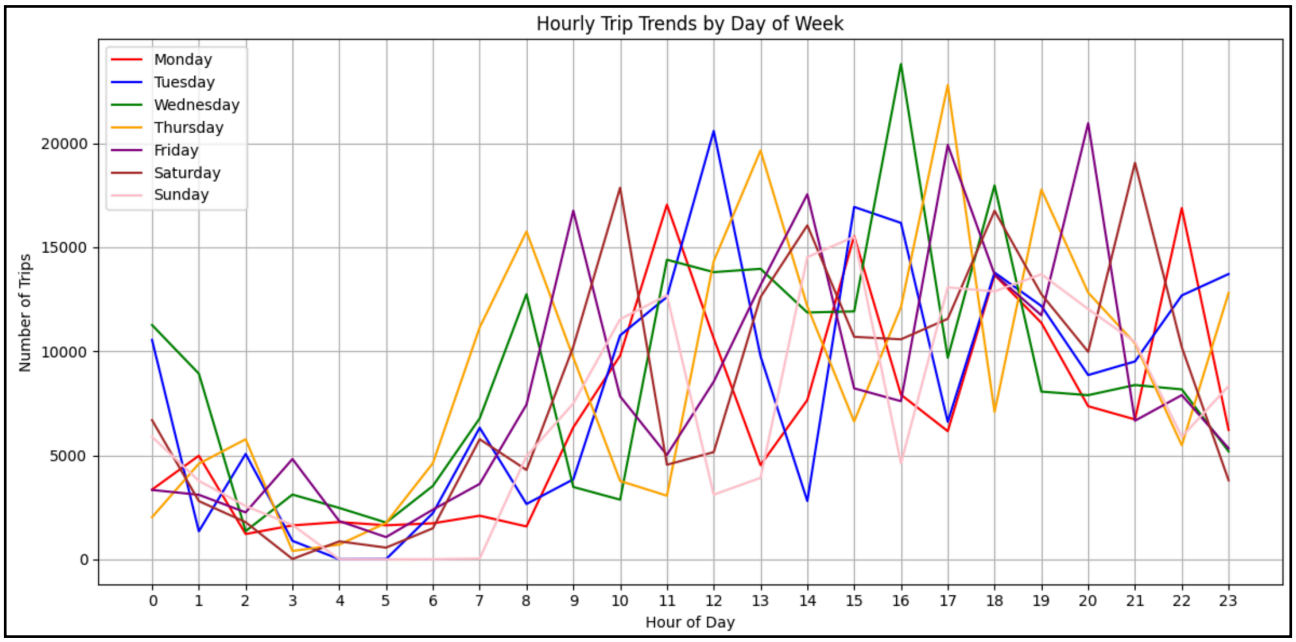
Output: There two kind of analysis that can be done using the data.¶

1. Per day hourly analysis to find busy and quite hours.
2. Overall averaged value in week day (Monday to Friday) vs weekend (Saturday and Sunday)

Method 1: Per day hourly analysis to find busy and quite hours.

1. Quite hours - This consistently found after 2:00 AM to early morning (before 6 AM). This is a little prolonged till 7:00 AM in the morning for Sunday, which make sense since it a holiday and people want to relax.
2. Busy hour - Every day has its own peak hour between morning to noon hours and then evening hours.
3. Example:- Monday has peak hour at 11, 15 and 22 hour. People start late in the morning and go home late. Tuesday has peak hour at noon. Wednesday has peak hour at 8, 11 and 16 in increasing order. Thursday has peak at hour 8, 13 and 17 in increasing order. Friday had peak hours at 9, 14, 17, 20 which make sense as people want to finish their work and go home. Saturday has peak hour at 10, 14, 18. Sunday is relatively a less busy day, the peak hour is at 11, 15, 17 and 19 hour.

Hourly Trend for every day in the week

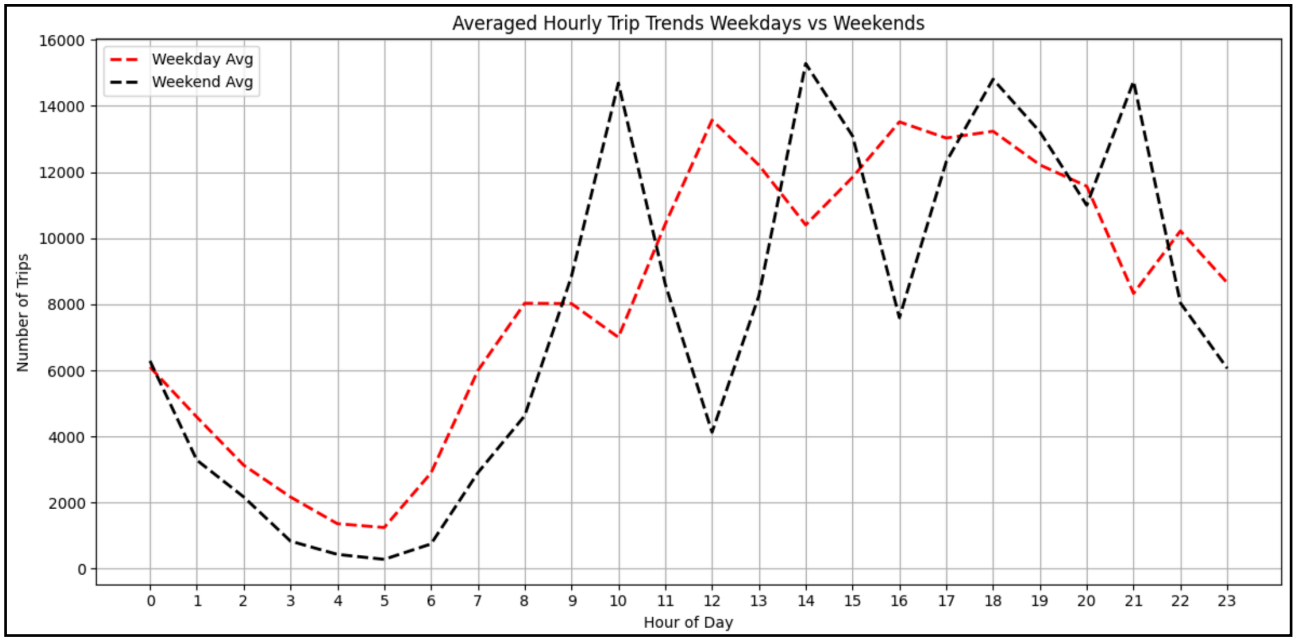


Method 2: Overall averaged value in week day (Monday to Friday) vs weekend (Saturday and Sunday)

1. Averaged hourly value for week days shows that traffic peaks at noon and remains busy till 18 and then goes down slowly. Quite hour start from 2:00 to 5:00 and then peaks up slowly
2. Averaged hourly value for weekend show more traffic at 10, 14, 18 and 21. Quite hour start from 2:00 to 5:00 and then peaks up slowly.

