Report: Optimising NYC Taxi Operations

Name: Ashish Kumar Tewari

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

1. Data Preparation

1.1. Loading the dataset

1.1.1. Sample the data and combine the files

```
# Import the libraries you will be using for analysis

from google.colab import auth
auth.authenticate_user()

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

# Take a small percentage of entries from each hour of every date.
# Iterating through the monthly data:
#   read a month file -> day -> hour: append sampled data -> move to next
hour -> move to next day after 24 hours -> move to next month file
# Create a single dataframe for the year combining all the monthly data

# Select the folder having data files
import os

# Select the folder having data files
os.chdir('/content/drive/My
Drive/contents/Assignments/EDA/data_NYC_Taxi/trip_records')

# Create a list of all the twelve files to read
file_list = os.listdir()

# initialise an empty dataframe
df = pd.DataFrame()

print("file_list",file_list)
```

```
# iterate through the list of files and sample one by one:
for file_name in file_list:
    try:
        # file path for the current file
        file_path = os.path.join(os.getcwd(), file_name)

        # Reading the current file
        current_df = pd.read_parquet(file_path)

        # Convert the 'tpep_pickup_datetime' column to datetime
        current_df['tpep_pickup_datetime'] =
pd.to_datetime(current_df['tpep_pickup_datetime'])


        # We will store the sampled data for the current date in this df by
appending the sampled data from each hour to this
        # After completing iteration through each date, we will append this
data to the final dataframe.
        sampled_data = pd.DataFrame()
        dateList = current_df['tpep_pickup_datetime'].dt.date.unique()
        print("date list == ", dateList)

        # Loop through dates and then loop through every hour of each date
        for date in current_df['tpep_pickup_datetime'].dt.date.unique():
            print("date",date)

            # Iterate through each hour of the selected date
            for hour in range(24):
                # Filter the data for the current date and hour
                print("hour",hour)
                hour_data =
current_df[(current_df['tpep_pickup_datetime'].dt.date == date) &
(current_df['tpep_pickup_datetime'].dt.hour == hour)]

                # Sample 5% of the hourly data randomly
                hourly_sample = hour_data.sample(frac =
0.05,random_state=42)

                # add data of this hour to the dataframe
                sampled_data = pd.concat([sampled_data, hourly_sample])


        # Concatenate the sampled data of all the dates to a single
```

```
dataframe
        df = pd.concat([df, sampled_data])
        df.info()

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

Load the data into one file

```
# Store the df in csv/parquet
df.to_parquet('sampledData.parquet')
```

Read the final sampled data from saved file

```
# Load the new data file

# Select the folder having data files

import os

os.chdir('/content/drive/My
Drive/contents/Assignments/EDA/data_NYC_Taxi/trip_records/')

df_sampled = pd.read_parquet('sampledData.parquet')
```

2. Data Cleaning

2.1. Fixing Columns

2.1.1. Fix the index

```
# fix the index
df_sampled.reset_index(drop=True, inplace=True)
```

2.1.2. Combine the two airport_fee columns

```
 # if Airport_fee column is blank then fill that value with airport_fee
```

```
column value if it's non null otherwise fill it with 0
df_sampled['Airport_fee'].fillna(df_sampled['airport_fee'], inplace=True)


# drop rows where Airport_fee is NaN and passenger count is NaN
df_filter = df_sampled[(df_sampled['Airport_fee'].isna()) &
(df_sampled['passenger_count'].isna())]
df_sampled.drop(df_filter.index, inplace=True)


# check if Airport_fee null values has decreased or not
df_sampled.isnull().sum()*100/df_sampled.shape[0]

# drop airport_fee column
df_sampled.drop('airport_fee', axis=1, inplace=True)

# rename Airport_fee to airport_fee
df_sampled.rename(columns={'Airport_fee': 'airport_fee'}, inplace=True)
```

2.2. Handling Missing Values

2.2.1. Find the proportion of missing values in each column

```
df_sampled.isna().sum()*100/df_sampled.shape[0]
```

	0
VendorID	0.000000
tpep_pickup_datetime	0.000000
tpep_dropoff_datetime	0.000000
passenger_count	3.420903
trip_distance	0.000000
RatecodeID	3.420903
store_and_fwd_flag	3.420903
PULocationID	0.000000
DOLocationID	0.000000
payment_type	0.000000
fare_amount	0.000000
extra	0.000000
mta_tax	0.000000
tip_amount	0.000000
tolls_amount	0.000000
improvement_surcharge	0.000000
total_amount	0.000000
congestion_surcharge	3.420903
airport_fee	92.170270
Airport_fee	11.250633

2.2.2. Handling missing values in passenger_count

```
df_sampled[df_sampled['passenger_count'].isna()].shape[0]

# remove rows where passenger count is 0
df_sampled = df_sampled[df_sampled['passenger_count'] != 0]
```

2.2.3. Handle missing values in RatecodeID

```
df_sampled['RatecodeID'].value_counts()
```

2.2.4. Impute NaN in congestion_surcharge

```
df_sampled['congestion_surcharge'].fillna(df['congestion_surcharge'].mean())
```

2.3. Handling Outliers and Standardising Values

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

- Payment Type == 0 is an invalid payment type and needs to be standardised with the mode value

```
df_sampled = df_sampled[df_sampled['payment_type'] > 0]
```

- Trip distance can not be zero where PULocationID and DOLocationID are different
- Trip Distances which are above 99.99 percentile can be considered as outliers and can be removed

```
df_sampled = df_sampled[~((df_sampled['trip_distance'] == 0) &
(df_sampled['PULocationID'] == df_sampled['DOLocationID']))]

distance_threshold = np.percentile(df_sampled['trip_distance'], 99.99)
df_sampled = df_sampled[df_sampled['trip_distance'] < distance_threshold]
```

- Tip Amount can only be a fraction of the Fare Amount, hence taking a ratio of tip_amount and fare amount, can give us outliers which need to be discarded

```
tip_per_fare_threshold =
np.percentile(df_sampled['tip_amount']/df_sampled['fare_amount'], 99.99)
df_sampled[df_sampled['tip_amount'] > tip_per_fare_threshold]
```

- remove where passenger_count > 6

```
df_sampled = df_sampled[df_sampled['passenger_count'] <= 6]
df_sampled['passenger_count'].value_counts()
```

- if tripDistance=0 and fareAmount=0 but the pickup and dropoff zones are different. Then there is some mismatch. Drop those records

```
df_filtered = df_sampled[(df_sampled['trip_distance'] == 0) &
(df_sampled['fare_amount'] == 0) & (df_sampled['PULocationID'] !=
df_sampled['DOLocationID'])]

# drop these rows
df_sampled.drop(df_filtered.index, inplace=True)
```

- entries where trip_distance is 0 and fare_amount > 300 and pickup drop zones are same

```
df_filtered_distance = df_sampled[(df_sampled['trip_distance'] == 0) &
(df_sampled['fare_amount'] > 300) & (df_sampled['PULocationID'] ==
df_sampled['DOLocationID'])]

df_sampled.drop(df_filtered_distance.index, inplace=True)
```

- replace RatecodeID to 1 if RatecodeID is 99

```
df_sampled['RatecodeID'] = df_sampled['RatecodeID'].apply(lambda x: 1.0 if
x == 99.0 else x)
```

- remove outliers where amount per mile is more than 25000

```
df_sampled =
df_sampled[df_sampled['fare_amount']/df_sampled['trip_distance'] < 25000]
```

3. Exploratory Data Analysis

3.1. General EDA: Finding Patterns and Trends

3.1.1. Classify variables into categorical and numerical

Below are the Numerical values-

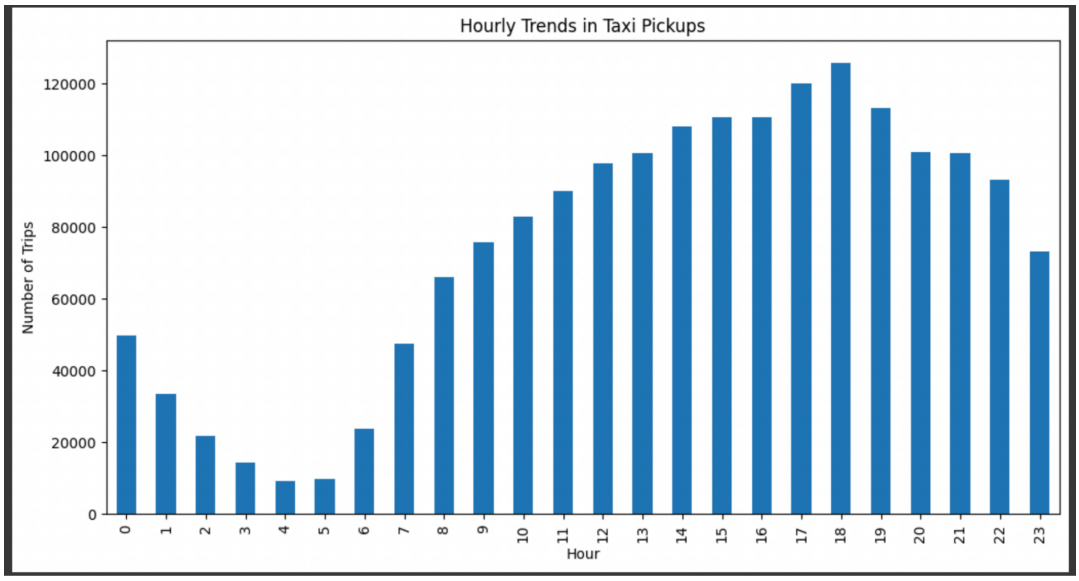
VendorID: Categorical Value
tpep_pickup_datetime: Numerical Value
tpep_dropoff_datetime: Numerical Value
passenger_count: Categorical Value
trip_distance: Numerical Value
RatecodeID: Categorical Value
PULocationID: Categorical Value
DOLocationID: Categorical Value
payment_type: Categorical Value
pickup_hour: Numerical Value
trip_duration: Numerical Value

fare_amount : Numerical Value
extra : Numerical Value
mta_tax : Numerical Value
tip_amount : Numerical Value
tolls amount : Numerical Value
iimprovement_surcharge : Numerical Value
total_amount : Numerical Value
congestion_surcharge : Numerical Value
airport_fee : Numerical Value

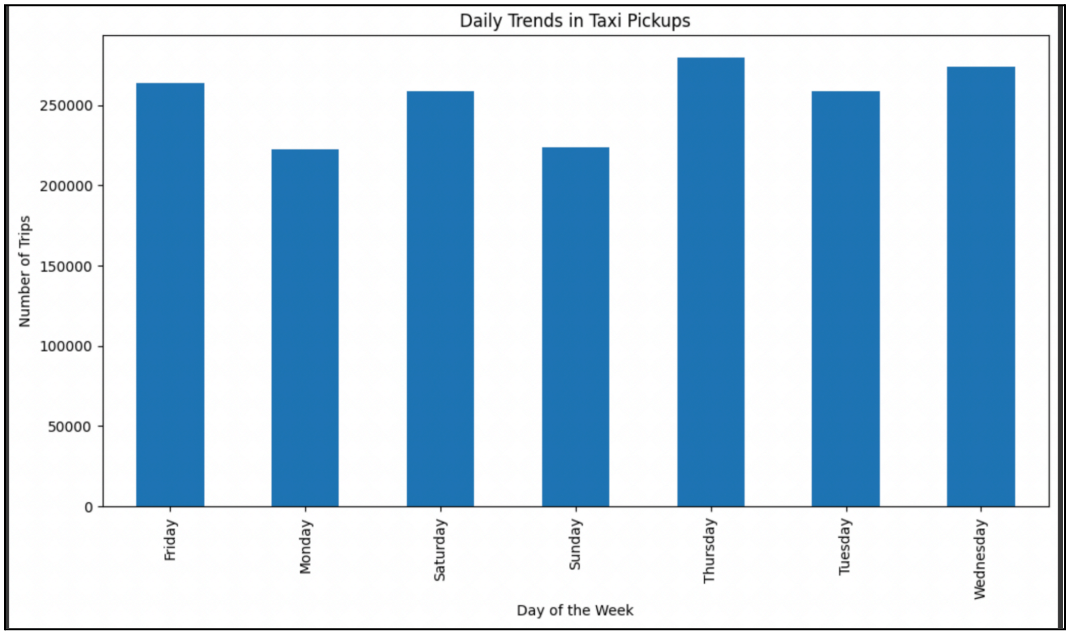
3.1.2. Analyse the distribution of taxi pickups by hours, days of the week, and months

```
# Find and show the hourly trends in taxi pickups
# create a new column pickup_hour
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour

# find the hourly trip
hourly_trips = df.groupby(['pickup_hour']).size()
# plot the trend for hourly trip
```

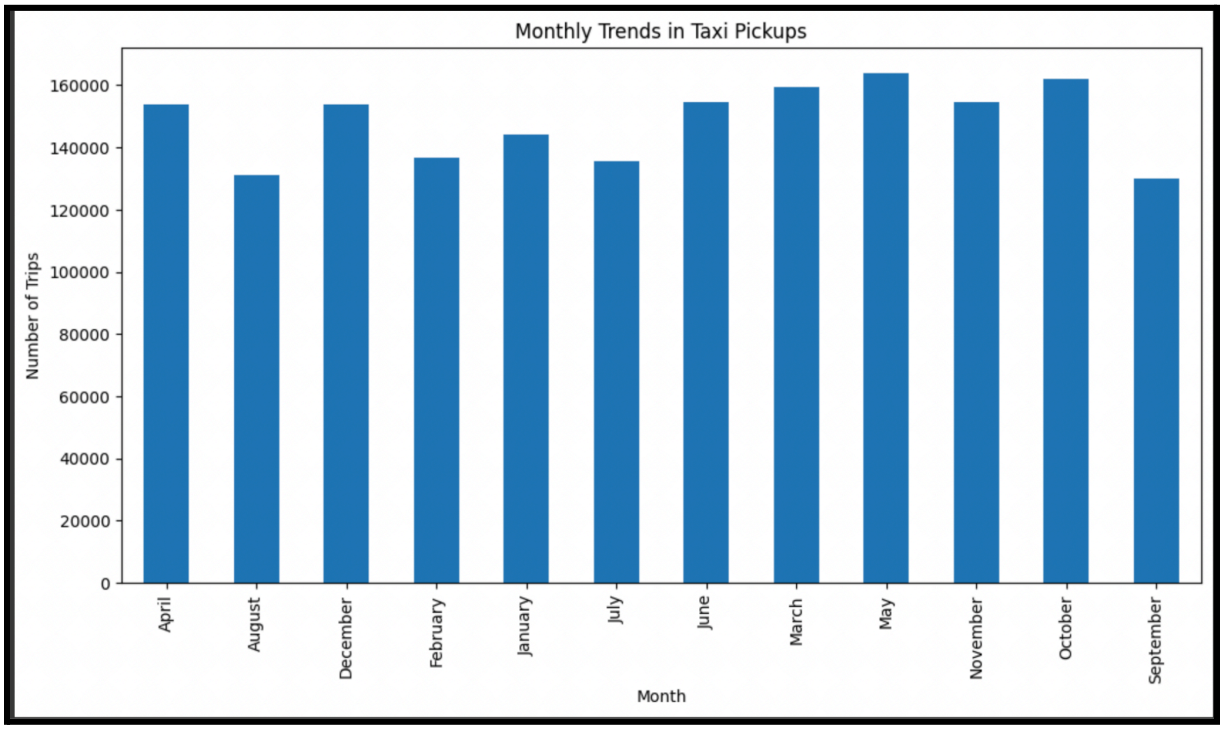
```
plt.figure(figsize=(12, 6))
hourly_trips.plot(kind="bar")
plt.title("Hourly Trends in Taxi Pickups")
plt.xlabel("Hour")
plt.ylabel("Number of Trips")
plt.show()
```




