Report: Optimizing NYC Taxi Operations

Submitted By: Ashwini KS

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

Data set is loaded using the panda dataframe module. The dataset given are in parquet format and the pd.read_parquet () to read the given file.

There are 12 files of data, corresponding to 12 months of the year. Each data file contains large amount of data, and is in need of sampling.

Sampling technique used is Systematic Sampling. The logic behind this is to use 5% sample from each hour in a day for all the months. This would make a large data set as well, but this is a workable sample.

As a test, and to be more efficient in using the space/memory, I have first done the sampling just to first month, then extrapolated this idea to other months, merging the sample from each month to a single DataFrame.

1) Load the January data.
2) Check the columns => Note that there are date columns
3) Split to Date, hour => group by the above and get 5% data from each hour
4) If the data above looks good, repeat the same in a loop to all the other files.
5) Check the above df using different functions like head(), info() etc.
6) Download the file for use.

2. Data Cleaning

2.1. Fixing Columns

2.1.1. Fix the index

```
# Fix the index and drop any columns that are not needed
df.reset_index(drop=True, inplace=True)
df.columns
df.drop(columns=['pick_date', 'pick_hr', 'store_and_fwd_flag'], inplace=True)
```

We fix the index using reset_index() and drop a few columns that may not be necessary

2.1.2. Combine the two airport_fee columns

```
# Combine the two airport fee columns
df.loc[:, ['airport_fee', 'Airport_fee']]
df.isnull().sum()
df['Airport_Fee'] = df['Airport_fee'].fillna(0) + df['airport_fee'].fillna(0) #combine the values in column
df.drop(columns=['Airport_fee', 'airport_fee'], inplace=True)
```

Combined the airport_fee columns

2.2. Handling Missing Values

2.2.1. Find the proportion of missing values in each column

```
# Find the proportion of missing values in each column
#first do a central tendency analysis => Univariate to obtain missing values
print(df.isnull().sum())

#to get the proportion, missing values in each column => percentage of nulls
print(((df.isnull().sum()/df.shape[0])*100).round(2).astype(str) + '%')
```

```
VendorID                     0
tpep_pickup_datetime         0
tpep_dropoff_datetime        0
passenger_count          64874
trip_distance                0
RatecodeID               64874
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     64874
Airport_Fee                  0
dtype: int64
VendorID                  0.0%
tpep_pickup_datetime      0.0%
tpep_dropoff_datetime     0.0%
passenger_count          3.42%
trip_distance             0.0%
RatecodeID               3.42%
PULocationID              0.0%
DOLocationID              0.0%
payment_type              0.0%
fare_amount               0.0%
extra                     0.0%
mta_tax                   0.0%
tip_amount                0.0%
tolls_amount              0.0%
improvement_surcharge     0.0%
total_amount              0.0%
congestion_surcharge     3.42%
Airport_Fee               0.0%
dtype: object
```

Found the proportions of missing values. Maximum proportion is 3.42&

2.2.2. Handling missing values in passenger_count

Passenger_count has many missing values. Since this is a numerical, integer values, we would need to impute this with the mode, which would not change the data overall.

Also, 0 passengers do not make sense for trips, hence replace this with mode as well.

2.2.3. Handle missing values in RatecodeID

RatecodeID is also a numerical, integer values, we would need to impute this with the mode, which would not change the data overall.

2.2.4. Impute NaN in congestion_surcharge

On checking the describe of congestion_surcharge, we clearly see that this would need a median be imputed as this is a float like data. Hence we impute NaN with such values.

Overall, finally when checked, the missing values are handled.