3. In general traffic is much lower on weekend as compared to weekdays till 8:00. But then it peaks up. Between 10:00 to 21:00, there is contrasting trend for weekends vs weekdays busy hours.
4. Averaged hourly value for weekend show more traffic at 10, 14, 18 and 21 hour than weekday averaged values.

Hourly Trend for weekdays vs weekends



3.2.5 Identify top 10 zones with high hourly pickups. Do the same for hourly dropoffs. Show pickup and dropoff trends in these zones.

Command:

 # Find top 10 pickup and dropoff zones

Top 10 pickup zones

Top_10_zones_with_high_pickups = df1_14.groupby(['PULocationID']).agg(count_of_pickups_in_zone=('tpep_pickup_datetime', 'count')

).reset_index().sort_values(by = 'count_of_pickups_in_zone', ascending = False).head(10)

Top_10_zone_name_with_high_pickups = pd.merge(Top_10_zones_with_high_pickups,zones[['LocationID','zone']], how = 'inner', left_on = 'PULocationID', right_on = 'LocationID')[['zone', 'count_of_pickups_in_zone']]

#Top_10_zone_name_with_high_pickups

print('\033[1mTop 10 pickup zones\033[0m \n',Top_10_zone_name_with_high_pickups,'\n')

print('_____ \n\n')

#Top 10 dropoff zones

Top_10_zones_with_high_dropoff = df1_14.groupby(['DOLocationID']).agg(count_of_dropoff_in_zone=('tpep_dropoff_datetime', 'count')).reset_index().sort_values(by = 'count_of_dropoff_in_zone', ascending = False).head(10)

Top_10_zone_name_with_high_dropoffs = pd.merge(Top_10_zones_with_high_dropoff,zones[['LocationID','zone']], how = 'inner', left_on = 'DOLocationID', right_on = 'LocationID')[['zone', 'count_of_dropoff_in_zone']]

print('\033[1mTop 10 dropoff zones\033[0m \n',Top_10_zone_name_with_high_dropoffs)

Output *:* Top 10 pickup and dropoff zones

Top 10 pickup zones

	zone	count_of_pickups_in_zone
0	Upper East Side South	68714
1	Midtown Center	66652
2	JFK Airport	63064
3	Upper East Side North	62557
4	Midtown East	51884
5	LaGuardia Airport	51090
6	Lincoln Square East	48079
7	Penn Station/Madison Sq West	47768
8	Times Sq/Theatre District	43760
9	Murray Hill	42492

Top 10 dropoff zones

	zone	count_of_dropoff_in_zone
0	Upper East Side North	66302
1	Upper East Side South	61355
2	Midtown Center	54346
3	Murray Hill	42517
4	Upper West Side South	41976
5	Lincoln Square East	40754
6	Midtown East	40572
7	Lenox Hill West	39101
8	Times Sq/Theatre District	38640
9	East Chelsea	35487

3.2.6 Find the ratio of pickups and dropoffs in each zone. Display the 10 highest (pickup/drop) and 10 lowest (pickup/drop) ratios.

Command:

```
# Find the top 10 and bottom 10 pickup/dropoff ratios

## Zone wise pickup counts

zones_wise_count_with_pickups =
df1_14.groupby(['PULocationID']).agg(count_of_pickups_in_zone=('tpep_pickup_datetime',
'count')).reset_index()

zones_name_wise_count_with_pickups =
pd.merge(zones_wise_count_with_pickups,zones[['LocationID','zone']], how = 'inner', left_on  =
'PULocationID', right_on = 'LocationID')[['zone', 'count_of_pickups_in_zone']]

## Zone wise dropoff counts

zones_wise_count_with_dropoff =
df1_14.groupby(['DOLocationID']).agg(count_of_dropoff_in_zone=('tpep_dropoff_datetime',
'count')).reset_index()

zones_name_wise_count_with_dropoff =
pd.merge(zones_wise_count_with_dropoff,zones[['LocationID','zone']], how = 'inner', left_on  =
'DOLocationID', right_on = 'LocationID')[['zone', 'count_of_dropoff_in_zone']]

## Merge zone wise pickup and dropoff count data.

zones_wise_count_with_pickups_and_dropoff =
pd.merge(zones_name_wise_count_with_pickups,zones_name_wise_count_with_dropoff,
how='outer', on =  'zone').drop_duplicates()

## Find ratio(pickup/drop)

zones_wise_count_with_pickups_and_dropoff['ratio(pickup/drop)'] =
zones_wise_count_with_pickups_and_dropoff['count_of_pickups_in_zone'] /
zones_wise_count_with_pickups_and_dropoff['count_of_dropoff_in_zone']
'''

Since the analysis is run on just 5% of sample, there are zone with no pickup and drop count. These
all values will get flagged as "NaN" value and their ratio (pickup/drop) will also be "NaN".

Hence, dropping rows with ratio as "NaN" values for further analysis.

'''

## Filter out NaN values in from ratio(pickup/drop)

zones_wise_ratio_with_non_nan =
zones_wise_count_with_pickups_and_dropoff[~(zones_wise_count_with_pickups_and_dropoff['ra
tio(pickup/drop)'].isnull())]


## Sort ratio(pickup/drop) in ascending order
```

zones_wise_ratio_sorted = zones_wise_ratio_with_non_nan.sort_values(by = 'ratio(pickup/drop)', ascending = False)

print('\033[1m10 highest (pickup/drop) ratios\033[0m \n',zones_wise_ratio_sorted[['zone','ratio(pickup/drop)']].head(10), '\n\n==========================\n')

print('\033[1m10 lowest (pickup/drop) ratios\033[0m \n',zones_wise_ratio_sorted[['zone','ratio(pickup/drop)']].tail(10), '\n')

Output:

Since the analysis is run on just 5% of sample, there are zone with no pickup and drop count. These all values will get flagged as "NaN" value and their ratio (pickup/drop) will also be "NaN".