```
# Find days of the week for the daily trends in taxi pickups (days of the week)
df['pickup_day'] = df['tpep_pickup_datetime'].dt.day_name()
pickup_day_trends = df.groupby(['pickup_day']).size()

# show the daily trends in taxi pickups (days of the week)
plt.figure(figsize=(12,6))
pickup_day_trends.plot(kind="bar")
plt.title("Daily Trends in Taxi Pickups")
plt.xlabel("Day of the Week")
plt.ylabel("Number of Trips")
plt.show()
```




```
# Show the monthly trends in pickups
df['pickup_month'] = df['tpep_pickup_datetime'].dt.month_name()
pickup_month_trend = df.groupby(['pickup_month']).size()

plt.figure(figsize=(12,6))
pickup_month_trend.plot(kind="bar")
plt.title("Monthly Trends in Taxi Pickups")
plt.xlabel("Month")
plt.ylabel("Number of Trips")
plt.show()
```



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

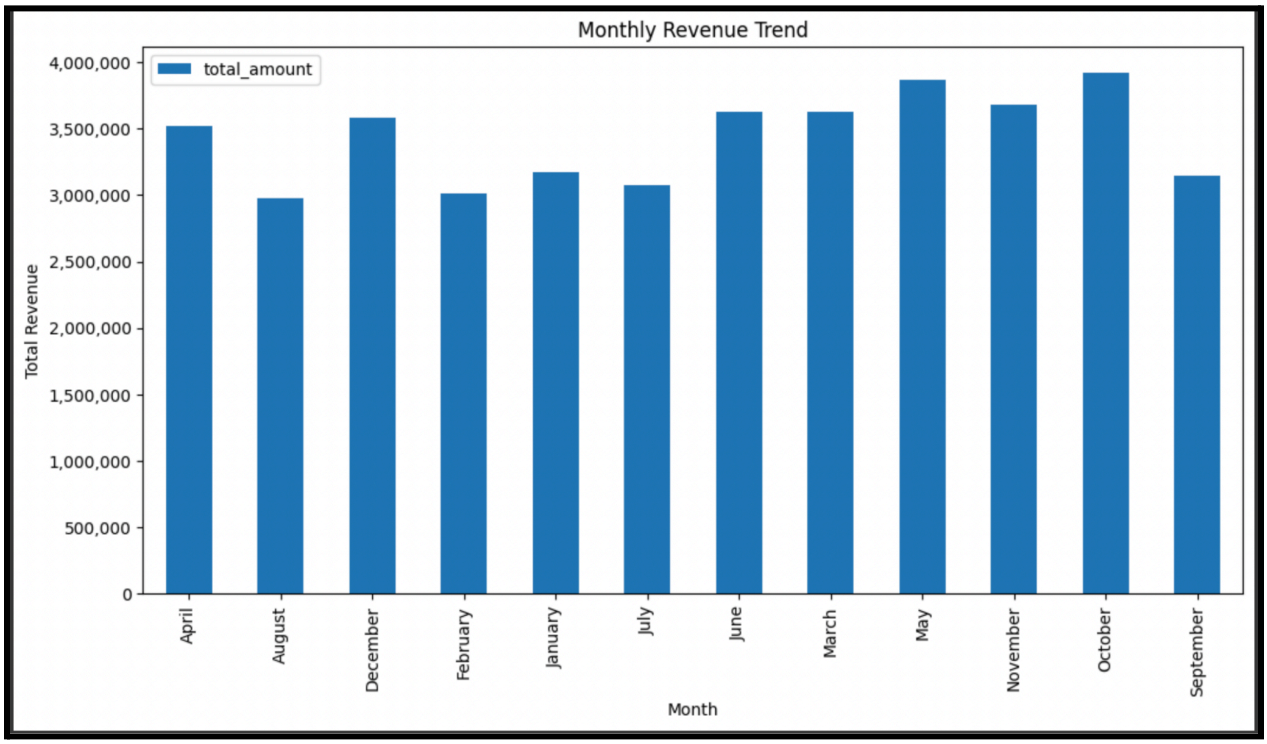
```
df_filtered = df_sampled[(df_sampled['trip_distance'] == 0) &
(df_sampled['fare_amount'] == 0) & (df_sampled['PULocationID'] !=
df_sampled['DOLocationID'])]

# drop these rows
df_sampled.drop(df_filtered.index, inplace=True)

# Fix columns with negative (monetary) values

df_negative_monetry_values = df_sampled[(df_sampled['fare_amount'] < 0)
| (df_sampled['total_amount'] < 0) | (df_sampled['tolls_amount'] < 0) |
(df_sampled['tip_amount'] < 0) | (df_sampled['extra'] < 0) |
(df_sampled['mta_tax'] < 0) | (df_sampled['improvement_surcharge'] < 0)
| (df_sampled['congestion_surcharge'] < 0) | (df_sampled['airport_fee']
```

```
< 0)]

# Drop the rows where
df_sampled.drop(df_negative_monetry_values.index, inplace=True)

# Create a df with non zero entries for the selected parameters.
df_zeros = df[(df['fare_amount']<=0) | (df['total_amount']<=0)]
df.drop(df_zeros.index, inplace=True)

# check if any zero fare_amount, total _amount and tip_amount is zero
but trip_distance is not zero
df[(df['fare_amount']<=0) & (df['tip_amount']<=0) &
(df['total_amount']<=0) & (df['trip_distance'] > 0)]

# remove all the rows where fare_amount, total _amount and tip_amount is
zero but trip_distance is not zero
df_zero_amount = df[(df['fare_amount']<=0) & (df['tip_amount']<=0) &
(df['total_amount']<=0) & (df['trip_distance'] > 0)]
df.drop(df_zero_amount.index, inplace=True)

#check if any rows where fare_amount, total _amount and tip_amount is
zero but trip_distance is not zero present
df[(df['fare_amount']<=0) & (df['tip_amount']<=0) &
(df['total_amount']<=0) & (df['trip_distance'] > 0)]
```

3.1.4. Analyse the monthly revenue trends

```
# Group data by month and analyse monthly revenue
import matplotlib.ticker as ticker

monthly_revenue =
df.groupby(['pickup_month'])['total_amount'].sum().reset_index()
ax = monthly_revenue.plot(kind="bar", x="pickup_month", y="total_amount",
figsize=(12, 6))
# round off y ticks otherwise if will appear as only 1.0, 2.0
ax.yaxis.set_major_formatter(ticker.FuncFormatter(lambda x, pos:
```

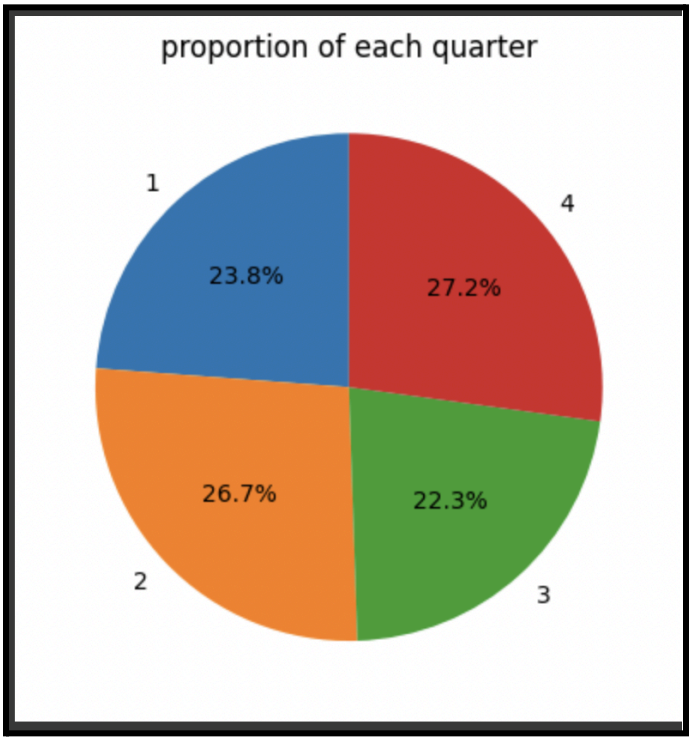
```
'{:,.0f}'.format(x)))
#increase ticks to be more clear on y axis
ax.yaxis.set_major_locator(ticker.MaxNLocator(10))
plt.title("Monthly Revenue Trend")
plt.xlabel("Month")
plt.ylabel("Total Revenue")
plt.show()
```



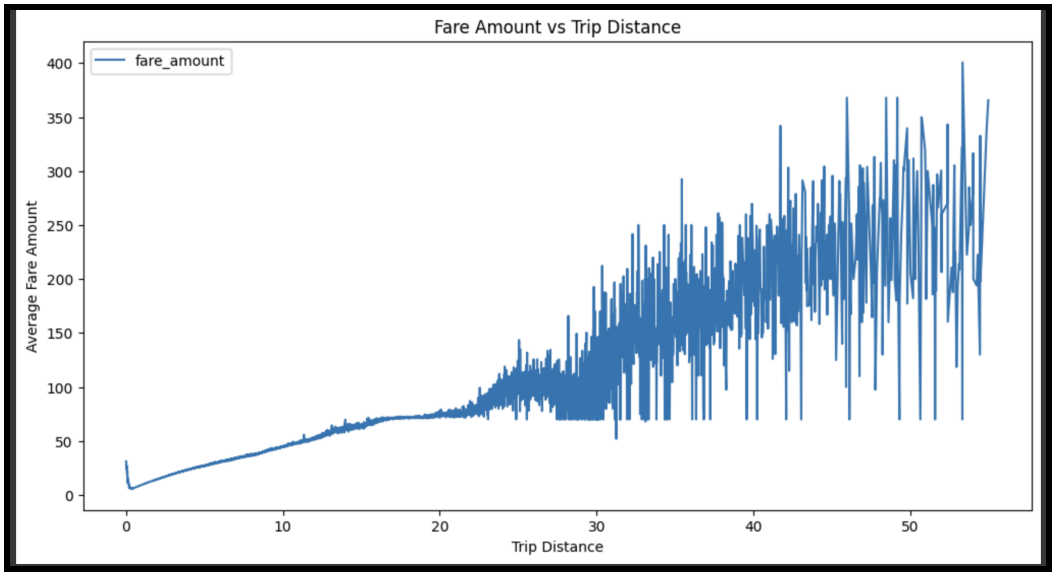
3.1.5. Find the proportion of each quarter's revenue in the yearly revenue

```
# Calculate proportion of each quarter

df['quarter'] = df['tpep_pickup_datetime'].dt.quarter

df_quarter_proportion =
df.groupby(['quarter'])['total_amount'].sum()*100/df['total_amount'].sum()
# df_quarter_proportion.head()
```