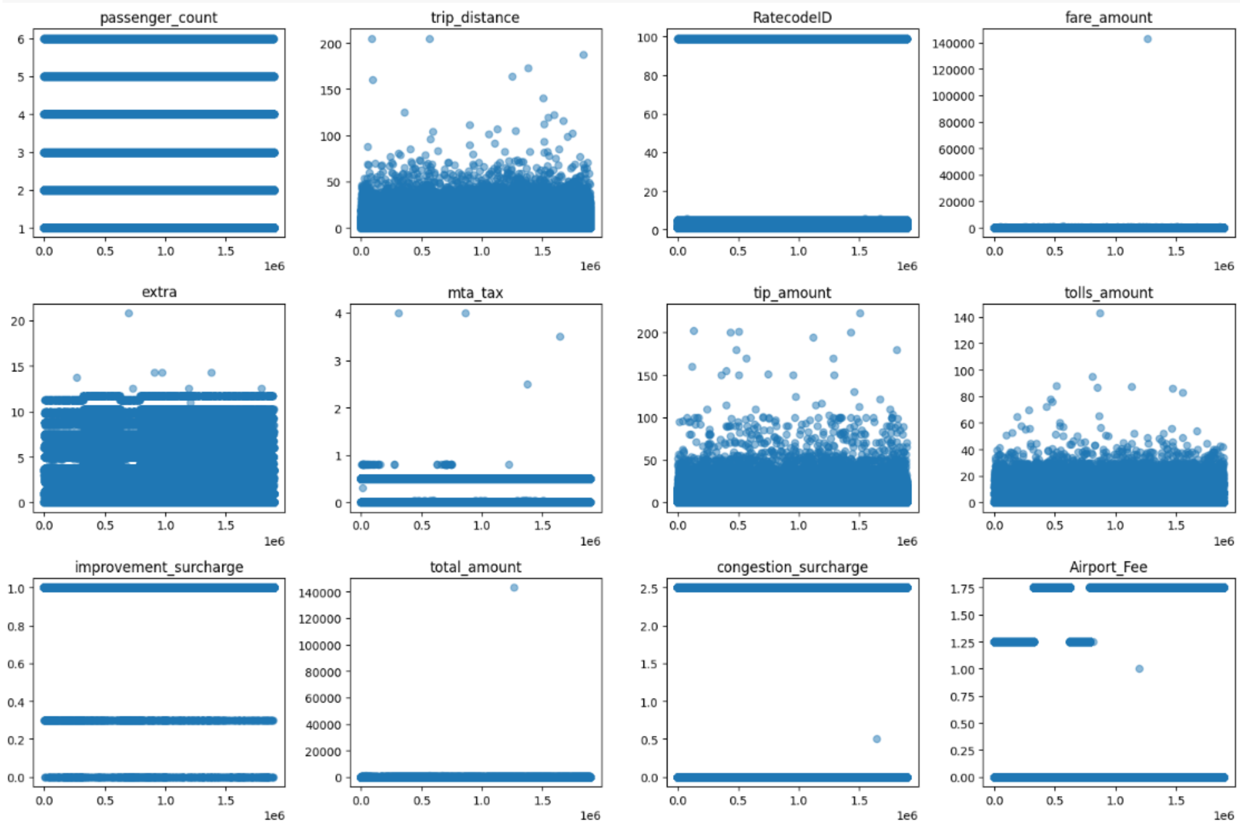
```
# Handle any remaining missing values
df.isnull().sum()
#no other missing values detected
```

	0
VendorID	0
tpep_pickup_datetime	0
tpep_dropoff_datetime	0
passenger_count	0
trip_distance	0
RatecodeID	0
PULocationID	0
DOLocationID	0
payment_type	0
fare_amount	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 and Standardising Values

To check outliers, we have univariate as may use Whisker/Box plot or scatter plot as well. In scatter plot we can see an overall scatter of the points. They may be an actual outlier or it could be a value that makes sense.



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

1) Payment Type : This is a categorical value and won't have an outlier defined.
2) Trip distance : There are outliers in this column. As in scatter plot, there really high trip distances.
3) Tip_amount: As in above scatter plot, you can see outliers.

Same can be seen by comparing the mean and medians of the above 2 columns.

In order to handle outliers, in passenger count, we remove the count > 7.

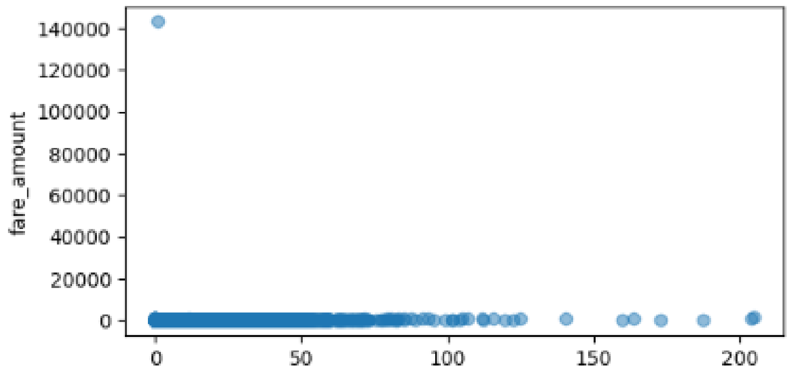
As per univariate analysis, we can also see the trip distance >= 250 is not necessary. We also see that payment type has NaN values, which is then replaced with the mode of that column.

from scatter plot, we have congestion charges and airport fee to also have a peculiar value. Let us check the above using boxplot. On checking for the outlier value, it looks legit and hence we keep it.