Hence, dropping rows with ratio as "NaN" values for further analysis.

10 highest (pickup/drop) ratios

	zone	ratio(pickup/drop)
69	East Elmhurst	14.120267
123	JFK Airport	4.763502
133	LaGuardia Airport	2.892815
203	Saint Michaels Cemetery/Woodside	2.000000
182	Penn Station/Madison Sq West	1.650188
108	Greenwich Village South	1.386735
40	Central Park	1.385507
245	West Village	1.333067
157	Midtown East	1.278813
98	Garment District	1.227710

==========

10 lowest (pickup/drop) ratios

	zone	ratio(pickup/drop)
11	Bay Ridge	0.011019
253	Windsor Terrace	0.009615
204	Schuylerville/Edgewater Park	0.009434
35	Cambria Heights	0.009434
19	Bensonhurst West	0.009174
17	Belmont	0.009174
168	Newark Airport	0.009079
160	Midwood	0.008889
201	Saint Albans	0.006993
223	Sunset Park East	0.006579

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

Command:

Top pickup zone in night hour (between 11pm to 5am)

pickup_night = df1_14[df1_14.Pickup_hour.isin([23,0,1,2,3,4,5])]

zones_wise_pickup_night_count = pickup_night.groupby(['PULocationID']).agg(count_of_pickups_in_zone=('tpep_pickup_datetime', 'count')).reset_index()

```
zones_name_wise_pickup_night_count =
pd.merge(zones_wise_pickup_night_count,zones[['LocationID','zone']], how = 'inner', left_on  =
'PULocationID', right_on = 'LocationID').drop_duplicates().sort_values(by =
'count_of_pickups_in_zone', ascending = False)

zones_name_wise_pickup_night_count[['zone', 'count_of_pickups_in_zone']]

print('\033[1mTop 10 pickup zones during night hours(11pm to 5am)\033[0m
\n\n',zones_name_wise_pickup_night_count[['zone', 'count_of_pickups_in_zone']].head(10),
'\n\n=====================================\n')


## Top dropoff zone in night hour (between 11pm to 5am)

df1_14['Dropoff_hour'] = df1_14['tpep_dropoff_datetime'].dt.hour

dropoff_night = df1_14[df1_14.Dropoff_hour.isin([23,0,1,2,3,4,5])]

zones_wise_dropoff_night_count =
dropoff_night.groupby(['DOLocationID']).agg(count_of_dropoff_in_zone=('tpep_dropoff_datetime'
, 'count')).reset_index()

zones_name_wise_dropoff_night_count =
pd.merge(zones_wise_dropoff_night_count,zones[['LocationID','zone']], how = 'inner', left_on  =
'DOLocationID', right_on = 'LocationID').drop_duplicates().sort_values(by =
'count_of_dropoff_in_zone', ascending = False)

zones_name_wise_dropoff_night_count[['zone', 'count_of_dropoff_in_zone']]

print('\033[1mTop 10 dropoff zones during night hours(11pm to 5am)\033[0m
\n\n',zones_name_wise_dropoff_night_count[['zone', 'count_of_dropoff_in_zone']].head(10), '\n')
```

Output: Top 10 pickup and drop-off sones during night hours are:

```
Top 10 pickup zones during night hours(11pm to 5am)

                             zone  count_of_pickups_in_zone
145         Upper East Side South                      9090
96                 Midtown Center                      8741
75                    JFK Airport                      8225
144         Upper East Side North                      8224
97                   Midtown East                      6822
80              LaGuardia Airport                      6655
110  Penn Station/Madison Sq West                      6053
83             Lincoln Square East                      6036
139     Times Sq/Theatre District                      5544
103                   Murray Hill                      5531

=====================================

Top 10 dropoff zones during night hours(11pm to 5am)

                       zone  count_of_dropoff_in_zone
77            East Village                      7141
46            Clinton East                      5464
163            Murray Hill                      5321
101               Gramercy                      5146
134        Lenox Hill West                      4894
66            East Chelsea                      4777
254         Yorkville West                      4617
240           West Village                      4394
230  Upper West Side South                      4172
227  Upper East Side North                      4124
```

3.2.8 Find the revenue share for nighttime and daytime hours.

Command:

 # Filter for night hours (11 PM to 5 AM)

Revenue is Based on Pickup Time

###pickup_night

nighttime_hours = pickup_night.copy(deep=True)

night_revenue = nighttime_hours.total_amount.sum()

print(f'\033[1mRevenue collected during night hours(11pm to 5am) is {round(night_revenue,2)} \033[0m \n\n')

###pickup_day

daytime_hours = df1_14[(df1_14.Pickup_hour.isin([22., 6., 7., 8., 9., 10., 11., 12., 13., 14., 15., 16.,17., 18., 19., 20., 21.]))]

day_revenue = daytime_hours.total_amount.sum()

print(f'\033[1mRevenue collected during day hours(6am to 10pm) is {round(day_revenue,2)} \033[0m \n\n')

comparison of days revenue with night revenue

fold_change = round(day_revenue,2)/round(night_revenue,2)

print(f'\033[1mRevenue collected during day hours(6am to 10pm) is {round(fold_change,2)} time of night revenue\033[0m \n\n')

Output:

Revenue collected during night hours(11pm to 5am) is 5262030.74.

Revenue collected during day hours(6am to 10pm) is 35094976.05.

Revenue collected during day hours(6am to 10pm) is 6.67 time of night revenue .