```
plt.pie(df_quarter_proportion, labels=df_quarter_proportion.index,
autopct='%1.1f%%', startangle=90)
plt.title("proportion of each quarter")
plt.show()
```

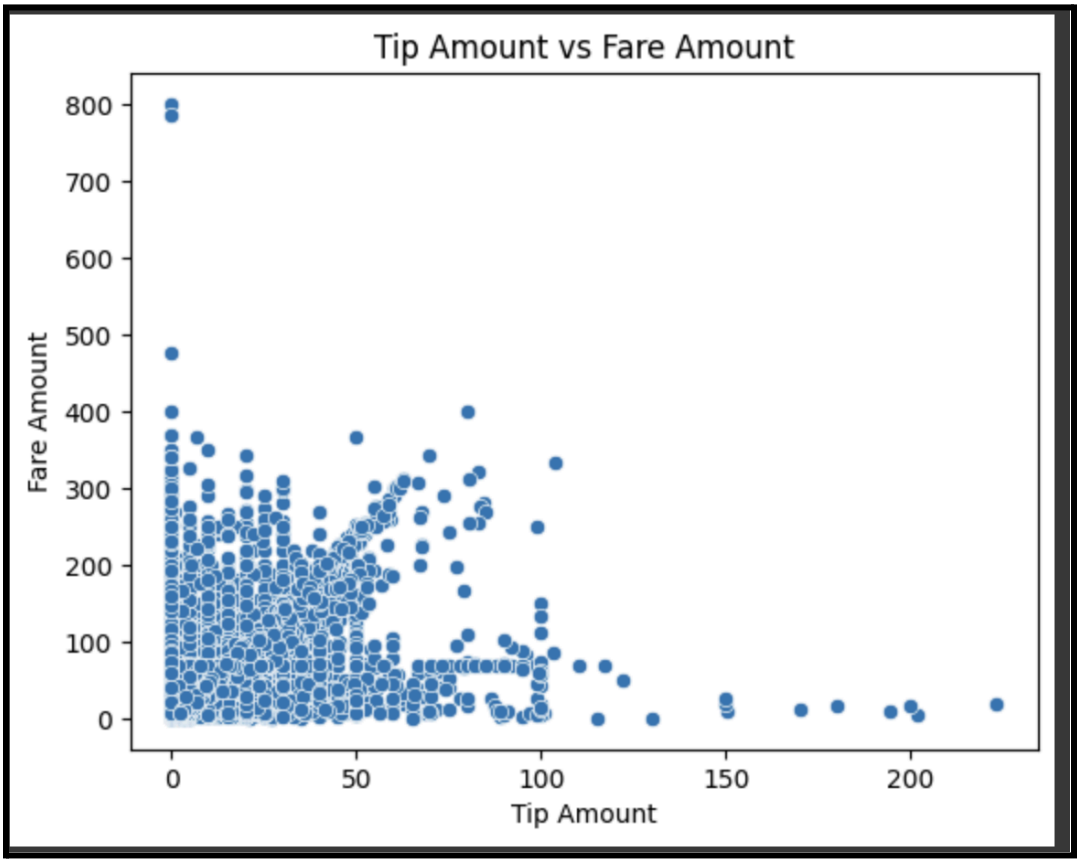


3.1.6. Analyse and visualise the relationship between distance and fare amount

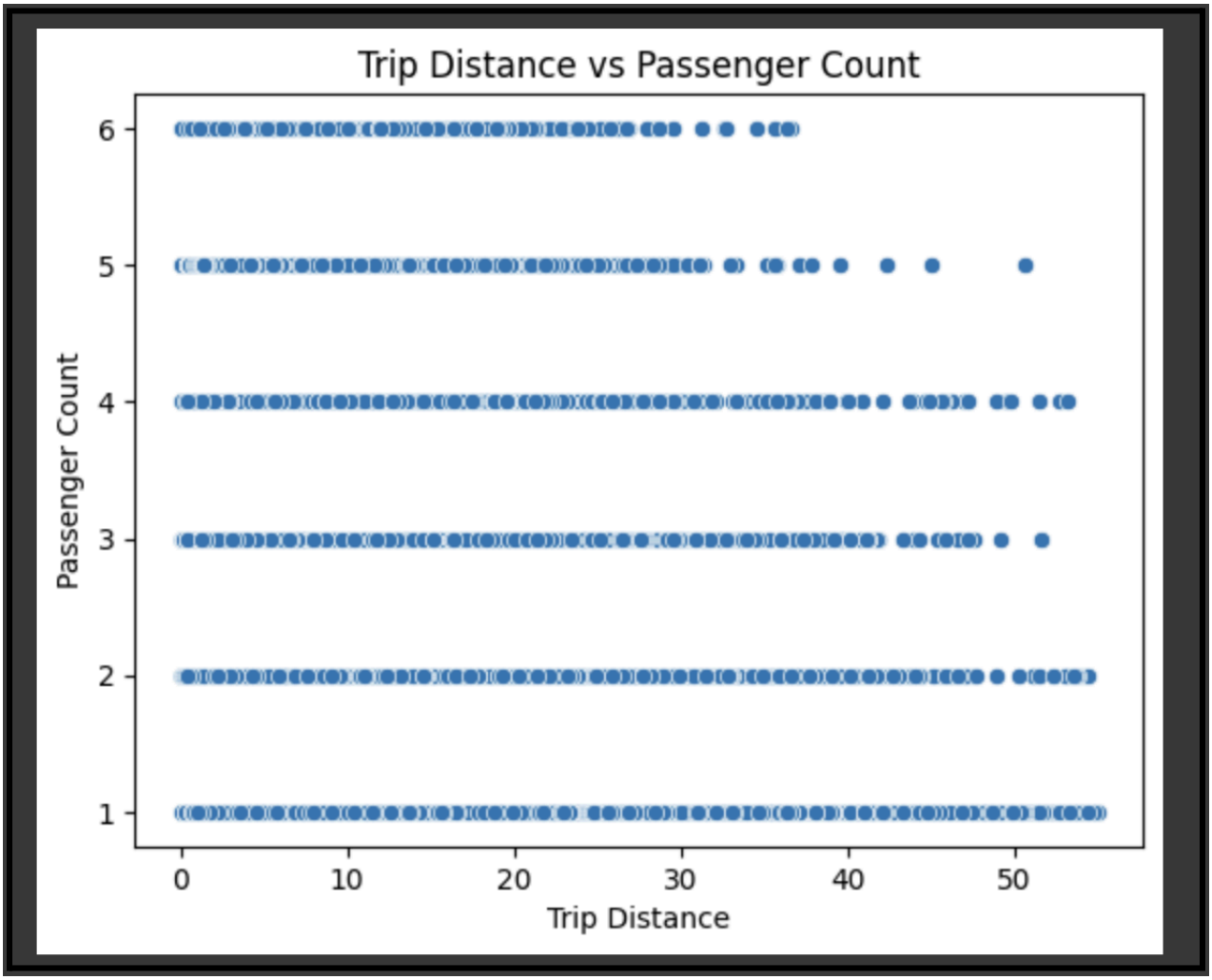


Correlation between trip distance and fare_amount : 0.9496716627835351

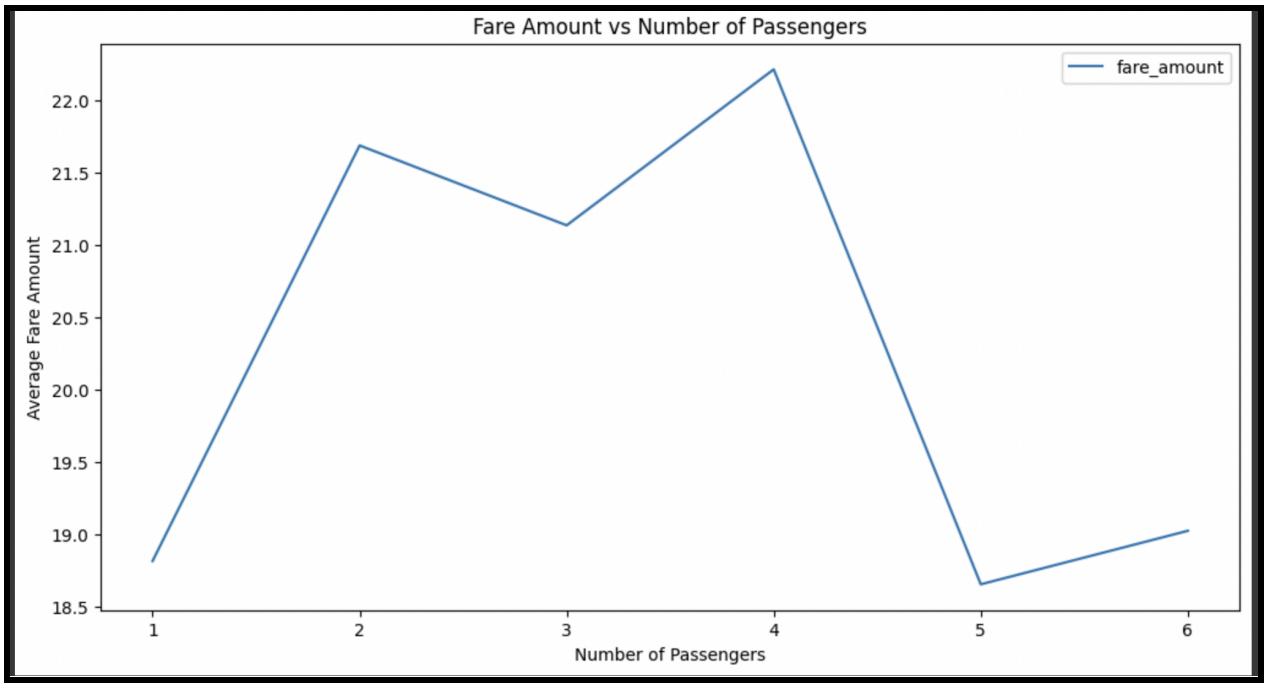
3.1.7. Analyse the relationship between fare/tips and trips/passengers

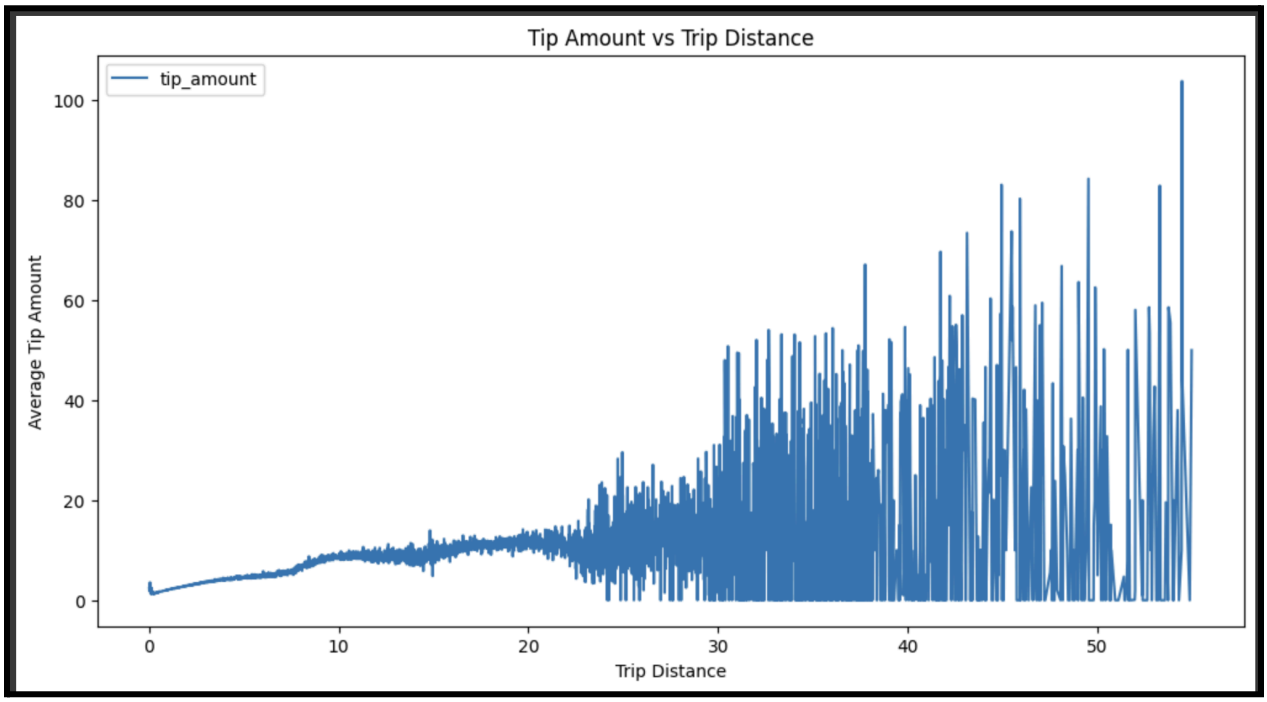


Correlation between tip amount and fare amount: 0.6097554737165118

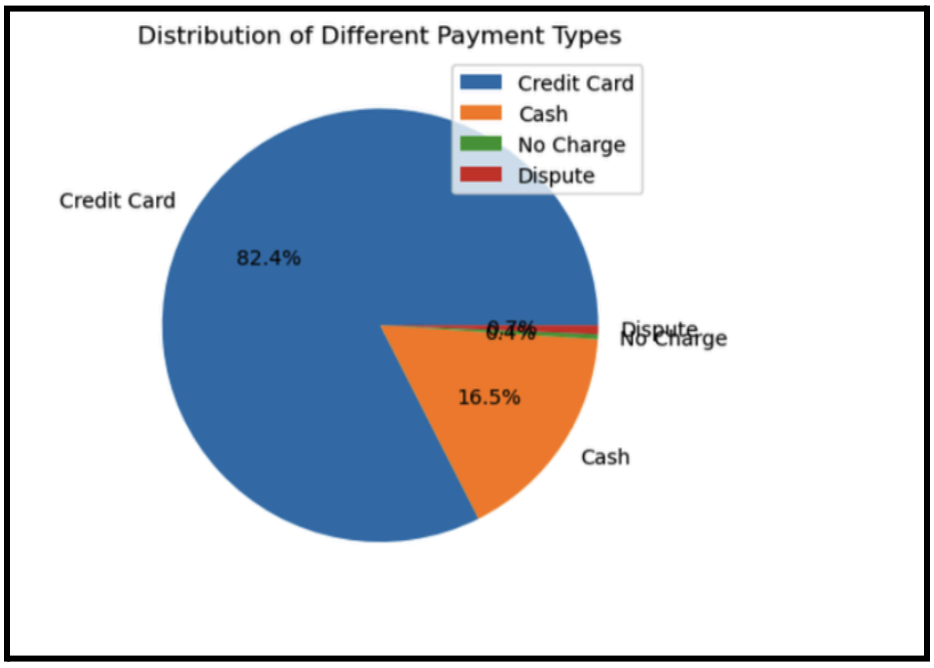


Correlation between trip distance and passenger: 0.03816104561144689


Fare Amount vs Number of Passengers
fare_amount
Average Fare Amount
22.0
21.5
21.0
20.5
20.0
19.5
19.0
18.5
1
2
3
4
5
6
Number of Passengers

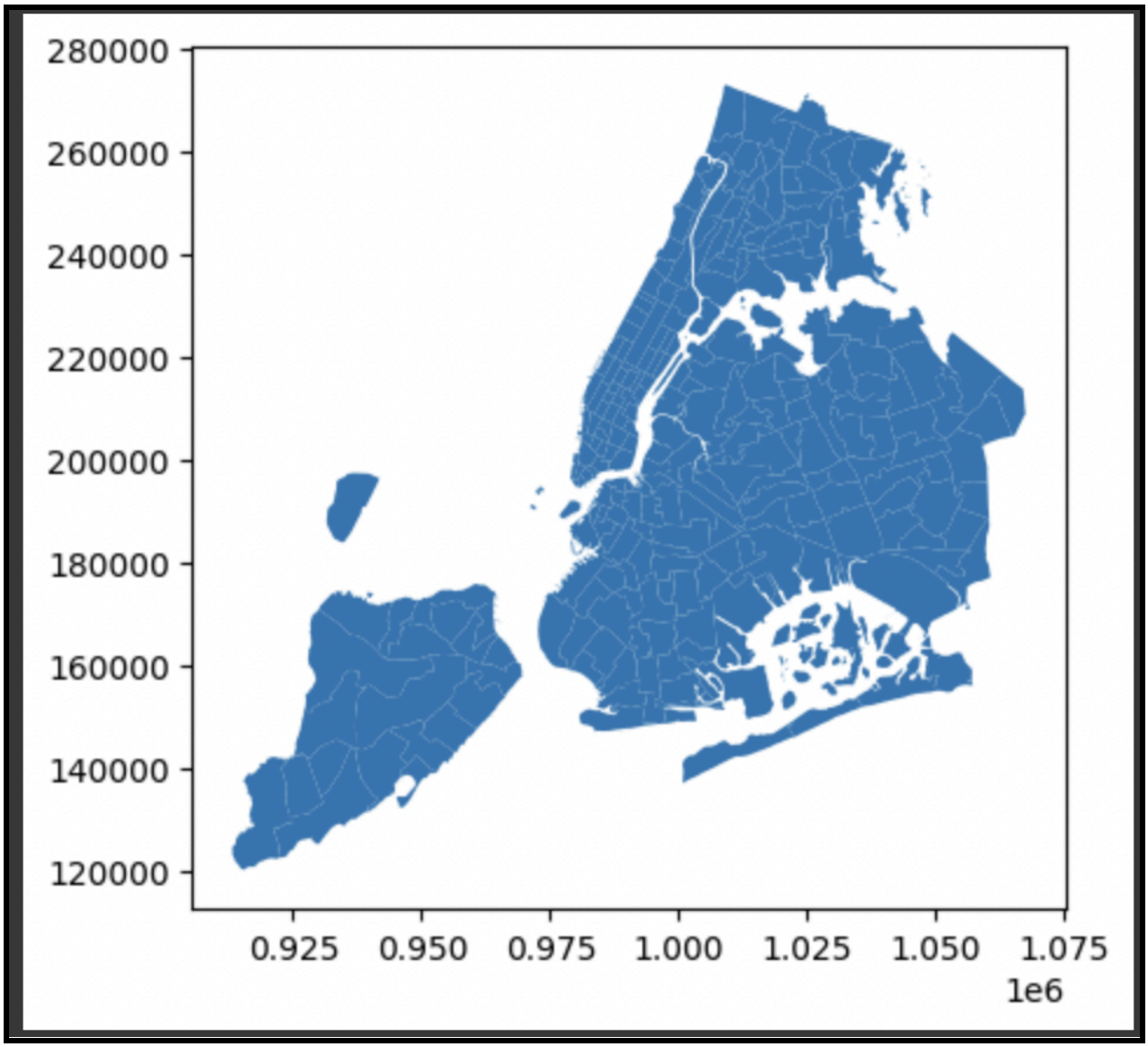

Tip Amount vs Trip Distance
tip_amount
Average Tip Amount
100
80
60
40
20
0
0
10
20
30
40
50
Trip Distance

3.1.8. Analyse the distribution of different payment types



3.1.9. Load the taxi zones shapefile and display it