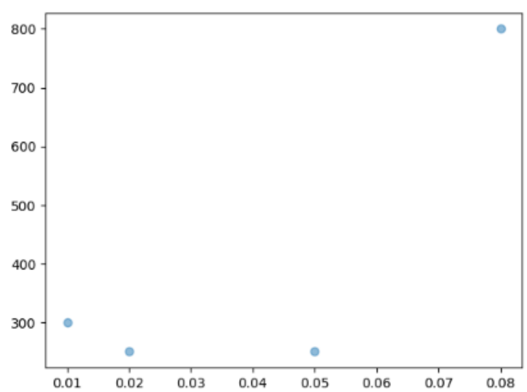
From Bivariate analysis:

```
[ ] #BIVARIATE ANALYSIS
    plt.figure(figsize = (6, 3))
    plt.scatter(df['trip_distance'], df['fare_amount'], alpha=0.5)
    plt.xlabel('Trip Distance')
    plt.ylabel('fare_amount')
    #clearly note that fare amount is ver high for a small distance => outlier
```

```
Text(0, 0.5, 'fare_amount')
```




```
df = df[df['fare_amount'] < 100000]  # now re-run the previous code, you will see the scatter plot change
#again a zero distance should not have non-zero fare
df = df[(~(df['trip_distance'] == 0) & (df['fare_amount'] > 0))] # now re-run the previous code, you will see the scatter plot change

#also as mentioned, if there are trip_distances nearly zero ==> let us start analysing for less than .5 miles for now
df_filtered = df[(df['trip_distance'] < 0.1) & (df['fare_amount'] > 200)]
plt.scatter(df_filtered['trip_distance'], df_filtered['fare_amount'], alpha=0.5)
plt.show()

#from above, we see that there are cases with less than a 0.1 mile distance and >200 USD fare_amount => which is unrealistic
#since we have high amount of data, we can ignore such columns
df = df[~((df['trip_distance'] < 0.1) & (df['fare_amount'] > 200))]
```




```
[ ] #now let us check for all those with trip distance 0 and fare amount 0
    df[((df['trip_distance'] == 0) & (df['fare_amount'] == 0))]

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

VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amc

We now save this df as a cleaned file.

3. Exploratory Data Analysis

3.1. General EDA: Finding Patterns and Trends

3.1.1. Classify variables into categorical and numerical

```
df.columns.tolist()
```

```
['VendorID',
 'tpep_pickup_datetime',
 'tpep_dropoff_datetime',
 'passenger_count',
 'trip_distance',
 'RatecodeID',
 'PULocationID',
 'DOLocationID',
 'payment_type',
 'fare_amount',
 'extra',
 'mta_tax',
 'tip_amount',
 'tolls_amount',
 'improvement_surcharge',
 'total_amount',
 'congestion_surcharge',
 'Airport_Fee',
 'trip_duration']
```

In the above list, we have quantity, count and price columns which are numerical. We now use this idea to bifurcate numerical columns and non-numerical columns.

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

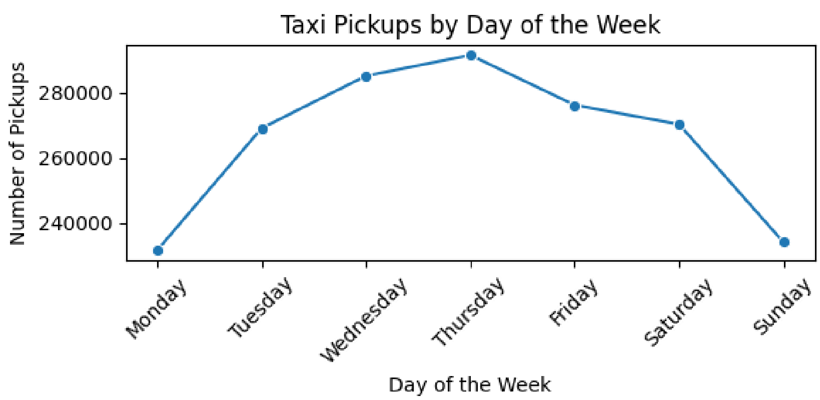
1) Taxi Pickups by hours:

```
# Find and show the daily trends in taxi pickups (days of the week)
df['pickup_hr'] = df['tpep_pickup_datetime'].dt.hour

hr_counts = df['pickup_hr'].value_counts().reset_index()
hr_counts.columns = ['pickup_hr', 'count']
#hr_counts.columns

plt.figure(figsize = (6,3))
sns.lineplot(data=hr_counts, x='pickup_hr', y='count', marker='o')

plt.title('Taxi Pickups by HR')
plt.xlabel('Hour Number')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
#clearly, the highest pickups between 15 to 20 hrs, looks like the peak is at 18 ish i.e. ~6 pm
```

The peak is at 6pm i.e. 18 hours.

2) Taxi pickups by weeks:

```
# Find and show the daily trends in taxi pickups (days of the week)
df['pickup_week'] = df['tpep_pickup_datetime'].dt.dayofweek
day_name_map = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday',
                3: 'Thursday', 4: 'Friday', 5: 'Saturday', 6: 'Sunday'}
df['pickup_week'] = df['pickup_week'].map(day_name_map)

day_counts = df['pickup_week'].value_counts().reindex(
    ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
)
day_counts = pd.DataFrame(data = day_counts.reset_index())
#day_counts.columns

plt.figure(figsize = (6,3))
sns.lineplot(data=day_counts, x='pickup_week', y='count', marker='o')

plt.title('Taxi Pickups by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
#clearly, the highest pickups are on Thursday, Wednesday is the 2nd highest. Least is on Monday.
```

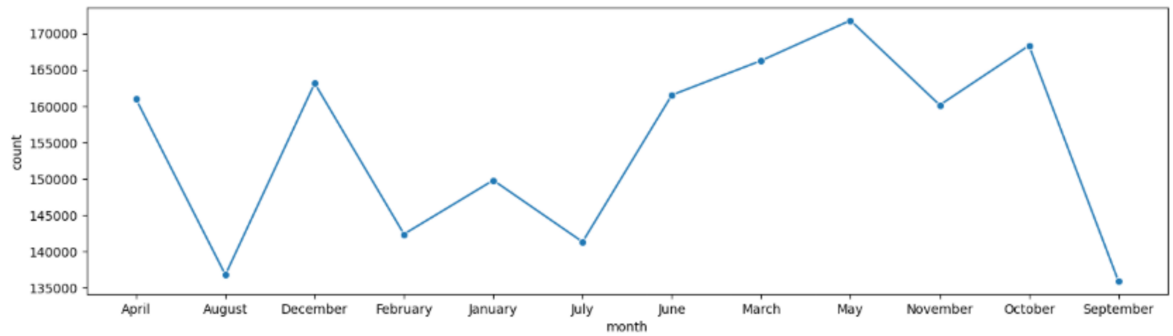


Pickups are at peak on Thursdays and lowest on Mondays.

3) Taxi pickups by Month and quarter:

```
# Show the monthly trends in pickups
df_month = df['tpep_pickup_datetime'].dt.month_name().value_counts().sort_index()
df_month = pd.DataFrame(data = df_month.reset_index())
df_month.columns = ['month', 'count']
#print(df_month)

#plot of the above
plt.figure(figsize = (15,4))
sns.lineplot(data=df_month, x='month', y='count', marker='o')

#clearly May seems to have the most pick ups
```

<Axes: xlabel='month', ylabel='count'>



May month has highest pickups and September has the lowest.

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

2.1.3 [5 marks]

Fix columns with negative (monetary) values

```
[16] # check where values of fare amount are negative
     #first get all the columns that have negative values

     df.describe() #shows some negative values in the metrics

     num_cols = df.select_dtypes(include=['float', 'float64', 'float32']).columns.tolist()
     neg_cols = []
     for cols in num_cols:
       if df[df[cols] < 0].shape[0] > 0:
         neg_cols.append(cols)
     print(neg_cols)

     #then
```

```
['extra', 'mta_tax', 'improvement_surcharge', 'total_amount', 'congestion_surcharge', 'Airport_Fee']
```

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

```
[ ] # Analyse RatecodeID for the negative fare amounts
    df_neg = pd.DataFrame()
    for col in neg_cols:
      df_neg[col] = df[col]

    #Add the ratecodeID as well, to analyse
    df_neg['RatecodeID'] = df['RatecodeID']

    #sns.pairplot(df_neg)
```

```
[ ] # Find which columns have negative values
    #check the above code to get all negative columns
```

```
[ ] # fix these negative values
    for cols in neg_cols:
      df[cols] = np.abs(df[cols])
```

Negative columns fixed. To identify them we have used describe() function.

```
# Analyse the above parameters
df[df['fare_amount'] == 0] #we have removed the rows with fare_amount as 0
df[df['tip_amount'] == 0] #this contains 0 values, but this could also give us an idea on how many passengers tip the taxi drivers
df[df['total_amount'] == 0] #no 0 values => this is correct
df[df['trip_distance'] == 0]  #we have removed the rows with trip_distance as 0
```

VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amount	...	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount	co

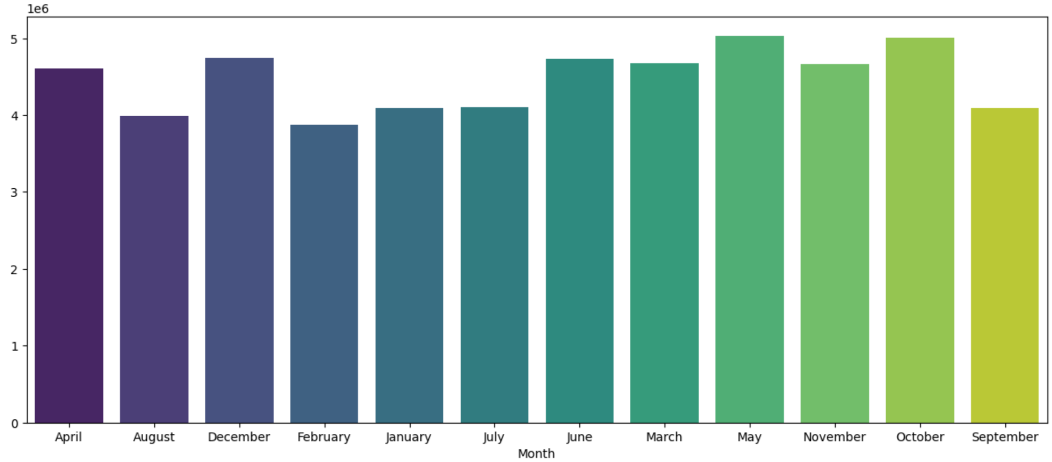
0 rows × 21 columns

Zero values filtered out

3.1.4. Analyse the monthly revenue trends

```python
# Group data by month and analyse monthly revenue
df['Month'] = df['tpep_pickup_datetime'].dt.month_name()
monthly_revenue = df.groupby('Month')['total_amount'].sum()
#monthly_revenue.sort_values()
#print(df_month)

plt.figure(figsize=(15, 6))
sns.barplot(x=monthly_revenue.index, y=monthly_revenue.values, palette="viridis")
#max in may, least in Febraury => Note how the pickups are actually least in September but the revenue is least in August
```