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

Command:

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

df1_14['fare_per_mile'] = round(df1_14['fare_amount']/df1_14['trip_distance'],2)

df1_14['fare_per_mile_per_passenger'] = round(df1_14['fare_per_mile']/ df1_14['passenger_count'],2)

fare_based_passenger_count = df1_14.groupby(['passenger_count']).agg(Avg_fare_per_mile_per_passenger = ('fare_per_mile_per_passenger','mean')).reset_index()

print('\033[1mConclusion - More the passenger count for a trip, there is lesser average fare per mile per passenger \033[0m \n\n', fare_based_passenger_count,'\n\n')

Output: More the passenger count for a trip, there is lesser average fare per mile per passenger

passenger_count	Avg_fare_per_mile_per_passenger
1	9.185894
2	4.920596
3	3.316006
4	3.768017
5	1.589689
6	1.284483

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

Command:

#Average fare per mile for different days

Average_fare_per_mile_for_different_days = df1_14.groupby(['days_of_week']).agg(Avg_fare_per_mile = ('fare_per_mile','mean')).reindex(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

Average_fare_per_mile_for_different_days

print('\033[1mAverage fare per mile is relatively higher on Sunday of all the days of week\033[0m \n\n',Average_fare_per_mile_for_different_days)

plt.figure(figsize=(8, 6))

 # Daily trends

plt.plot(Average_fare_per_mile_for_different_days.index, Average_fare_per_mile_for_different_days.Avg_fare_per_mile)

 # Plot formatting

plt.title('Average fare per mile by Day of Week')

plt.xlabel('Day of Week')

plt.ylabel('Average fare per mile')

plt.grid(True)

plt.ylim(0,12)

plt.xticks(rotation = 90)

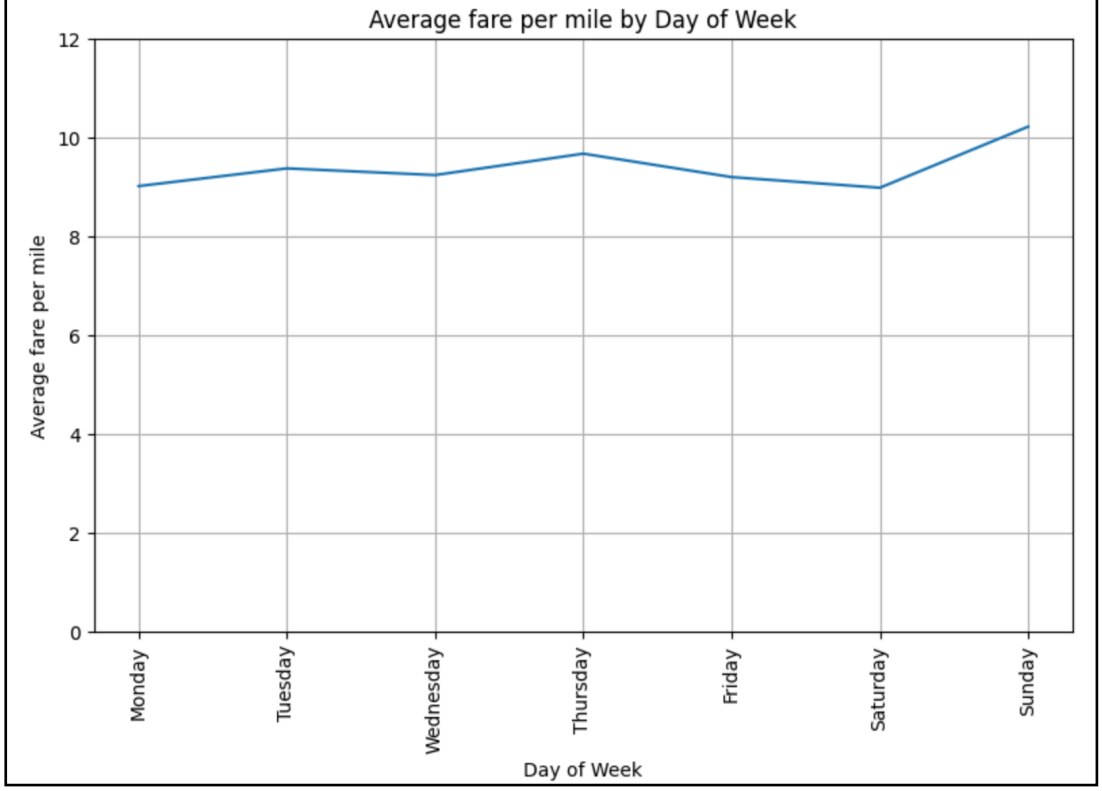
plt.tight_layout()

plt.show()

```
#Average fare per mile for different times of the day

Average_fare_per_mile_for_different_hour_of_day =
df1_14.groupby(['Pickup_hour']).agg(Avg_fare_per_mile = ('fare_per_mile','mean'))

Average_fare_per_mile_for_different_hour_of_day

print('\033[1mAverage fare per mile is relatively higher(over 10) at 10,20 and 23 hour of
day\033[0m \n\n',Average_fare_per_mile_for_different_hour_of_day)

plt.figure(figsize=(8, 6))

# Daily trends

plt.plot(Average_fare_per_mile_for_different_hour_of_day.index,
Average_fare_per_mile_for_different_hour_of_day.Avg_fare_per_mile)

# Plot formatting

plt.title('Average fare per mile by hour of day')

plt.xlabel('Hour of day')

plt.ylabel('Average fare per mile')

plt.grid(True)

plt.ylim(0,12)

plt.xticks(range(0, 24))

plt.xticks(rotation = 90)

plt.tight_layout()

plt.show()
```