```
import geopandas as gpd
import os

# Read the shapefile using geopandas
os.chdir('/content/drive/My
Drive/contents/Assignments/EDA/data_NYC_Taxi/taxi_zones/')
zones = gpd.read_file('taxi_zones.shp') # read the .shp file using gpd
zones.head()
```



3.1.10. Merge the zone data with trips data

```
# Merge zones and trip records using locationID and PULocationID


df_merged = df.merge(zones, left_on='PULocationID', right_on='LocationID')

df_merged.shape
```

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

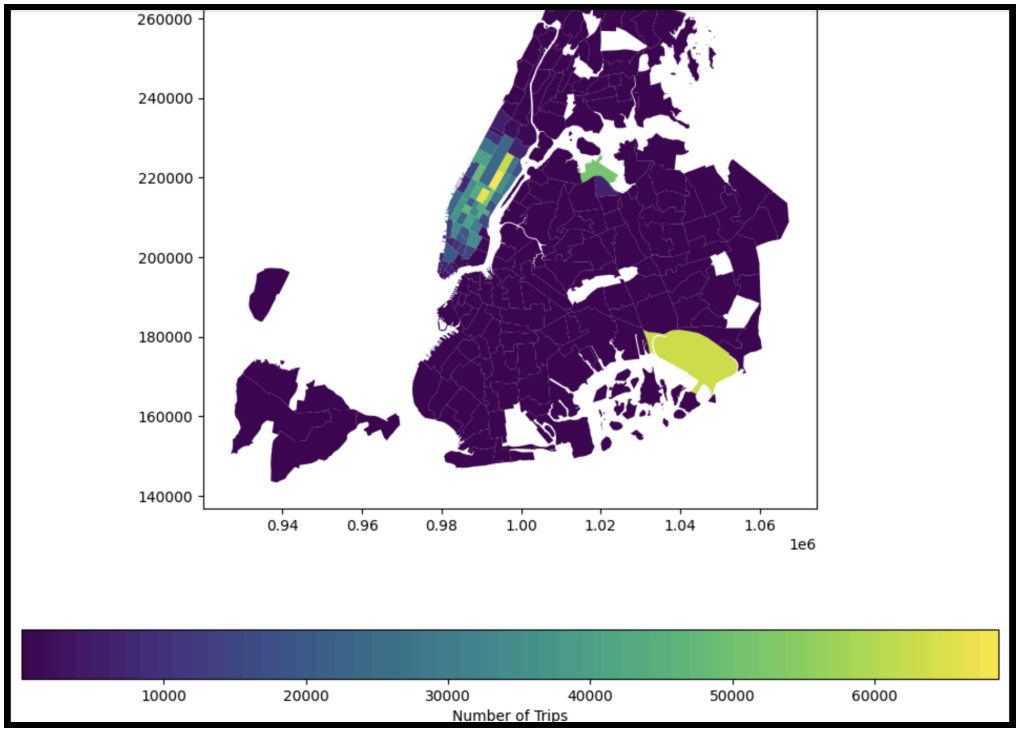
```
# Group data by location and calculate the number of trips

df_trips_per_location =
df_merged.groupby(['LocationID']).size().reset_index(name='trip_count')
df_trips_per_location.head()
```

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

```
df_merged_zones_with_tripCount = zones.merge(df_trips_per_location,
left_on='LocationID', right_on='LocationID')
df_merged_zones_with_tripCount.head()
zones = df_merged_zones_with_tripCount
```

3.1.13. Plot a map of the zones showing number of trips

```
sorted_trip_zones = zones.sort_values(by='trip_count', ascending=False)

fig, ax = plt.subplots(1, 1, figsize = (12, 10))
# sorted_trip_zones
sorted_trip_zones.plot('trip_count', ax=ax, legend=True,
legend_kwds={'label': "Number of Trips", 'orientation': "horizontal"})
plt.show()
```



3.1.14. Conclude with results

1. Busiest hours: 5 PM - 6 PM, busiest days: Thursday & Wednesday, busiest months: May & Oct

2. Trends in quarterly revenue : 4th and 2nd are higher

3. Fare is lower when pax count is 3 compared to 2. Short trip has more avg fare amount, this might be becoz of extra charges during the busiest hour.

4. Tip amount is +ve correlated with trip distance, but has a medium correlation. So ppl are likely to pay more tips if the trip distance is large.

5. Midtown centre, Upper East Side South, Upper East Side North
JFK Airport, upper East Side South, Midtown centre are the busiest zone.

3.2. Detailed EDA: Insights and Strategies

3.2.1. Identify slow routes by comparing average speeds on different routes

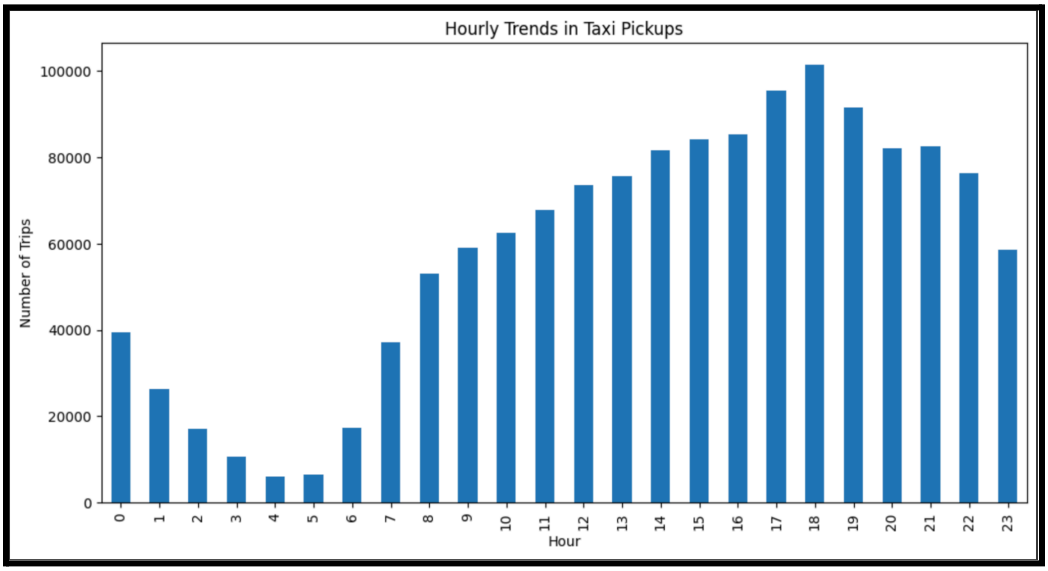
```
# Find routes which have the slowest speeds at different times of the day
df['speed'] = df['trip_distance']/df['trip duration']
# some of the speed is NaN.
# case 1: where DOLocationID = 264 as DOLocationID is invalid.
# case 2: where PULocationID and PULocationID is same.


# remove rows where speed is NaN and DOLocationID = 264
df_na_speed_to_be_removed = df[(df['speed'].isna()) & (df['DOLocationID']
== 264)]
df.drop(df_na_speed_to_be_removed.index, inplace=True)

# replace NaN speed with 0
df['speed'].fillna(0, inplace=True)

# Find the routes which have the slowest speeds at different times of the
day
df.groupby(['PULocationID',
'DOLocationID','pickup_hour'])['speed'].mean().sort_values()
```

			speed
PULocationID	DOLocationID	pickup_hour	
243	264	17	0.000130
45	45	10	0.000991
162	162	4	0.001080
70	138	6	0.001429
237	238	4	0.002006
...	...	...	...
141	264	20	inf
211	211	16	inf
162	13	7	inf
114	114	10	inf
164	164	17	inf

100780 rows × 1 columns

3.2.2. Calculate the hourly number of trips and identify the busy hours

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

hourly_trips = df.groupby(['pickup_hour']).size()
plt.figure(figsize=(12, 6))
hourly_trips.plot(kind="bar")
plt.title("Hourly Trends in Taxi Pickups")
plt.xlabel("Hour")
plt.ylabel("Number of Trips")
plt.show()
```



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

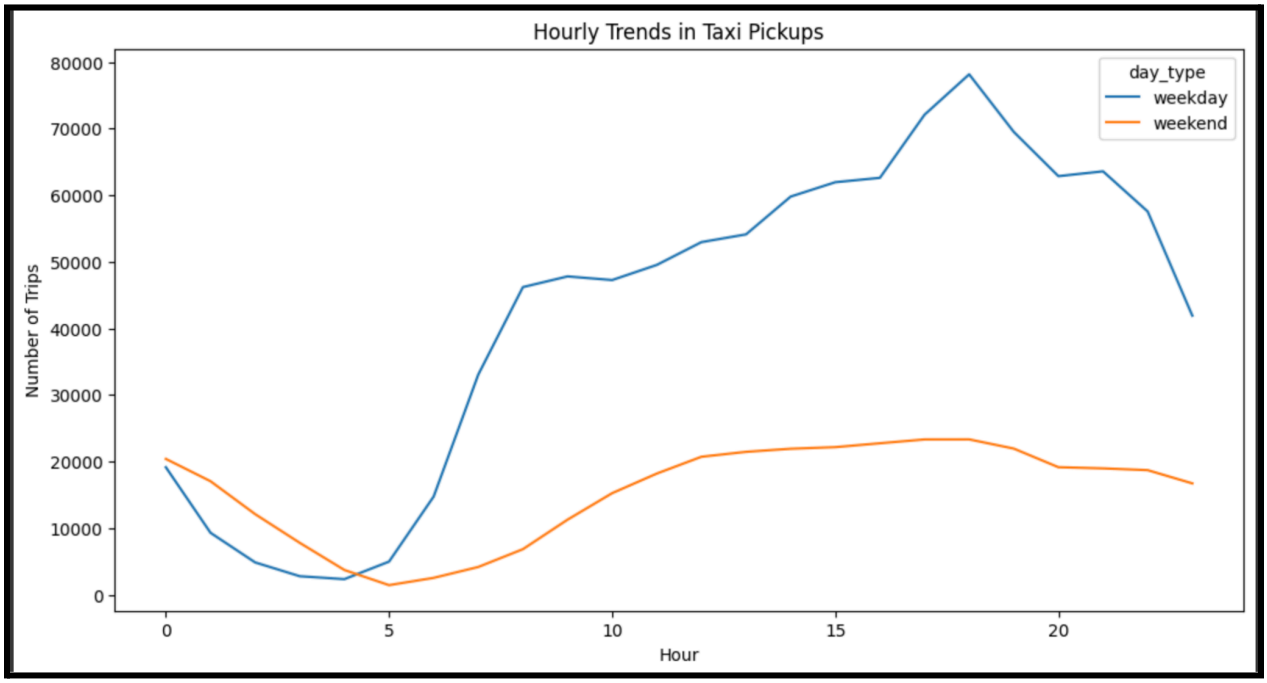
```
# Scale up the number of trips

# Fill in the value of your sampling fraction and use that to scale up the
numbers
df_hourly_trip =
hourly_trips.reset_index(name='trip_count').sort_values(by='trip_count',
ascending=False)
df_hourly_trip['trip_count'] = df_hourly_trip['trip_count']/0.05
df_hourly_trip
```

	pickup_hour	trip_count
18	18	2029820.0
17	17	1908320.0
19	19	1829200.0
16	16	1707320.0
15	15	1682500.0
21	21	1651440.0
20	20	1640420.0
14	14	1634520.0
22	22	1525520.0
13	13	1511740.0
12	12	1473660.0
11	11	1354720.0
10	10	1250460.0
9	9	1181780.0
23	23	1173060.0
8	8	1061000.0
0	0	790880.0
7	7	744880.0
1	1	527200.0