```
/tmp/ipython-input-3908273391.py:8: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=monthly_revenue.index, y=monthly_revenue.values, palette="viridis")
<Axes: xlabel='Month'>
```

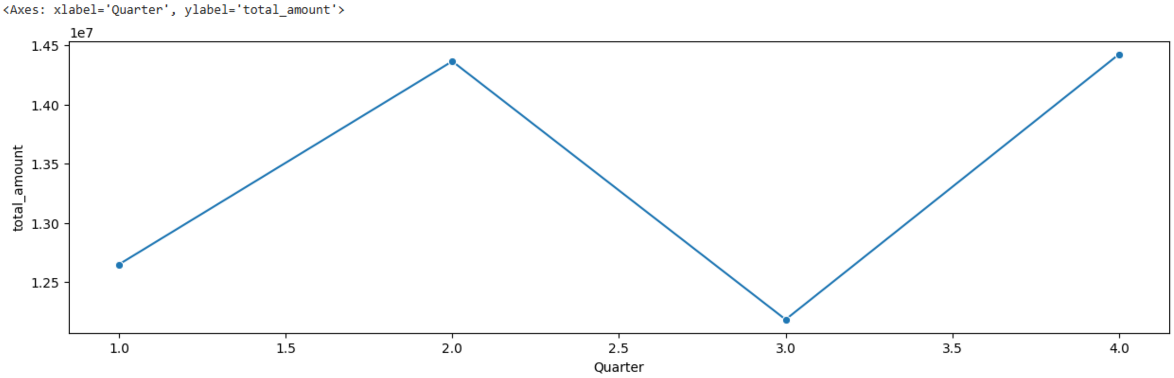


Highest in May, and least in February.

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

```python
# Calculate proportion of each quarter
df['Quarter'] = df['tpep_pickup_datetime'].dt.quarter
quarter_revenue = df.groupby('Quarter')['total_amount'].sum()
#quarter_revenue.sort_values()

quarter_revenue = pd.DataFrame(data = quarter_revenue.reset_index())
quarter_revenue.columns = ['Quarter', 'total_amount']
#print(quarter_revenue)
plt.figure(figsize = (15,4))
sns.lineplot(data=quarter_revenue, x='Quarter', y='total_amount', marker='o')
#the least is in 3rd quarter. Also see how the year 2023, has a zig zag/up and down of quarterly revenue
```

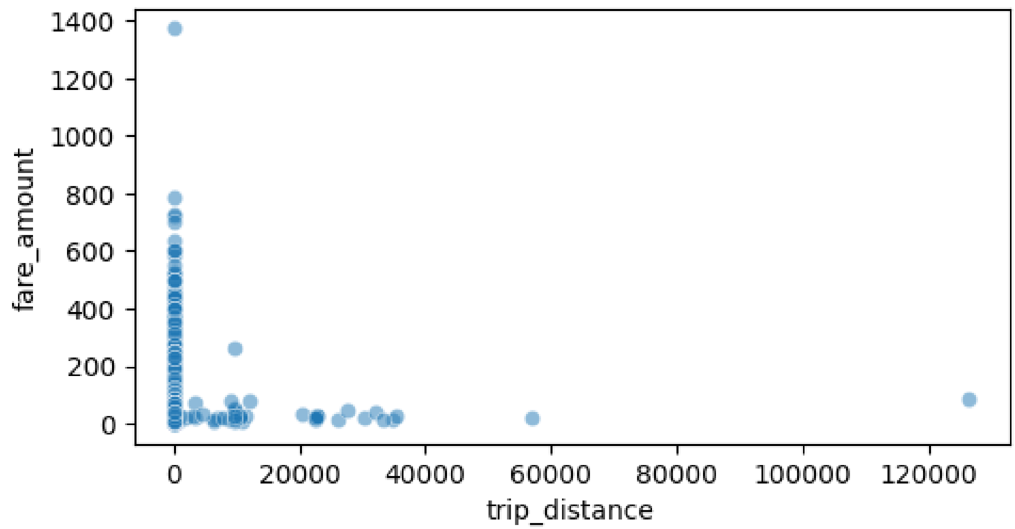
```
<Axes: xlabel='Quarter', ylabel='total_amount'>
```



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

```python
plt.figure(figsize=(6,3))
sns.scatterplot(data=df, x='trip_distance', y='fare_amount', alpha=0.5)
```

```
<Axes: xlabel='trip_distance', ylabel='fare_amount'>
```




```python
# Show how trip fare is affected by distance
df["trip_distance"].corr(df["fare_amount"]) #since correlation coeff is 0.036, it clearly shows that these features are not very well correlated (3.6%)
#Trip fare is directly proportional to trip_distance
```

```
np.float64(0.0361817539537539)
```

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

```python
# Show relationship between fare and trip duration
pick = df['tpep_pickup_datetime']
drop = df['tpep_dropoff_datetime']
df['trip_duration'] = (drop - pick).dt.total_seconds()/60
df["trip_duration"].corr(df["fare_amount"])