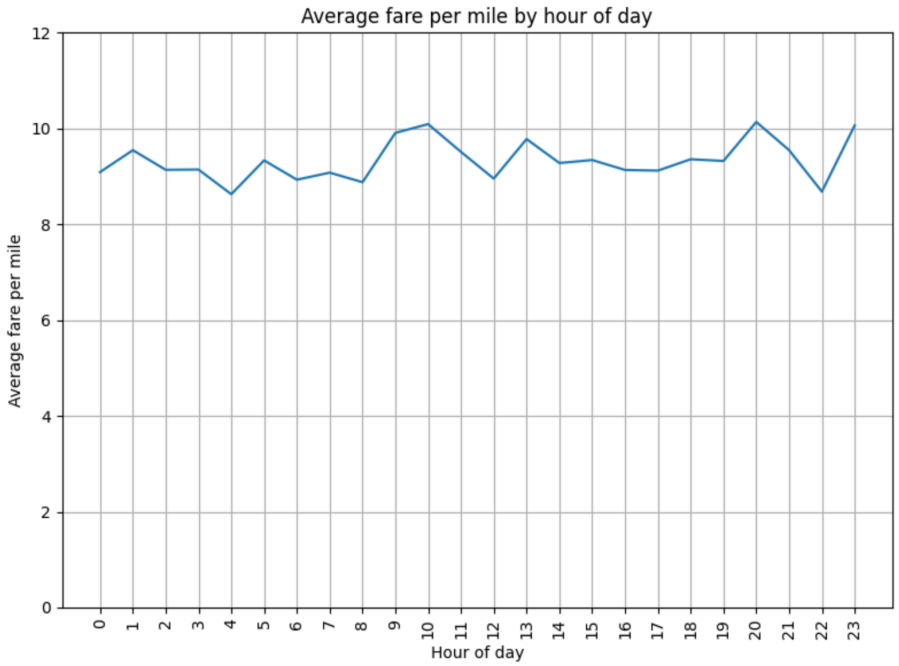
Output: Average fare per mile is relatively higher on Sunday of all the days of week.

days_of_week	Avg_fare_per_mile
Monday	9.027040
Tuesday	9.385368
Wednesday	9.250473
Thursday	9.683485
Friday	9.208735
Saturday	8.993628
Sunday	10.230180



Average fare per mile is relatively higher(over 10) at 10,20 and 23 hour of day

Pickup_hour	Avg_fare_per_mile
0.0	9.089970
1.0	9.547013
2.0	9.137831
3.0	9.143350
4.0	8.628054
5.0	9.333927
6.0	8.931048
7.0	9.078208
8.0	8.879060
9.0	9.907960
10.0	10.091293
11.0	9.511313
12.0	8.953768
13.0	9.778977
14.0	9.279100
15.0	9.341220
16.0	9.134349
17.0	9.120590
18.0	9.357966
19.0	9.321764
20.0	10.136615
21.0	9.547929
22.0	8.681847
23.0	10.065190



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

Output: Average fare per mile is relatively higher for Vendor 2 (VeriFone Inc.). Even based on hours, Average fare per mile for VendorID 2 is higher than VendorID1 for every hour.

Average fare per mile is relatively higher for Vendor 2 (VeriFone Inc.)

VendorID	Avg_fare_per_mile
1	7.900854
2	9.872120

Even based on hours, Average fare per mile for VendorID 2 is higher than VendorID1 for every hour

	Pickup_hour	Avg_fare_per_mile_VendorID_1	Avg_fare_per_mile_VendorID_2
0	0.0	7.961883	9.445770
1	1.0	7.972942	10.061606
2	2.0	8.003628	9.509975
3	3.0	8.317804	9.417507
4	4.0	8.187591	8.766127
5	5.0	8.610283	9.567857
6	6.0	7.822009	9.287584
7	7.0	7.964732	9.445846
8	8.0	8.000140	9.163285
9	9.0	7.779188	10.587209
10	10.0	7.735476	10.847650
11	11.0	7.824936	10.064172
12	12.0	7.963140	9.282891
13	13.0	8.032543	10.343843
14	14.0	7.846689	9.734779
15	15.0	7.925230	9.801253
16	16.0	8.007938	9.497988
17	17.0	7.816516	9.537049
18	18.0	7.825948	9.864305
19	19.0	7.689094	9.847435
20	20.0	7.889030	10.865104
21	21.0	8.035335	10.030459
22	22.0	7.815440	8.964543
23	23.0	7.993274	10.746480

3.2.12 Compare the fare rates of the different vendors in a tiered fashion. Analyse the average fare per mile for distances upto 2 miles. Analyse the fare per mile for distances from 2 to 5 miles. And then for distances more than 5 miles.

Output: There are two ways this analysis was performed.

Method 1: Overall fare per mile was averaged for distance tiers

Average fare per mile is much higher for "Upto 2 miles" distance trip. VendorID - 2 has even higher fair than VendorID - 1

Average fare per mile is much higher for "Upto 2 miles" distance trip. VendorID – 2 has even higher fair than VendorID – 1

	distance_tier	Avg_fare_per_mile_vendorID1	Avg_fare_per_mile_vendorID2
1	From 2 to 5 miles	6.354308	6.545466
2	More than 5 miles	4.469885	4.506004
0	Upto 2 miles	9.494907	13.325603

Method 2: Hourly fare per mile was averaged for distance tiers

Even in hourly analysis, average fare per mile is much higher for "Upto 2 miles" distance trip . VendorID - 2 has even higher fair than VendorID - 1

	distance_tier	Pickup_hour	Avg_fare_per_mile_vendorID1	Avg_fare_per_mile_vendorID2
0	From 2 to 5 miles	0.0	6.353784	6.509017
1	From 2 to 5 miles	1.0	6.432883	6.697411
2	From 2 to 5 miles	2.0	6.455736	6.666682
3	From 2 to 5 miles	3.0	6.497714	6.743646
4	From 2 to 5 miles	4.0	6.474403	6.757426
...	...	...	...	...
67	Upto 2 miles	19.0	9.279463	13.568596
68	Upto 2 miles	20.0	9.614003	15.474088
69	Upto 2 miles	21.0	9.610379	13.487665
70	Upto 2 miles	22.0	9.368161	11.833250
71	Upto 2 miles	23.0	9.606797	14.742152

72 rows × 4 columns

3.2.13 Analyse average tip percentages based on trip distances, passenger counts and time of pickup. What factors lead to low tip percentages?

Output:

trip distances vs average tip percentages: More the distance lesser is tip percentage.

More the distance lesser is tip percentage.

distance_tier	Avg_tip_perc
From 2 to 5 miles	23.160504
More than 5 miles	23.585587
Upto 2 miles	28.843776

passenger count vs average tip percentages: There is no relationship between passenger count and tip amount. It seems to be in same range.

There is no relationship between passenger count and tip amount. It seems to be in same range.

passenger_count	Avg_tip_perc
1	26.516163
2	25.825526
3	25.811287
4	25.901587
5	26.054756
6	26.171913

Pickup hour vs average tip percentages: Tip amount seems to be same range except for pick up at 17 hour which has relatively better tip amount than other.