3.2.4. Compare hourly traffic on weekdays and weekends

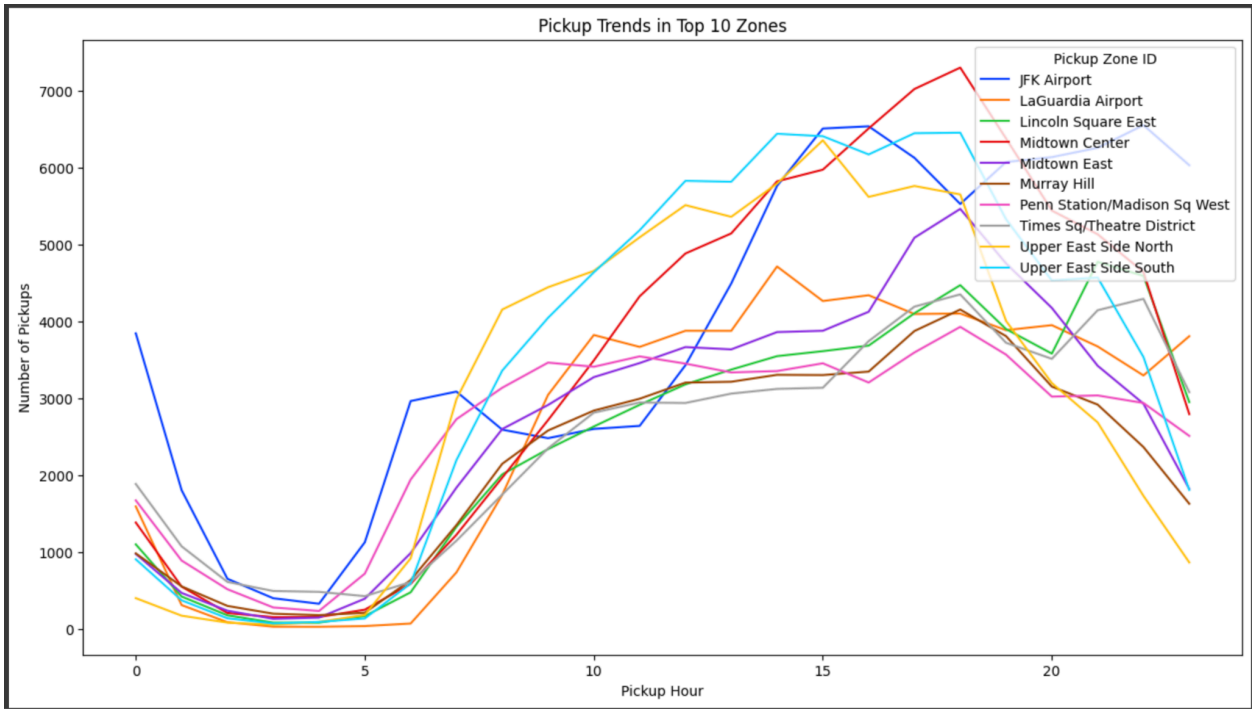
```
# Compare traffic trends for the weekdays and weekends

# add new column day_type
df['day_type'] = df['pickup_day'].apply(lambda x: 'weekend' if x in
['Sunday', 'Saturday'] else 'weekday')


df_hourly_traffice_based_day_type = df.groupby(['day_type',
'pickup_hour'])['tpep_pickup_datetime'].count().reset_index(name='trip_coun
t')

# plot the graph using sns. df_hourly_traffice_based_day_type.plot doesn't
take hue as arg
plt.figure(figsize=(12, 6))
plt.title("Hourly Trends in Taxi Pickups")
plt.xlabel("Hour")
plt.ylabel("Number of Trips")
```

```
sns.lineplot(data=df_hourly_traffice_based_day_type, x='pickup_hour',
y='trip_count', hue='day_type')
```



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

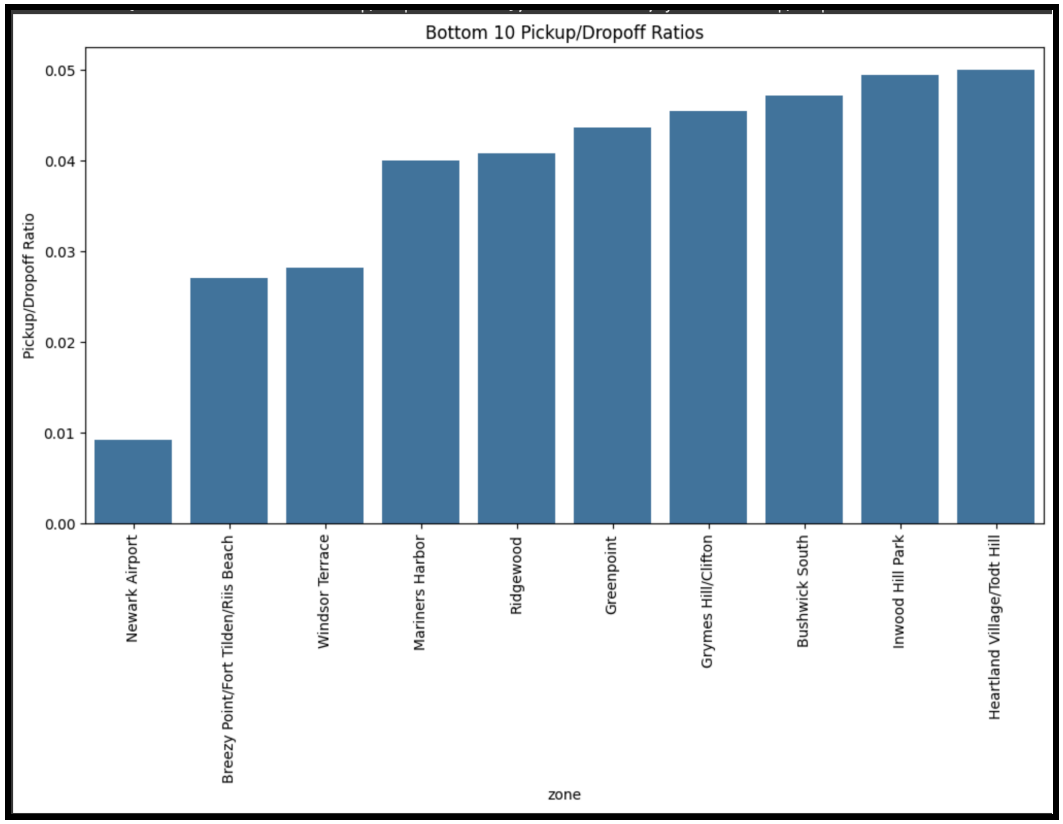
```
# Find top 10 pickup and dropoff zones
df_pickup_zones_by_hour =
df.groupby(['PULocationID','pickup_hour'])['tpep_pickup_datetime'].count().
reset_index(name='pickup_count')

# find the total hourly pickup for each zone
mean_hourly_pickups_by_zone =
df_pickup_zones_by_hour.groupby(['PULocationID'])['pickup_count'].mean().re
set_index(name="mean_hourly_pickups")

# sort based on total_hourly_pickups and find the top 10 zones
top_10_zones_by_total_hourly_pickups =
mean_hourly_pickups_by_zone.sort_values(by='mean_hourly_pickups',
ascending=False).head(10)

#show the top 10 zone which has high total hourly pickup
top_10_zones_by_total_hourly_pickups
```

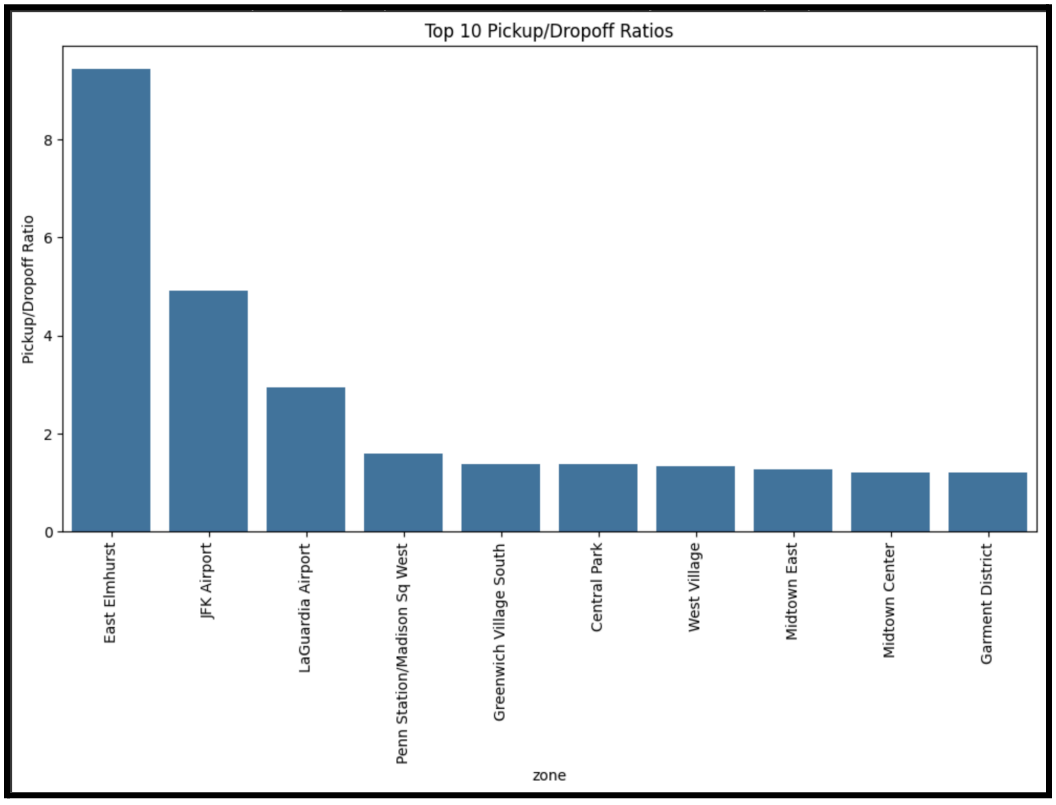


	zone	mean_hourly_pickups
0	JFK Airport	3911.875000
1	Upper East Side South	3541.750000
2	Midtown Center	3499.458333
3	Upper East Side North	3154.416667
4	Midtown East	2674.000000
5	LaGuardia Airport	2624.583333
6	Penn Station/Madison Sq West	2577.958333
7	Times Sq/Theatre District	2492.291667
8	Lincoln Square East	2476.666667
9	Murray Hill	2215.791667

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

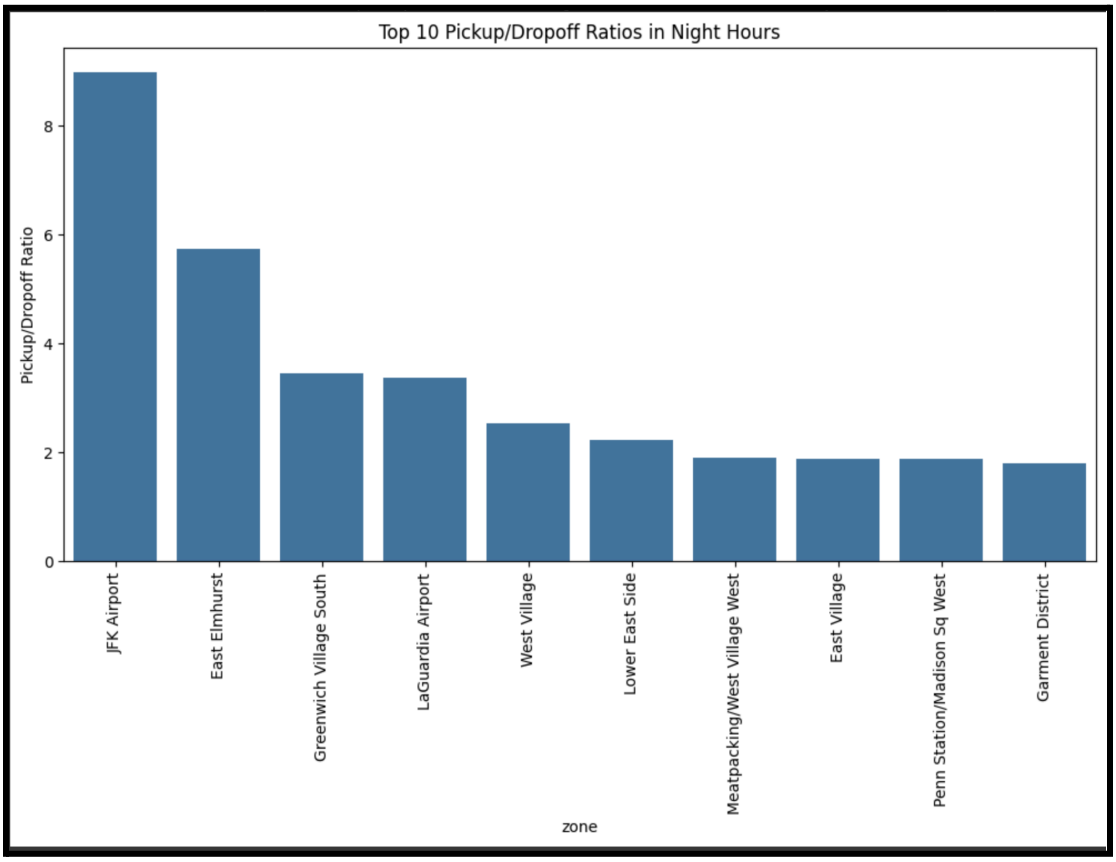
```
# Find the top 10 and bottom 10 pickup/dropoff ratios
df_pickups_by_zones =
df.groupby(['PULocationID'])['tpep_pickup_datetime'].count().reset_index(na
me='pickup_count')
df_dropoffs_by_zones =
df.groupby(['DOLocationID'])['tpep_dropoff_datetime'].count().reset_index(n
ame='dropoff_count')

# merge df_pickups_by_zones and df_dropoffs_by_zones based on PULocationID
df_zones_by_pickups_and_dropoffs =
df_pickups_by_zones.merge(df_dropoffs_by_zones, how='inner',
right_on='DOLocationID', left_on='PULocationID', suffixes=('_pickup',
'_dropoff'))

# create a column for pickup_count/dropoff_count
df_zones_by_pickups_and_dropoffs['pickup_dropoff_ratio'] =
df_zones_by_pickups_and_dropoffs['pickup_count']/df_zones_by_pickups_and_dr
opoffs['dropoff_count']

# top 10 pickup/dropoff ratios
df_zones_by_pickups_and_dropoffs.sort_values(by='pickup_dropoff_ratio',
ascending=False).head(10)


# create a graph for the top 10 pickup/dropoff ratios
plt.figure(figsize=(12, 6))
plt.title("Top 10 Pickup/Dropoff Ratios")
plt.xlabel("PULocationID")
plt.ylabel("Pickup/Dropoff Ratio")
sns.barplot(df_zones_by_pickups_and_dropoffs.sort_values(by='pickup_dropoff
_ratio', ascending=False).head(10), x='PULocationID',
y='pickup_dropoff_ratio')
```


Bottom 10 Pickup/Dropoff Ratios
Pickup/Dropoff Ratio
0.00
0.01
0.02
0.03
0.04
0.05
Newark Airport
Breezy Point/Fort Tilden/Riis Beach
Windsor Terrace
Mariners Harbor
Ridgewood
Greenpoint
Grymes Hill/Clifton
Bushwick South
Inwood Hill Park
Heartland Village/Todt Hill
zone