#plt.figure(figsize = (15,4))
#sns.lineplot(data=df, x='trip_duration', y='fare_amount', marker='o')
```

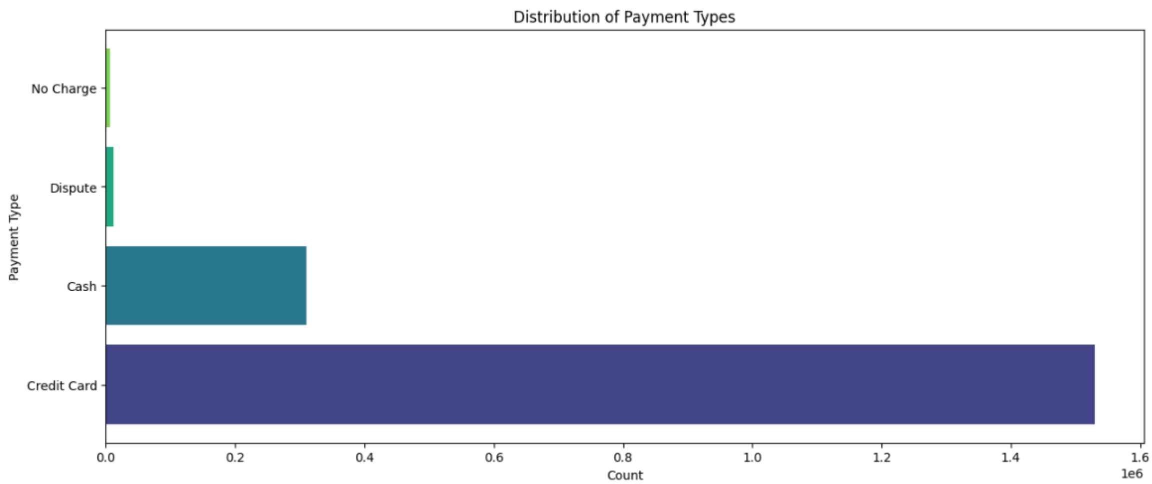
```
np.float64(0.27933033266772384)
```

```python
# Show relationship between tip and trip distance
df['tip_amount'].corr(df["trip_distance"])
```

```
np.float64(0.5864234289103515)
```

3.1.8. Analyse the distribution of different payment types

```python
# Analyse the distribution of different payment types (payment_type).
df_payments = df['payment_type'].value_counts().reset_index() #this would give the counts of all the types of payment
#df_payments.columns
payment_map = {1: 'Credit Card', 2: 'Cash', 3: 'No Charge', 4: 'Dispute'}
df_payments['payment_type'] = df_payments['payment_type'].map(payment_map)
df_payments.columns = ['payment_type', 'count']
df_payments

colors = sns.color_palette("viridis", len(df_payments))

plt.figure(figsize=(15, 6))
plt.barh(df_payments['payment_type'], df_payments['count'], color=colors)
plt.xlabel('Count')
plt.ylabel('Payment Type')
plt.title('Distribution of Payment Types')
plt.show()
```



3.1.9. Load the taxi zones shapefile and display it

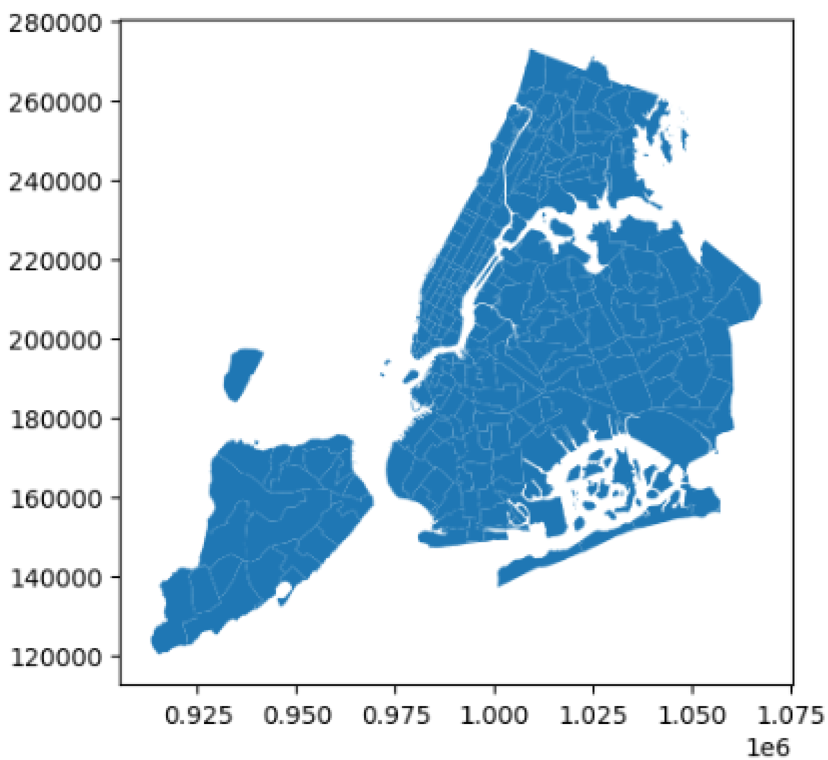
```python
import geopandas as gpd

df = pd.read_parquet('/content/cleaned_data.parquet')
# Read the shapefile using geopandas
zones = gpd.read_file("/content/taxi_zones.shp")# read the .shp file using gpd
zones.head()
```

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

Next steps: Generate code with zones View recommended plots New interactive sheet

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