Tip amount seems to be same range except for pick up at 17 hour which has relatively better tip amount than other.

Pickup_hour	Avg_tip_perc
0.0	25.919159
1.0	25.890842
2.0	26.054651
3.0	25.517150
4.0	26.211295
5.0	26.871709
6.0	26.388689
7.0	25.979963
8.0	26.325957
9.0	26.054736
10.0	26.072623
11.0	26.182420
12.0	26.205929
13.0	26.166253
14.0	26.065856
15.0	25.892773
16.0	26.280462
17.0	30.089353
18.0	26.273574
19.0	26.218472
20.0	26.062206
21.0	25.841783
22.0	26.120954
23.0	26.065108

Multivariate analysis using distance_tier, passenger_count, Pickup_hour with Avg_tip_perc - Even with multivariant analysis, more the distance lesser is the tip percentage.

Even with multivariant analysis, more the distance lesser is the tip percentage.

	distance_tier	passenger_count	Pickup_hour	Avg_tip_perc
0	More than 5 miles	6	5.0	18.465968
1	More than 5 miles	6	21.0	19.218965
2	More than 5 miles	2	4.0	20.363200
3	From 2 to 5 miles	5	3.0	20.397421
4	More than 5 miles	4	4.0	20.651451
..	...	...	...	...
427	Upto 2 miles	3	5.0	30.318807
428	Upto 2 miles	6	4.0	31.628135
429	Upto 2 miles	4	4.0	32.412509
430	Upto 2 miles	6	16.0	34.262445
431	More than 5 miles	1	17.0	54.156559

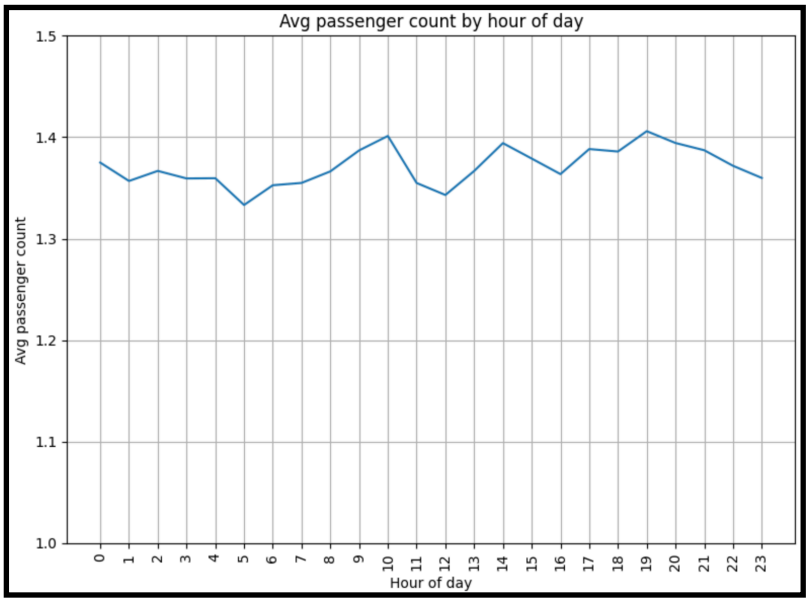
3.2.14 Analyse the variation of passenger count across hours and days of the week.

Output:

variation of passenger count across hours

Averaged passenger count across hours show peak at 10, 14 and evening hours of the day

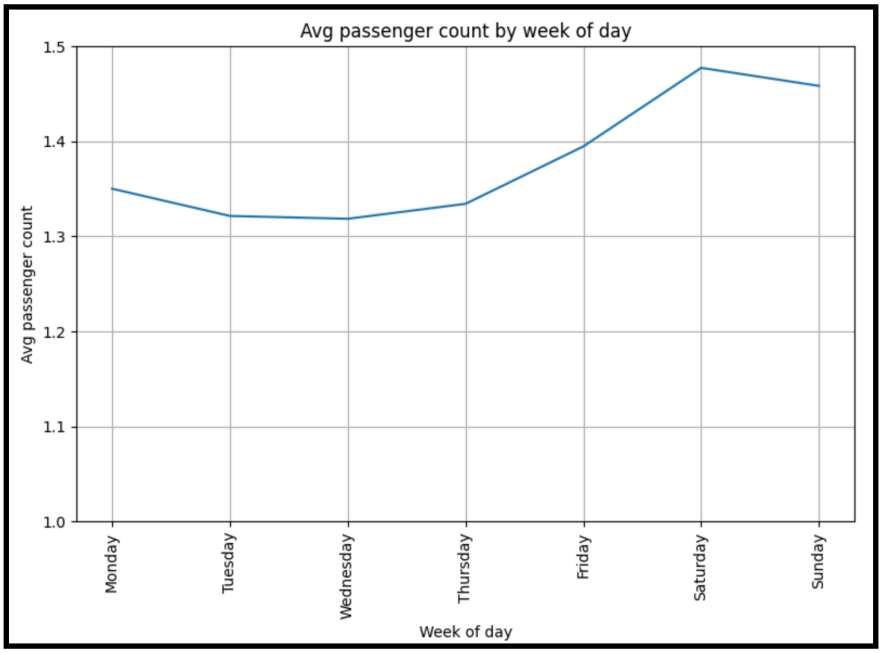
Pickup_hour	avg_passenger_count
0.0	1.375006
1.0	1.356906
2.0	1.366827
3.0	1.359409
4.0	1.359619
5.0	1.333235
6.0	1.352709
7.0	1.354992
8.0	1.366366
9.0	1.387009
10.0	1.401248
11.0	1.354929
12.0	1.34309
13.0	1.366764
14.0	1.394187
15.0	1.378978
16.0	1.363676
17.0	1.388347
18.0	1.385917
19.0	1.405945
20.0	1.394313
21.0	1.387176
22.0	1.371842
23.0	1.359891



variation of passenger count across days of the week

Averaged passenger count is higher toward end of the week and specially on Saturday and Sunday

days_of_week	avg_passenger_count
Monday	1.350275
Tuesday	1.321591
Wednesday	1.318623
Thursday	1.334373
Friday	1.394813
Saturday	1.477517
Sunday	1.458523