Top 10 Pickup/Dropoff Ratios
Pickup/Dropoff Ratio
0
2
4
6
8
East Elmhurst
JFK Airport
LaGuardia Airport
Penn Station/Madison Sq West
Greenwich Village South
Central Park
West Village
Midtown East
Midtown Center
Garment District
zone

3.2.7. Identify the top zones with high traffic during night hours

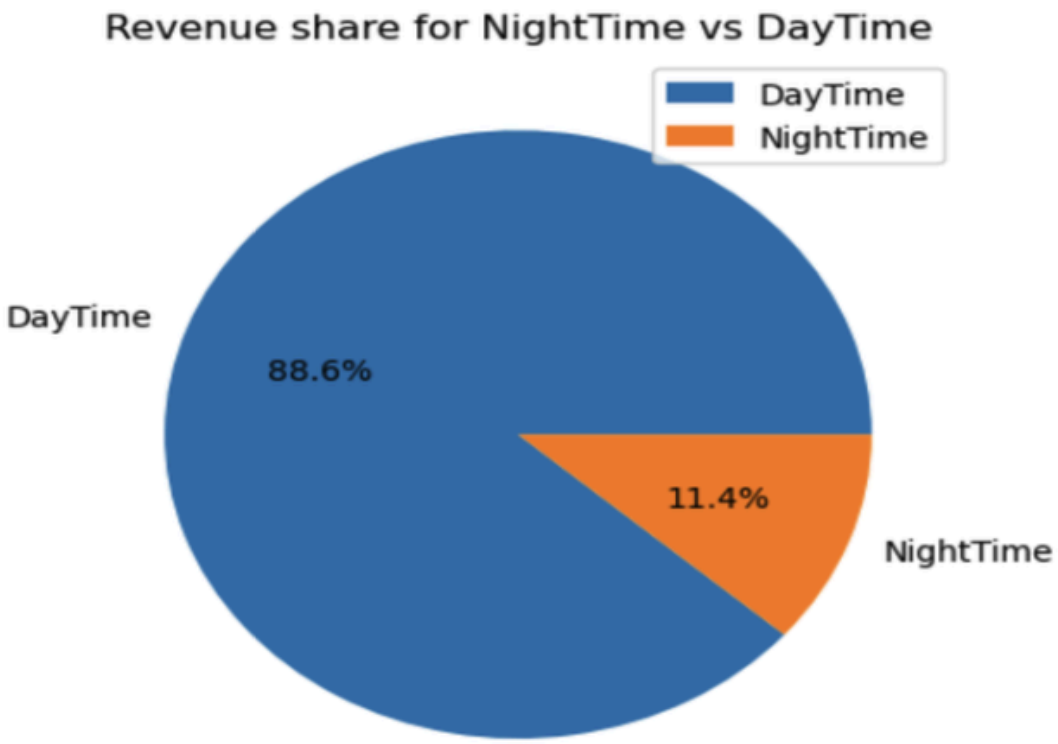
```
# During night hours (11pm to 5am) find the top 10 pickup and dropoff zones
# Note that the top zones should be of night hours and not the overall top
zones

# filter the data by ight hours (11pm to 5am)
df_by_night_hour = df[(df['pickup_hour'] >= 23) | (df['pickup_hour'] <= 5)]

# zones by pickups
df_night_hour_zones_by_pickups =
df_by_night_hour.groupby(['PULocationID'])['tpep_pickup_datetime'].count().
reset_index(name='pickup_count')
# zones by drop off
df_night_hour_zones_by_dropoffs =
df_by_night_hour.groupby(['DOLocationID'])['tpep_dropoff_datetime'].count()
.reset_index(name='dropoff_count')

# merge zones by ID
df_night_hour_zones_by_pickups_and_dropoffs =
df_night_hour_zones_by_pickups.merge(df_night_hour_zones_by_dropoffs,
how='inner', left_on='PULocationID', right_on='DOLocationID',
suffixes=('_pickup', '_dropoff'))

# add a column for pickup and dropffs ratio
df_night_hour_zones_by_pickups_and_dropoffs['pickup_dropoff_ratio'] =
df_night_hour_zones_by_pickups_and_dropoffs['pickup_count']/df_night_hour_z
ones_by_pickups_and_dropoffs['dropoff_count']

# find the top 10 zones by pickup and dropffs ratio
df_night_hour_top_10_zones_by_pickups_and_dropoffs =
df_night_hour_zones_by_pickups_and_dropoffs.sort_values(by='pickup_dropoff_
ratio', ascending=False).head(10).reset_index(drop=True)
df_night_hour_top_10_zones_by_pickups_and_dropoffs

# create a graph of pickup_dropoff_ratio of top 10 zones nightly
plt.figure(figsize=(12, 6))
plt.title("Top 10 Pickup/Dropoff Ratios in Night Hours")
plt.xlabel("PULocationID")
plt.ylabel("Pickup/Dropoff Ratio")
sns.barplot(df_night_hour_top_10_zones_by_pickups_and_dropoffs,
x='PULocationID', y='pickup_dropoff_ratio')
```



3.2.8. Find the revenue share for nighttime and daytime hours

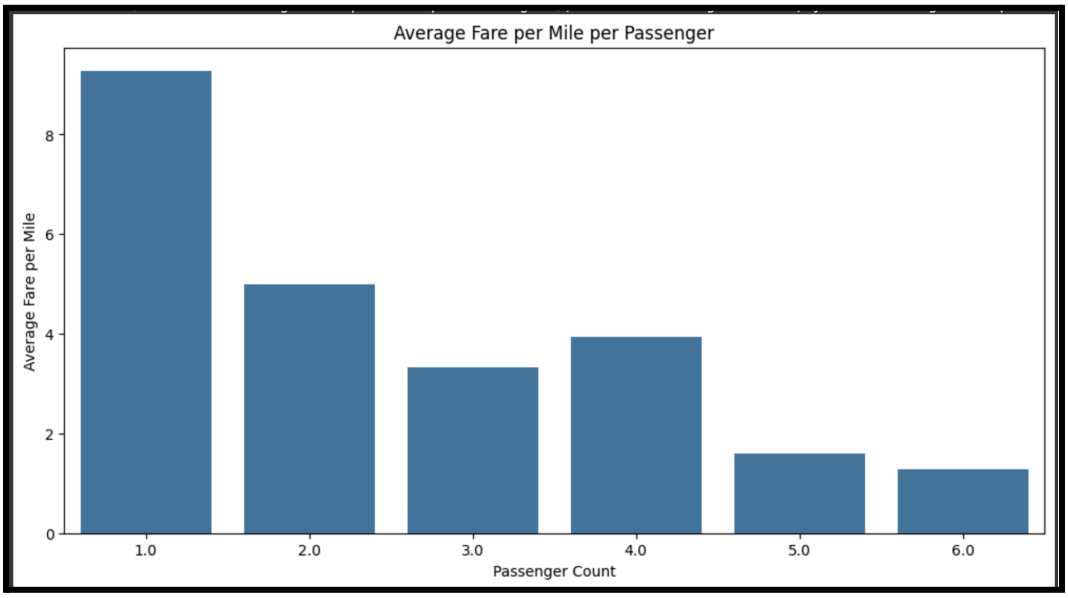
```
# Filter for night hours (11 PM to 5 AM)

# filter for night hours
df_night_hours = df[(df['pickup_hour'] >= 23) | (df['pickup_hour'] <= 5)]

# hourly revenue in night
df_night_hourly_revenue =
df_night_hours.groupby(['pickup_hour'])['total_amount'].sum().reset_index(n
ame='night_hourly_revenue')

# night revenue share ~ 12%
df_night_hourly_revenue['night_hourly_revenue'].sum()*100/df['total_amount'
].sum()



# find the day hour revenue share
```

```
df_day_hours = df[(df['pickup_hour'] > 5) & (df['pickup_hour'] < 23)]

# hourly revenue in day
df_day_hourly_revenue =
df_day_hours.groupby(['pickup_hour'])['total_amount'].sum().reset_index(nam
e='day_hourly_revenue')

# day revenue share ~ 88%
df_day_hourly_revenue['day_hourly_revenue'].sum()*100/df['total_amount'].su
m()
```



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

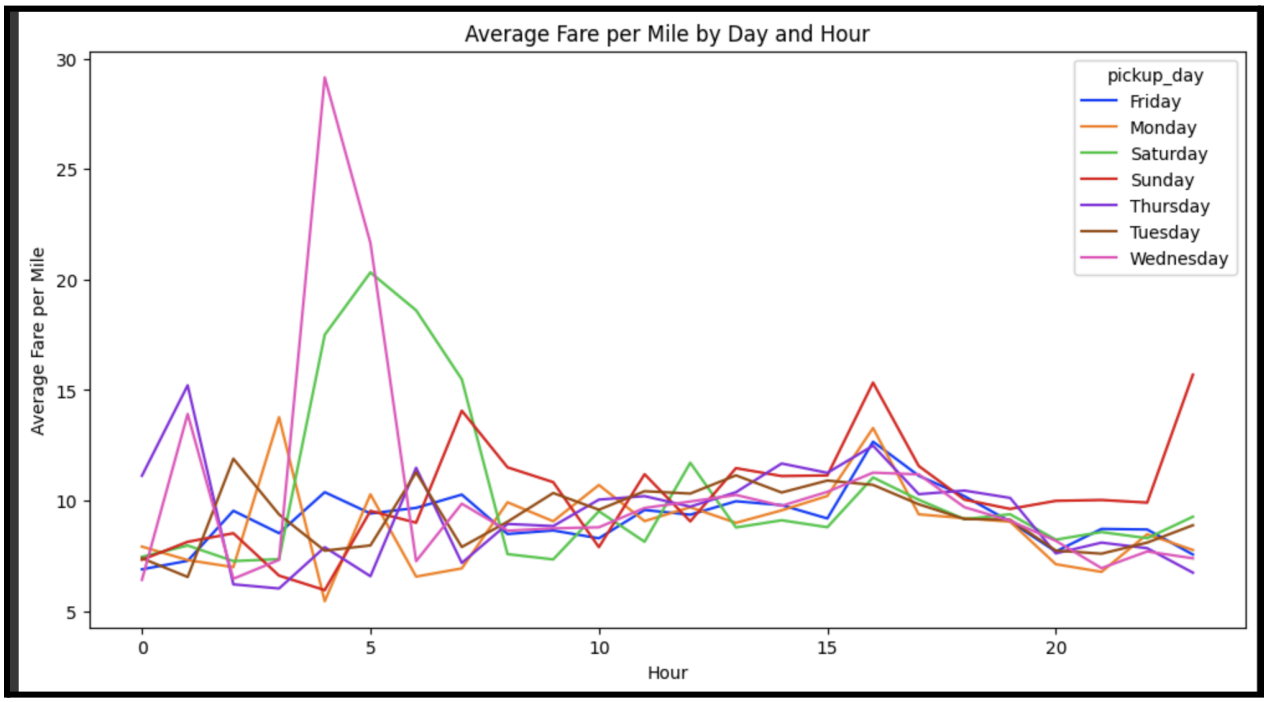
```
# Analyse the fare per mile per passenger for different passenger counts

# filter the records where trip_distance is non zero
df_non_zero_trip_distance = df[df['trip_distance'] != 0]

# get fare_per_mile for different trips
df_non_zero_trip_distance['fare_per_mile'] =
df_non_zero_trip_distance['fare_amount']/df_non_zero_trip_distance['trip_di
stance']

# get average fare per mile for each passenger
df_avg_fare_per_mile_by_passenger_count =
df_non_zero_trip_distance.groupby(['passenger_count'])['fare_per_mile'].mea
n().reset_index(name='average_fare_per_mile')

df_avg_fare_per_mile_by_passenger_count['avg_fare_per_mile_per_passsenger']
=
df_avg_fare_per_mile_by_passenger_count['average_fare_per_mile']/df_avg_far
e_per_mile_by_passenger_count['passenger_count']

df_avg_fare_per_mile_by_passenger_count

# create a graph for average fare per mile for each passenger
plt.figure(figsize=(12, 6))
plt.title("Average Fare per Mile per Passenger")
plt.xlabel("Passenger Count")
plt.ylabel("Average Fare per Mile")
sns.barplot(df_avg_fare_per_mile_by_passenger_count, x='passenger_count',
y='avg_fare_per_mile_per_passsenger')
```