```
<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 zone data with trips data

```python
# Merge zones and trip records using locationID and PULocationID
df_merged = pd.merge(df, zones, left_on='PULocationID', right_on='LocationID')
print(df.columns)
print(zones.columns)
df_merged.loc[:,['LocationID', 'PULocationID', 'DOLocationID']]
```

```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'RatecodeID', 'PULocationID',
       'DOLocationID', 'payment_type', 'fare_amount', 'extra', 'mta_tax',
       'tip_amount', 'tolls_amount', 'improvement_surcharge', 'total_amount',
       'congestion_surcharge', 'Airport_Fee', 'trip_duration'],
      dtype='object')
Index(['OBJECTID', 'Shape_Leng', 'Shape_Area', 'zone', 'LocationID', 'borough',
       'geometry'],
      dtype='object')
```

	LocationID	PULocationID	DOLocationID
0	141	141	140
1	138	138	256
2	161	161	237
3	237	237	141
4	143	143	142
...	...	...	...
1841592	48	48	25
1841593	263	263	262
1841594	161	161	261
1841595	79	79	137
1841596	142	142	261

1841597 rows × 3 columns

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

```python
# Merge trip counts back to the zones GeoDataFrame
df_merged = pd.merge(df_merged, trip_counts, left_on='LocationID', right_on='LocationID')
df_merged[df_merged['LocationID'] == 141].shape #validated

df_merged = gpd.GeoDataFrame(df_merged, geometry='geometry', crs="EPSG:4326")
df_merged.head(10)
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amount	...	Airport_Fee	trip_duration	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough
0	2	2022-12-31 23:51:30	2022-12-31 23:56:06	1.0	0.86	1.0	141	140	1	6.50	...	0.00	4.600000	141	0.041514	0.000077	Lenox Hill West	141	Manhattan
1	2	2023-01-01 00:07:18	2023-01-01 00:23:15	1.0	7.74	1.0	138	256	2	32.40	...	1.25	15.950000	138	0.107467	0.000537	LaGuardia Airport	138	Queens
2	2	2023-01-01 00:16:41	2023-01-01 00:21:46	2.0	1.24	1.0	161	237	1	7.90	...	0.00	5.083333	161	0.035804	0.000072	Midtown Center	161	Manhattan
3	2	2023-01-01 00:14:03	2023-01-01 00:24:36	3.0	1.44	1.0	237	141	2	11.40	...	0.00	10.550000	237	0.042213	0.000096	Upper East Side South	237	Manhattan
4	2	2023-01-01 00:24:30	2023-01-01 00:29:55	1.0	0.54	1.0	143	142	2	6.50	...	0.00	5.416667	143	0.054180	0.000151	Lincoln Square West	143	Manhattan
5	2	2023-01-01 00:43:00	2023-01-01 01:01:00	1.0	19.24	1.0	66	107	1	25.64	...	0.00	18.000000	66	0.054633	0.000108	DUMBO/Vinegar Hill	66	Brooklyn
6	1	2023-01-01 00:42:56	2023-01-01 01:16:33	2.0	7.10	1.0	246	37	1	34.50	...	0.00	33.616667	246	0.069467	0.000281	West Chelsea/Hudson Yards	246	Manhattan
7	2	2023-01-01 00:58:00	2023-01-01 01:08:31	2.0	1.59	1.0	79	164	1	11.40	...	0.00	10.516667	79	0.042625	0.000108	East Village	79	Manhattan
8	2	2023-01-01 00:16:06	2023-01-01 00:31:59	1.0	3.16	1.0	79	256	1	19.10	...	0.00	15.883333	79	0.042625	0.000108	East Village	79	Manhattan

Number of trips per zone =>

```python
trip_counts
```

	LocationID	trip_count
0	132	95145
1	237	87913
2	161	86622
3	236	78945
4	162	66092
...	...	...
248	115	1
249	84	1
250	156	1
251	172	1
252	187	1

253 rows × 2 columns

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

Complete in the above two steps

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

```
[ ] df_simplified = df_merged.copy()
    df_simplified['geometry'] = df_simplified['geometry'].simplify(tolerance=0.001, preserve_topology=True)

    # Create figure and axis
    fig, ax = plt.subplots(1, 1, figsize=(12, 10))

    # Plot choropleth
    df_simplified.plot(
        column='trip_count',
        cmap='YlOrRd',
        linewidth=0.5,
        edgecolor='0.5',
        ax=ax,
        legend=True,
        legend_kwds={
            'label': "Number of Trips",
            'orientation': "vertical"
        },
        missing_kwds={
            "color": "lightgrey",
            "edgecolor": "white",
            "hatch": "///",
            "label": "No data"
        }
    )