3.2.15 Analyse the variation of passenger counts across zones

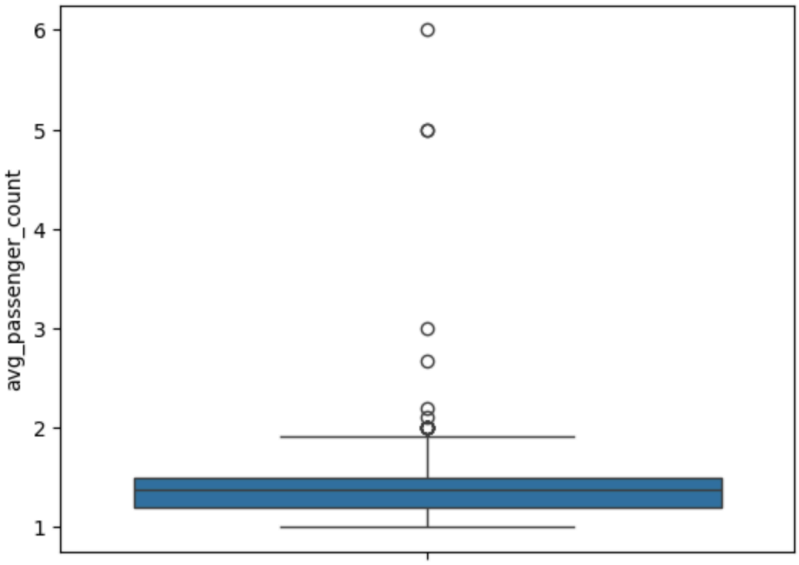
Output: There are fewer number of outlier beyond upper whisker in zone based analysis of averaged passenger counts

variantion across zones

	zone	avg_passenger_count
148	Ocean Parkway South	6.0
39	Claremont/Bathgate	5.0
209	West Farms/Bronx River	5.0
3	Arrochar/Fort Wadsworth	3.0
84	Fresh Meadows	2.666667
..	...	...
197	University Heights/Morris Heights	1.0
68	East New York/Pennsylvania Avenue	1.0
173	Schuylerville/Edgewater Park	1.0
31	Cambria Heights	1.0
103	Hunts Point	1.0

[224 rows x 2 columns]

There are fewer number of outlier beyond upper whisker in zone based analysis of averaged passenger counts



	avg_passenger_count
count	224.0
mean	1.43196
std	0.5472
min	1.0
25%	1.2
50%	1.370054
75%	1.5
max	6.0

 # For a more detailed analysis, we can use the zones_with_trips GeoDataFrame

 # Create a new column for the average passenger count in each zone.

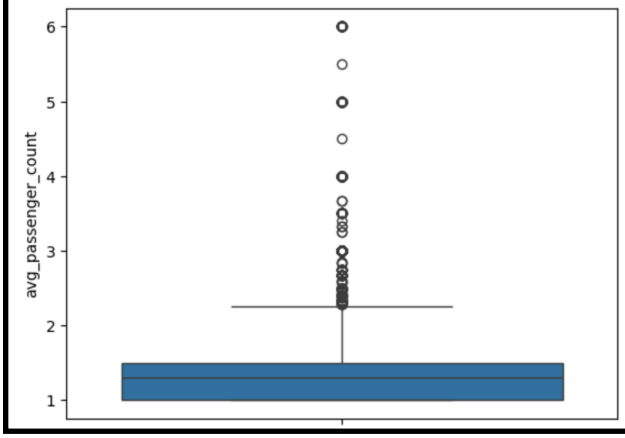
Output - There are too many number of outlier beyond upper whisker in zone_with_trip based analysis of averaged passenger counts and the lower whisker can not be seen.

Variantion in zones with trips

	Pickup_zone	Dropoff_zone	avg_passenger_count
3367	Flushing	LaGuardia Airport	6.0
10335	Upper West Side North	College Point	6.0
2390	East Elmhurst	Woodlawn/Wakefield	6.0
6028	Little Italy/NoLiTa	Rego Park	6.0
3455	Forest Hills	Cypress Hills	6.0
...	...	...	...
2739	East Village	Midwood	1.0
2741	East Village	Morrisania/Melrose	1.0
2744	East Village	Murray Hill-Queens	1.0
8014	Penn Station/Madison Sq West	Laurelton	1.0
8330	Ridgewood	Glendale	1.0

[11647 rows x 3 columns]

There are too many number of outlier beyond upper whisker in zone_with_trip based analysis of averaged passenger counts and the lower whisker can not be seen



	avg_passenger_count
count	11647.0
mean	1.387184
std	0.548733
min	1.0
25%	1.0
50%	1.309302
75%	1.5
max	6.0

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

Output: Overall the Frequency of different charges that applied are below:

Frequency(%) of different charges that are applied are:

```
 extra                  63.042595
mta_tax                 99.440599
improvement_surcharge   99.998985
congestion_surcharge    95.085332
Airport_Fee              8.278836
dtype: float64
```

Created a 'Surcharge/Extra' column that include (extra + mta_tax + improvement_surcharge + congestion_surcharge + Airport_Fee)

Frequency(%) of different charges (Surcharge/Extra) that are applied by pick up zone (pick up location) : Across difference pickup zones, there are variable % of zone paying different charges (surcharge or extra amounts). But ‘Surcharge/Extra’ which is inclusive of all the charges shows that 99.55% of pick up zone pays surcharge/Extra all the time.

Frequency(%) of different charges (Surcharge/Extra) that are applied by pick up zone (pick up location) :

	PULocationID	extra_applied	mta_tax_applied	improvement_surcharge_applied	congestion_surcharge_applied	Airport_Fee_applied	Surcharge/Extra_applied
0	1	11.428571	17.142857	100.0	2.857143	0.000000	100.0
1	2	0.000000	100.000000	100.0	100.000000	0.000000	100.0
2	4	86.294416	99.782451	100.0	99.782451	0.000000	100.0
3	6	100.000000	0.000000	100.0	0.000000	0.000000	100.0
4	7	50.996016	98.007968	100.0	41.035857	0.000000	100.0
...	...	...	...	...	...	...	...
218	258	50.000000	50.000000	100.0	0.000000	0.000000	100.0
219	260	40.776699	86.407767	100.0	45.631068	2.912621	100.0
220	261	57.634314	98.623920	100.0	98.741034	0.029278	100.0
221	262	48.128876	99.852833	100.0	99.789761	0.026280	100.0
222	263	58.567216	99.854392	100.0	99.428488	0.010921	100.0

223 rows × 7 columns

Frequency(%) of different charges (Surcharge/Extra) that are applied by drop-off zone (drop-off location) : Across difference drop-off zones, there are variable % of zone paying different charges (surcharge or extra amounts). But ‘Surcharge/Extra’ which is inclusive of all the charges shows that 99.61% of drop-off zone pays surcharge/Extra all the time.