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

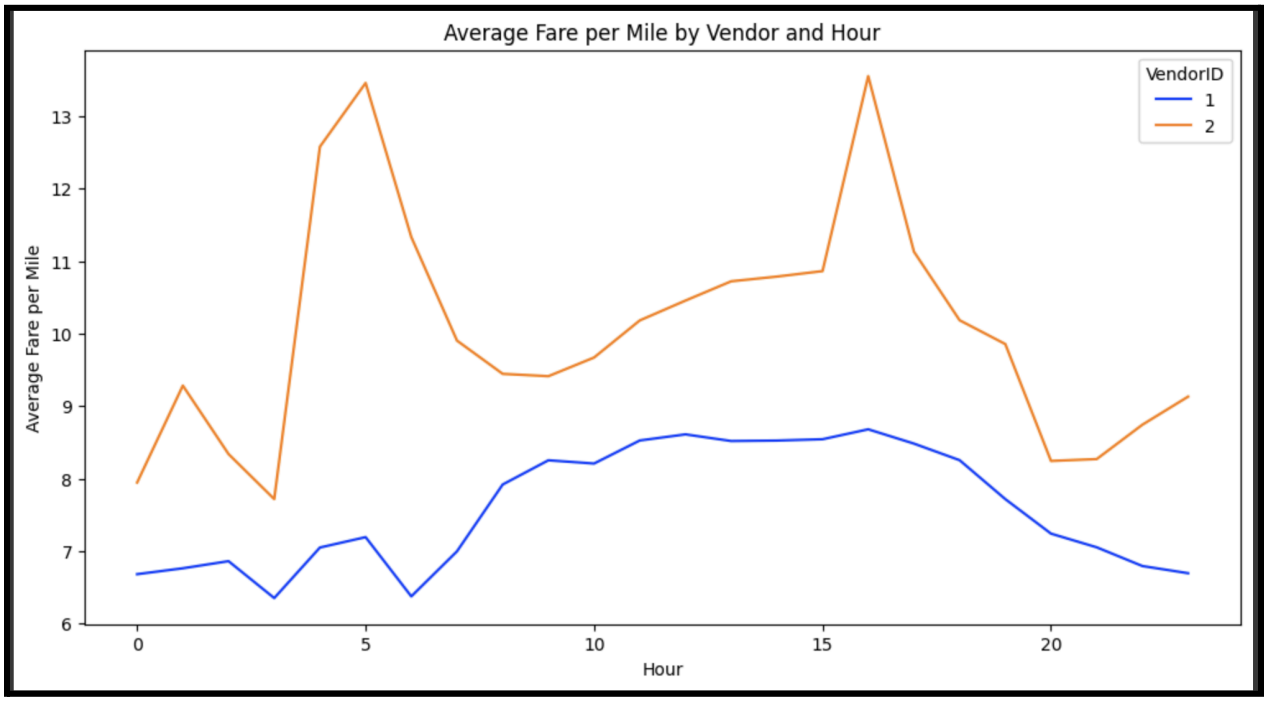
```
# Compare the average fare per mile for different days and for different
times of the day

# filter by non zero trip distance
df_non_zero_trip_distance = df[df['trip_distance'] != 0]

# find average fare per mile for each trip
df_non_zero_trip_distance['fare_per_mile'] =
df_non_zero_trip_distance['fare_amount']/df_non_zero_trip_distance['trip_di
stance']

# find avg fare by day and by time
df_avg_fare_per_mile_by_day_by_hour =
df_non_zero_trip_distance.groupby(['pickup_day',
'pickup_hour'])['fare_per_mile'].mean().reset_index(name='average_fare_per_
mile')


df_avg_fare_per_mile_by_day_by_hour
```

```
# compare the average fare per mile for different days and for different
times of the day

# df_avg_fare_by_day_by_hour.plot(kind='line',
x='pickup_hour',y='average_fare_per_mile', hue='pickup_day',
pallet='bright')

plt.figure(figsize=(12, 6))
plt.title("Average Fare per Mile by Day and Hour")
plt.xlabel("Hour")
plt.ylabel("Average Fare per Mile")

sns.lineplot(data=df_avg_fare_per_mile_by_day_by_hour,
x='pickup_hour',y='average_fare_per_mile', hue='pickup_day',
palette='bright')
```



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

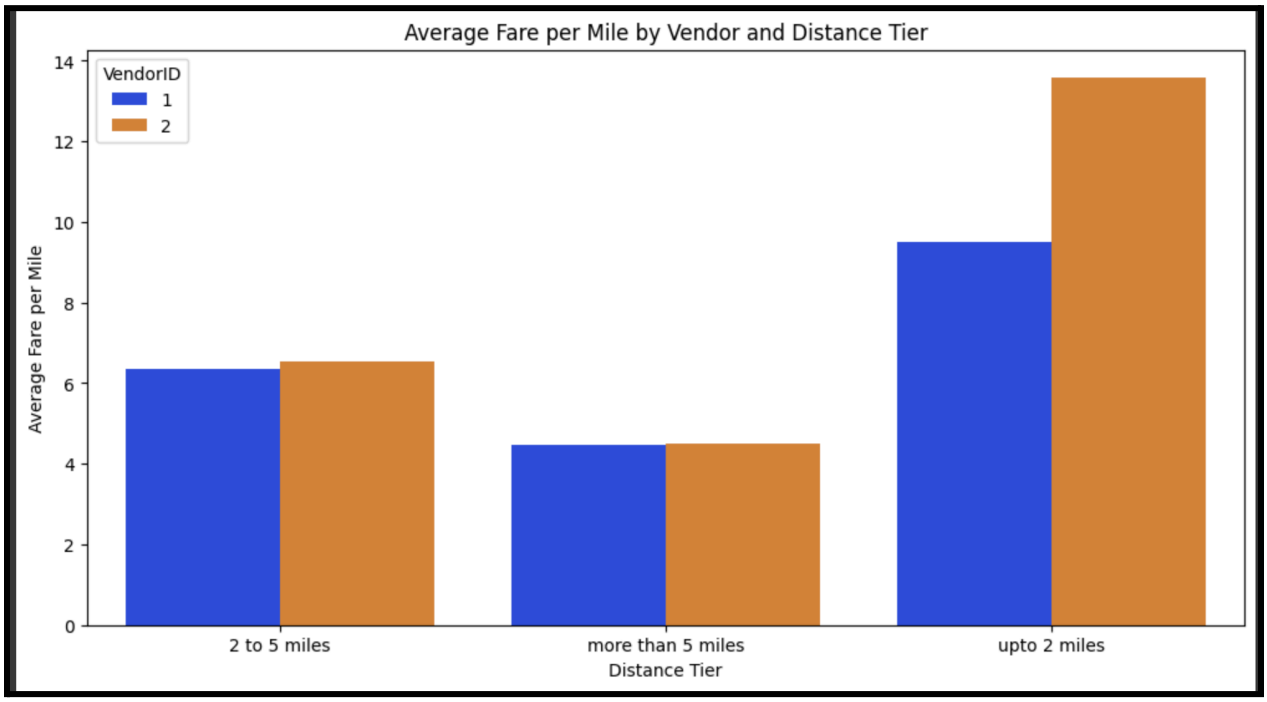
```
# Compare fare per mile for different vendors

# filter by non zero trip distance
df_non_zero_trip_distance = df[df['trip_distance'] != 0]

# find average fare per mile for each trip
df_non_zero_trip_distance['fare_per_mile'] =
df_non_zero_trip_distance['fare_amount']/df_non_zero_trip_distance['trip_di
stance']

# find avg fare by vendor and by hour
df_avg_fare_per_mile_by_vendor_by_hour =
df_non_zero_trip_distance.groupby(['VendorID',
'pickup_hour'])['fare_per_mile'].mean().reset_index(name='average_fare_per_
mile')

df_avg_fare_per_mile_by_vendor_by_hour

# Compare fare per mile for different vendors
plt.figure(figsize=(12, 6))
sns.lineplot(data=df_avg_fare_per_mile_by_vendor_by_hour, x='pickup_hour',
y='average_fare_per_mile', hue='VendorID', palette='bright')
plt.title("Average Fare per Mile by Vendor and Hour")
plt.xlabel("Hour")
plt.ylabel("Average Fare per Mile")
```



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

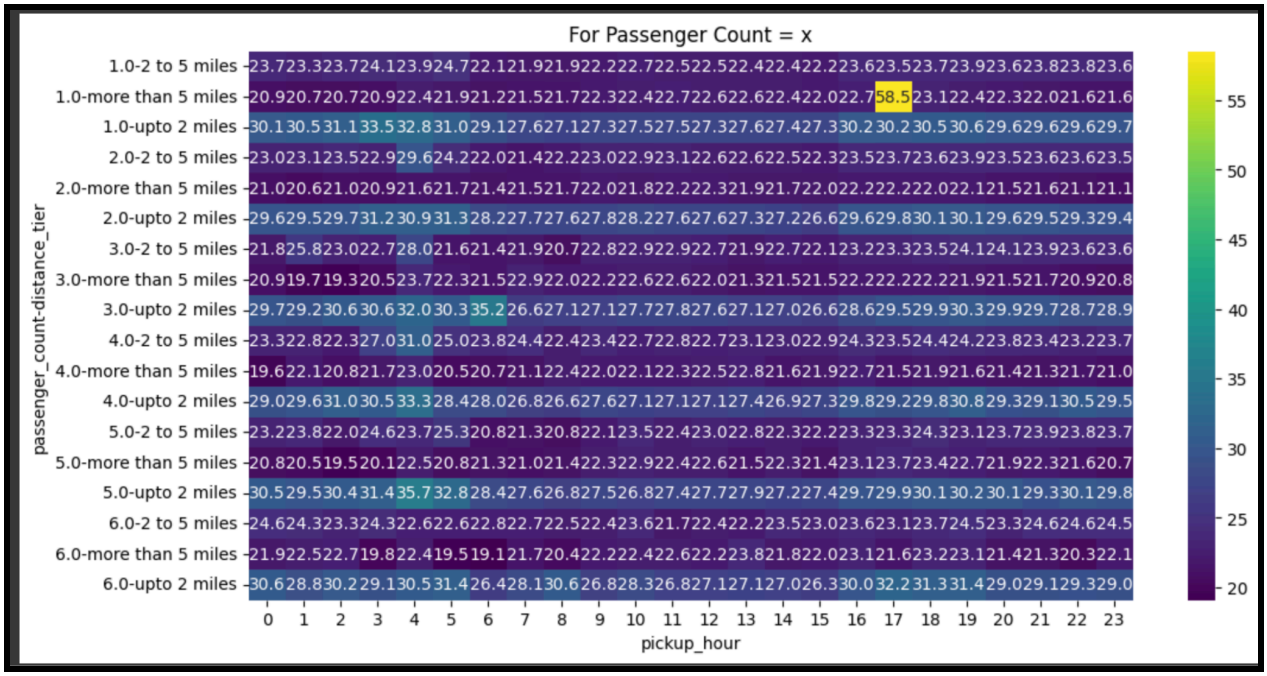
```
# Defining distance tiers

# add a new column for distance tiers. If (trip_distance > 0 &
trip_distance <= 2) then distance_tier = 'upto 2 miles' else if
(trip_distance > 2 & trip_distance <= 5) then distance_tier = '2 to 5
miles' else distance_tier = 'more than 5 miles'
df_non_zero_trip_distance['distance_tier'] =
df_non_zero_trip_distance.apply(lambda x: 'upto 2 miles' if
x['trip_distance'] <= 2 else '2 to 5 miles' if x['trip_distance'] <= 5 else
'more than 5 miles', axis=1)

# find the avg fare per mile by vendor by distance_tier
df_avg_fare_per_mile_by_vendor_by_distance_tier =
df_non_zero_trip_distance.groupby(['VendorID',
'distance_tier'])['fare_per_mile'].mean().reset_index(name='average_fare_pe
r_mile')
df_avg_fare_per_mile_by_vendor_by_distance_tier
```

```
# compare the avg fare per mile by vendor by distance_tier
plt.figure(figsize=(12, 6))
plt.title("Average Fare per Mile by Vendor and Distance Tier")
plt.xlabel("Distance Tier")
plt.ylabel("Average Fare per Mile")
sns.barplot(data=df_avg_fare_per_mile_by_vendor_by_distance_tier,
x='distance_tier', y='average_fare_per_mile', hue='VendorID',
palette='bright')
```



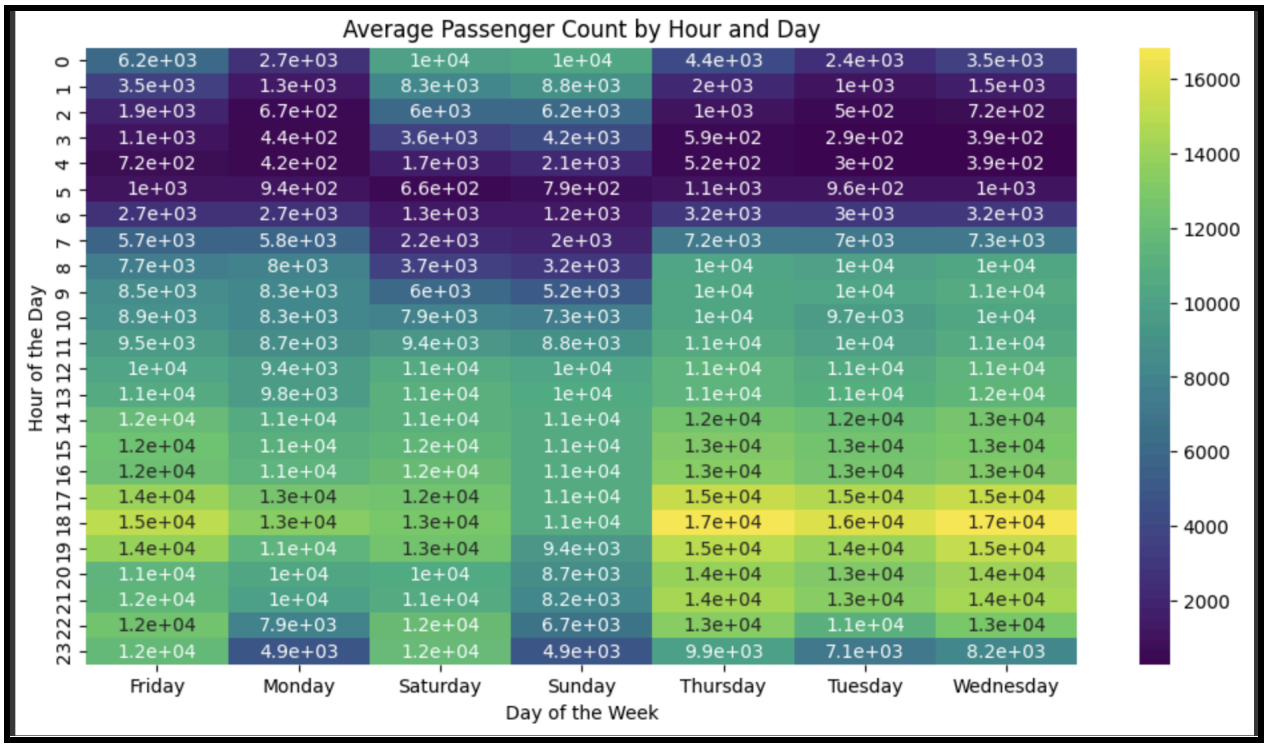
3.2.13. Analyse the tip percentages

```
pass_x = pd.pivot_table(df_non_zero_trip_distance, values='tip_percentage',
index=['passenger_count','distance_tier'], columns='pickup_hour',
aggfunc='mean')
plt.figure(figsize=(12, 6))
sns.heatmap(pass_x, annot=True,cmap='viridis', fmt=".1f")
plt.title('For Passenger Count = x')
plt.show()
```