Frequency (%) of different charges that are applied by drop-off zone (dropoff location) :

	DOLocationID	extra_applied	mta_tax_applied	improvement_surcharge_applied	congestion_surcharge_applied	Airport_Fee_applied	Surcharge/Extra_applied
0	1	25.239948	0.752270	100.000000	0.726329	4.020752	100.000000
1	3	69.230769	100.000000	100.000000	28.846154	61.538462	100.000000
2	4	77.044393	99.883178	100.000000	99.961059	5.996885	100.000000
3	5	80.000000	100.000000	100.000000	40.000000	40.000000	100.000000
4	6	68.181818	100.000000	100.000000	59.090909	40.909091	100.000000
...	...	...	...	...	...	...	...
253	261	61.843770	99.762808	100.000000	99.826059	14.278937	100.000000
254	262	68.519319	99.922206	100.000000	99.818480	6.435301	100.000000
255	263	68.553826	99.922179	100.000000	99.225032	5.664721	100.000000
256	264	92.107093	97.566064	99.930459	92.037552	6.502086	99.965229
257	265	55.439534	62.238951	100.000000	12.506071	63.963089	100.000000

258 rows × 7 columns

Times when extra charges are applied more frequently: Hourly analysis shows that either of the charges (Surcharge/Extra) are paid almost all the time. The distribution of different types charges per hours are different.

	Pickup_hour	extra_applied	mta_tax_applied	improvement_surcharge_applied	congestion_surcharge_applied	Airport_Fee_applied	Surcharge/Extra_applied
0	0.0	61.577921	99.343113	100.000000	94.944524	8.337589	100.000000
1	1.0	61.727433	99.379913	100.000000	95.198563	8.626999	100.000000
2	2.0	69.521096	99.425115	100.000000	95.180964	8.083383	100.000000
3	3.0	56.180583	99.440671	100.000000	94.646424	9.868158	100.000000
4	4.0	69.642857	99.465589	100.000000	95.242440	7.950991	100.000000
5	5.0	67.162421	99.278457	100.000000	95.376233	7.730820	100.000000
6	6.0	66.887749	99.486923	100.000000	95.050682	8.822425	100.000000
7	7.0	59.409512	99.491151	100.000000	95.403584	7.560041	100.000000
8	8.0	69.480927	99.518365	100.000000	95.657189	7.105130	100.000000
9	9.0	61.459955	99.489708	100.000000	95.040650	8.399931	100.000000
10	10.0	61.254057	99.430115	99.998447	95.172285	8.153853	100.000000
11	11.0	61.718266	99.420323	99.998558	94.974693	8.970569	100.000000
12	12.0	65.390173	99.511298	99.998686	95.349448	8.042564	100.000000
13	13.0	65.758404	99.474118	99.996133	95.393380	7.718086	100.000000
14	14.0	62.996064	99.428398	100.000000	95.056615	7.857100	100.000000
15	15.0	60.392712	99.412219	99.996487	95.037819	8.100133	100.000000
16	16.0	65.259458	99.437116	100.000000	95.499348	7.501087	100.000000
17	17.0	64.506895	99.448640	99.998886	95.124641	8.232529	99.998886
18	18.0	62.514744	99.503147	99.998956	95.073223	8.448587	100.000000
19	19.0	60.293160	99.405911	99.998858	94.685190	9.137543	100.000000
20	20.0	63.414848	99.363911	100.000000	94.689656	8.962849	100.000000
21	21.0	59.404605	99.378742	99.997189	95.042588	7.969527	100.000000
22	22.0	63.670195	99.492401	100.000000	94.650114	8.958157	100.000000
23	23.0	63.790550	99.416588	100.000000	95.013005	8.503721	100.000000

4 Conclusion

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

Ans:

1. Avoid/congesting routes (PUlocation-DUlocation) in particular hours where average speed of very slow. So that taxis are not held up in traffic zone.

2. Keep ample amount of taxi's deployed during busiest hour of day for operational efficiency, to meet the demand.

Demand keeps peaking from morning to evening time. But keeping ample number of taxis deployed in busiest hours which start from 3pm to 7pm will be key for operational efficiencies.

3. Understanding the pattern of taxi requirement and then keeping the zones/hours well staffed with Taxis will help. Example - During week days, the peak hour is from noon to late evening (8pm).

During weekends peak hour varies. Its at 10, 14, 18 and 21 hour of the day. Hence, understanding these patterns and having good number of taxis available will help in operational efficiencies.

4. Understanding the zone with high demand will help meet the operational efficiencies.

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.

Ans:

Below are the strategically positioning of cabs across different zones to make best use of insights uncovered. The positioning should be dependent on below factor:

1. High pickup zones.
2. Identify high drop-off zones and quickly reposition them to high pickup zones.
3. Looked for zones which have high drop-off as well high pickup. This strategy will help auto position the taxis.
4. Look for zones with high pickup/drop-off ratio. Position taxis around Airports and park which have higher pickups.
5. Position taxis in zones with active night life during nighttime hours.

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

Ans:

Pricing strategy to maximize revenue while maintaining competitive rates with other vendors:-

1. Time based dynamic faring - Morning and evening are rush hour and hence having slightly higher fair or additional rush surcharge will boost the pricing.

2. Having additional charge for more number of passenger_counts in a trip will boost the revenue.

3. Having slightly higher charge for more than 2 mile distance_trip will boost the revenue.

4. Having slightly higher charges on weekends will help.

5. Enhancing the charges from VendorID-1 still keeping it lesser than VendorID-2 will greatly boost the revenue.