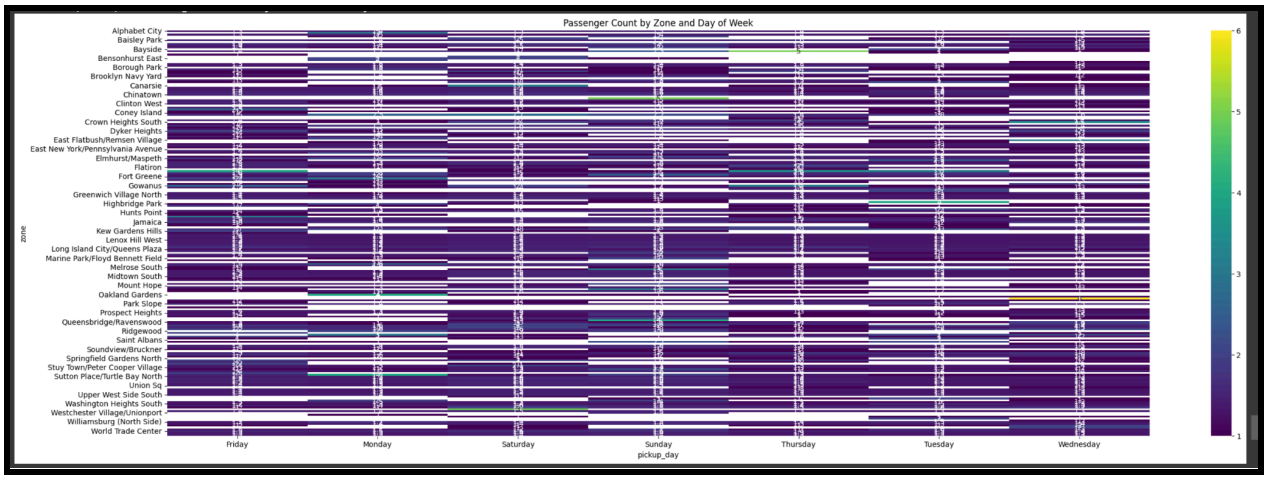
3.2.14. Analyse the trends in passenger count

```
# See how passenger count varies across hours and days

passenger_count_by_hour_day =
df_non_zero_trip_distance.groupby(['pickup_hour',
'pickup_day'])['passenger_count'].size().reset_index(name='avg_passenger_co
unt')
passenger_count_pivot = pd.pivot_table(passenger_count_by_hour_day,
index='pickup_hour', columns='pickup_day', values='avg_passenger_count')
passenger_count_pivot

plt.figure(figsize=(12, 6))  # Adjust figure size as needed
sns.heatmap(passenger_count_pivot, annot=True, cmap='viridis')
plt.title('Average Passenger Count by Hour and Day')
plt.xlabel('Day of the Week')
plt.ylabel('Hour of the Day')
plt.show()
```



3.2.15. Analyse the variation of passenger counts across zones

```
# How does passenger count vary across zones

# merge zones into df_temp
df_merged_with_zones = df_temp.merge(zones, how='inner',
left_on='PULocationID', right_on='LocationID')


zones_px_count_by_day = pd.pivot_table(df_merged_with_zones,
values='passenger_count', index='zone', columns='pickup_day',
aggfunc='mean')
plt.figure(figsize=(30, 10))
sns.heatmap(zones_px_count_by_day, annot=True, cmap='viridis')
plt.title('Passenger Count by Zone and Day of Week')
```


Passenger Count by Zone and Day of Week
zone
Alphabet City
Baisley Park
Bayside
Bensonhurst East
Borough Park
Brooklyn Navy Yard
Canarsie
Chinatown
Clinton West
Coney Island
Crown Heights South
Dyker Heights
East Flatbush/Remsen Village
East New York/Pennsylvania Avenue
Elmhurst/Maspeth
Flatiron
Fort Greene
Gowanus
Greenwich Village North
Highbridge Park
Hunts Point
Jamaica
Kew Gardens Hills
Lenox Hill West
Long Island City/Queens Plaza
Marine Park/Floyd Bennett Field
Melrose South
Midtown South
Mount Hope
Oakland Gardens
Park Slope
Prospect Heights
Queensbridge/Ravenswood
Ridgewood
Saint Albans
Soundview/Bruckner
Springfield Gardens North
Stuy Town/Peter Cooper Village
Sutton Place/Turtle Bay North
Union Sq
Upper West Side South
Washington Heights South
Westchester Village/Unionport
Williamsburg (North Side)
World Trade Center
Friday
Monday
Saturday
Sunday
Thursday
Tuesday
Wednesday
pickup_day
6
5
4
3
2
1

	zone	pickup_hour	passenger_count
1707	Midtown Center	18	5986
1706	Midtown Center	17	5698
2585	Upper East Side South	18	5313
2584	Upper East Side South	17	5311
1708	Midtown Center	19	5253
2558	Upper East Side North	15	5170
2581	Upper East Side South	14	5117
2582	Upper East Side South	15	5080
1705	Midtown Center	16	5050
2583	Upper East Side South	16	4924
2560	Upper East Side North	17	4814
2561	Upper East Side North	18	4771
1246	JFK Airport	22	4714
2579	Upper East Side South	12	4620
2557	Upper East Side North	14	4605
2580	Upper East Side South	13	4593
1704	Midtown Center	15	4586
2559	Upper East Side North	16	4583
1731	Midtown East	18	4578
1709	Midtown Center	20	4573

```
[ ]  df_zone_by_pax_count_by_hour[df_zone_by_pax_count_by_hour['zone'].str.cont
```

	zone	pickup_hour	passenger_count
1246	JFK Airport	22	4714
1245	JFK Airport	21	4502
1240	JFK Airport	16	4469
1239	JFK Airport	15	4385
1244	JFK Airport	20	4247
1247	JFK Airport	23	4184
1243	JFK Airport	19	4132
1241	JFK Airport	17	4065
1238	JFK Airport	14	3895
1370	LaGuardia Airport	14	3807
1242	JFK Airport	18	3600
1372	LaGuardia Airport	16	3551
1371	LaGuardia Airport	15	3396

```
# 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.
df_zones_with_trips = df_merged_with_zones.groupby(['zone'])['passenger_count'].mean().reset_index(name='avg_passenger_count')
# df_zones_with_trips.sort_values(by='avg_passenger_count', ascending=False)
df_zones_with_trips.sort_values(by='avg_passenger_count', ascending=False).head(30)
```

	zone	avg_passenger_count
147	Ocean Parkway South	6.000000
208	West Farms/Bronx River	5.000000
37	Claremont/Bathgate	5.000000
1	Arrochar/Fort Wadsworth	3.000000
81	Fresh Meadows	2.666667
145	Oakland Gardens	2.200000
77	Flushing	2.111111
88	Green-Wood Cemetery	2.000000
71	Erasmus	2.000000
213	Willets Point	2.000000
188	Sunset Park East	2.000000
142	Murray Hill-Queens	2.000000
46	Corona	2.000000
14	Belmont	2.000000
16	Bensonhurst West	2.000000
159	Queensboro Hill	2.000000

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

```
# How often is each surcharge applied?
df_merged_with_zones_with_extra_charges =
df_merged_with_zones[(df_merged_with_zones['improvement_surcharge'] > 0) |
```

```
(df_merged_with_zones['congestion_surcharge'] > 0)]
# df_merged_with_zones.info()


df_merged_with_zones_with_extra_charges['extra_charges'] =
df_merged_with_zones_with_extra_charges['improvement_surcharge'] +
df_merged_with_zones_with_extra_charges['congestion_surcharge']


# extra charges applied by hour
df_extra_charges_by_hour =
df_merged_with_zones_with_extra_charges.groupby(['extra_charges','pickup_ho
ur']).size().reset_index(name='count_of_extra_charge_applied')
df_extra_charges_by_hour.sort_values(by='count_of_extra_charge_applied',
ascending=False)


# extra charges applied by zone
df_extra_charges_by_zone =
df_merged_with_zones_with_extra_charges.groupby(['extra_charges','zone']).s
ize().reset_index(name='count_of_extra_charge_applied')
df_extra_charges_by_zone.sort_values(by='count_of_extra_charge_applied',
ascending=False).head(50)


# extra charges applied by zone by hour
df_extra_charges_by_zone_by_hour =
df_merged_with_zones_with_extra_charges.groupby(['extra_charges','zone',
'pickup_hour']).size().reset_index(name='count_of_extra_charge_applied')
df_extra_charges_by_zone_by_hour.sort_values(by='count_of_extra_charge_appl
ied', ascending=False).head(50)
```

	extra_charges	pickup_hour	count_of_extra_charge_applied
85	3.5	18	96350
84	3.5	17	90262
86	3.5	19	86473
83	3.5	16	80166
82	3.5	15	79040
...	...	...	...
39	2.5	8	1
42	2.5	14	1
40	2.5	9	1
41	2.5	12	1
38	2.5	1	1

91 rows × 3 columns

	extra_charges	zone	count_of_extra_charge_applied
436	3.5	Upper East Side South	68579
382	3.5	Midtown Center	66204
435	3.5	Upper East Side North	62302
383	3.5	Midtown East	51599
370	3.5	Lincoln Square East	47865
396	3.5	Penn Station/Madison Sq West	47518
429	3.5	Times Sq/Theatre District	43109
390	3.5	Murray Hill	42288
384	3.5	Midtown North	40813
438	3.5	Upper West Side South	40186
433	3.5	Union Sq	39621
320	3.5	East Chelsea	36577
358	3.5	JFK Airport	36143
305	3.5	Clinton East	35616
367	3.5	LaGuardia Airport	35375
369	3.5	Lenox Hill West	34198
328	3.5	East Village	34164
445	3.5	West Village	33299

	extra_charges	zone	pickup_hour	count_of_extra_charge_applied
4186	3.5	Midtown Center	18	5965
4185	3.5	Midtown Center	17	5667
4857	3.5	Upper East Side South	18	5306
4856	3.5	Upper East Side South	17	5301
4187	3.5	Midtown Center	19	5236
4830	3.5	Upper East Side North	15	5152
4853	3.5	Upper East Side South	14	5112
4854	3.5	Upper East Side South	15	5070
4184	3.5	Midtown Center	16	5020
4855	3.5	Upper East Side South	16	4915
4832	3.5	Upper East Side North	17	4795
4833	3.5	Upper East Side North	18	4754
4851	3.5	Upper East Side South	12	4610
4829	3.5	Upper East Side North	14	4583
4852	3.5	Upper East Side South	13	4580
4831	3.5	Upper East Side North	16	4569
4210	3.5	Midtown East	18	4566
4188	3.5	Midtown Center	20	4561
4183	3.5	Midtown Center	15	4550

4. Conclusions

4.1. Final Insights and Recommendations

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

Recommendations:

1. [Midtown centre, Upper East Side South, Upper East Side North] seems to have most of the offices. So Deploy more cabs from 12 PM to 6 PM specially on Wednesday and Thursday. But not required on weekends.

2. Deploy more cab at JFK Airport from 2 PM to 8 PM and LaGuardia Airport from 11 AM to 4 PM. Other airports have very less rush so only 1-2% cabs can be deployed there.

3. Avg speed from East Elmhurst to LaGuardia Airport and Upper East Side South to Upper West Side North is low compared to other routes.

Reasoning for #1,2

a. Midtown centre has max rush around 5 PM and 6 PM on Wednesday and Thursday. This rush keeps on increasing from 2 PM to 7 PM. This rush is least on Monday and Friday. Medium on Tuesday but most on Wednesday and Thursday.

b. Upper East Side South has max rush on 5 PM and 6 PM on Wednesday and Thursday. This rush keeps on increasing from 12 PM to 6 PM. This rush is least on Saturday. Also less on Monday and Friday.

c. Upper East Side North most rush on 5 PM and 6 PM on Wednesday and Thursday. This rush keeps on increasing from 12 PM to 6 PM. This rush is least on Saturday. Also less on Monday and Friday.

d. Most of the extra charges (i.e congestion, improvement surcharge) are applied on [Midtown centre, Upper East Side South, Upper East Side North] around 5 PM to 6 PM.

e. JFK Airport has the most rush on weekends compared to LaGuardia Airport. EastElumhust is also having max rush on day as well night.

f. Upper East Side South and Upper West Side North seem to have most offices. That's why speed seems to have less speed in the evening. Similarly East Elmhurst to LaGuardia Airport avg speed is low compared to others. Becoz of the airport rush.

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.

Recommendations

1. Place maximum cabs on Wednesday and Thursdays over [Midtown centre, Upper East Side South, Upper East Side North] as it has most of the offices. From 3-6 PM there is more rush. Rush starts going down from 6 PM onwards. Over the weekend there is no rush.

2. Place more can in JFK Airport from 2 PM to 8 PM and LaGuardia from 11 AM to 4 PM. JFK Airport has more rush on weekends than LaGuardia.

3. Increase more cab over weekend for morning trips as there is more demand.

4. Coney Island, Chinatown has avg. pax count that varies from 4 to 6 over the weekend, deploy more SUVs over these zones.

Reasoning for above points:

1. Wednesdays & Thursdays seem to have most trips around 5 PM to 6 PM. There is a trend from 3 PM to 6 PM rush increase then it keeps on decreasing by 9 PM. Mostly becoz of office hours.This trend is clearly visible across [Midtown centre, Upper East Side South, Upper East Side North].

2. On Weekends, there are more trips from 12 AM to 5 AM compared to weekdays. Mostly because people start their trips. Avg fare per mile is also more on weekends from 12 AM to 8 AM.

3. JFK Airport has much more traffic than LaGuardia over the weekend. For weekdays on an avg. JFK Airport has more traffic than LaGuardia.

4. Coney Island, Chinatown get most of the rides with 4-6 pax over Saturday, Sunday, Monday. These places seem to have tourist places.

5. Oct month has the most trips and revenue is the most from the 4th quarter.

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

Recommendations:

1. Vendor 2 has high prices as compared to Vendor 1 but still the number of trips are triple as compared to Vendor 1. Vendor 2 need not decrease its prices but Vendor 1 can increase their prices.

2. Increase the fare amount over the weekend for morning trips. Increase prices for JFK airport.

3. Most of the passengers travelling on weekdays and their offices are in [Midtown centre, Upper East Side South, Upper East Side North]. Most rides are having 1 or 2 pax. Tip% are also high. Fare amount can be increased over these areas in evening trips from 12 PM - 6 PM. Most of the congestion charges and improvements are applied themselves. If possible improvements charges can be decreased.

4. Vendor 1 can increase avg fare per mile for morning trips over weekend and evening trips over weekday but vendor 2 has to not increase it's avg fare per mile. But vendorID 6, needs to decrease it's pricing drastically for less than 2 miles since it's very high and it also has less number of trips

Reasoning for above points:

1. Vendor 2 has more avg, fare per mile than vendor 1 mostly for short trips (upto 2 miles) charges are much more than vendor 1.

2. Vendor 2 has increased prices over morning trips around 5 AM - 7 AMa nd Evening trips from 5 PM - 6 PM.

3. pax = 1, on an avg gives more tip% for long rides in the evening around 4-6 PM.

5. pax = 2 to 6, on an avg gives more tip% for short distance rides either in early morning 2 AM to 6 AM or in evening around 4 PM to 7 PM.