    # Title and cleanup
    ax.set_title("Zone-wise Trips", fontsize=16, fontweight='bold')
    ax.axis('off')

    plt.show()
```

```
trip_counts
```

	LocationID	trip_count
0	132	95145
1	237	87913
2	161	86622
3	236	78945
4	162	66092
...	...	...
248	115	1
249	84	1
250	156	1
251	172	1
252	187	1

253 rows × 2 columns

3.1.14. Conclude with results

Busiest hour is 6pm, busiest or max pickup day is Thursday and max pickup occur in May.

Revenue is highest in May, but quarter wise it is highest in 4^{th} Q.

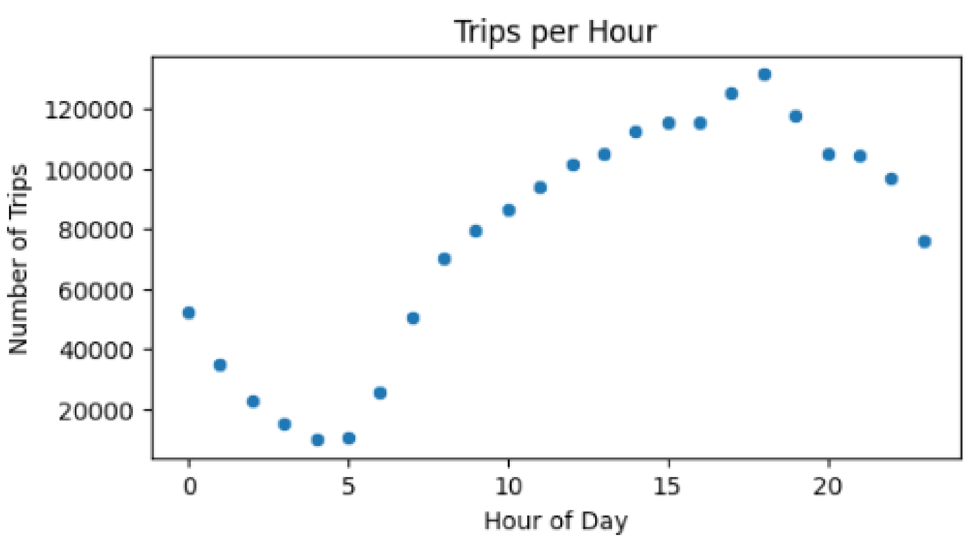
3.2. Detailed EDA: Insights and Strategies

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

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

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

df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour

# Count trips per hour and reset index
df_hr = df['pickup_hour'].value_counts().sort_values(ascending = False).reset_index()
df_hr.columns = ['pickup_hour', 'count']
#df_hr
print(df_hr.head(1)) #gives the hour at which the pickups are max


# Scatter plot
plt.figure(figsize=(6, 3))
sns.scatterplot(data=df_hr, x='pickup_hour', y='count', marker='o')
plt.title("Trips per Hour")
plt.xlabel("Hour of Day")
plt.ylabel("Number of Trips")
plt.show()
```

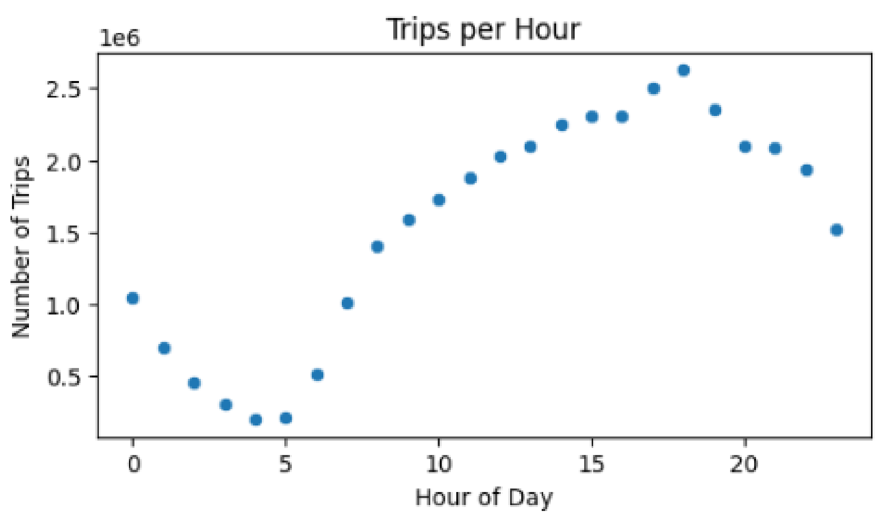
```
   pickup_hour   count
0           18  131416
```



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

```python
# Scale up the number of trips

# Fill in the value of your sampling fraction and use that to scale up the numbers
# Visualise the number of trips per hour and find the busiest hour
sampling_ratio = 0.05   # Change to your actual fraction
scale_factor = 1 / sampling_ratio

df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour

# Count trips per hour and reset index
df_hr = df['pickup_hour'].value_counts().sort_values(ascending = False).reset_index()
df_hr.columns = ['pickup_hour', 'count']
#df_hr
df_hr['count_scaled'] = df_hr['count'] * scale_factor
print(df_hr.head(5)) #gives the top 5 hours at which the pickups are max

# Scatter plot
plt.figure(figsize=(6, 3))
sns.scatterplot(data=df_hr, x='pickup_hour', y='count_scaled', marker='o')
plt.title("Trips per Hour")
plt.xlabel("Hour of Day")
plt.ylabel("Number of Trips")
plt.show()
```

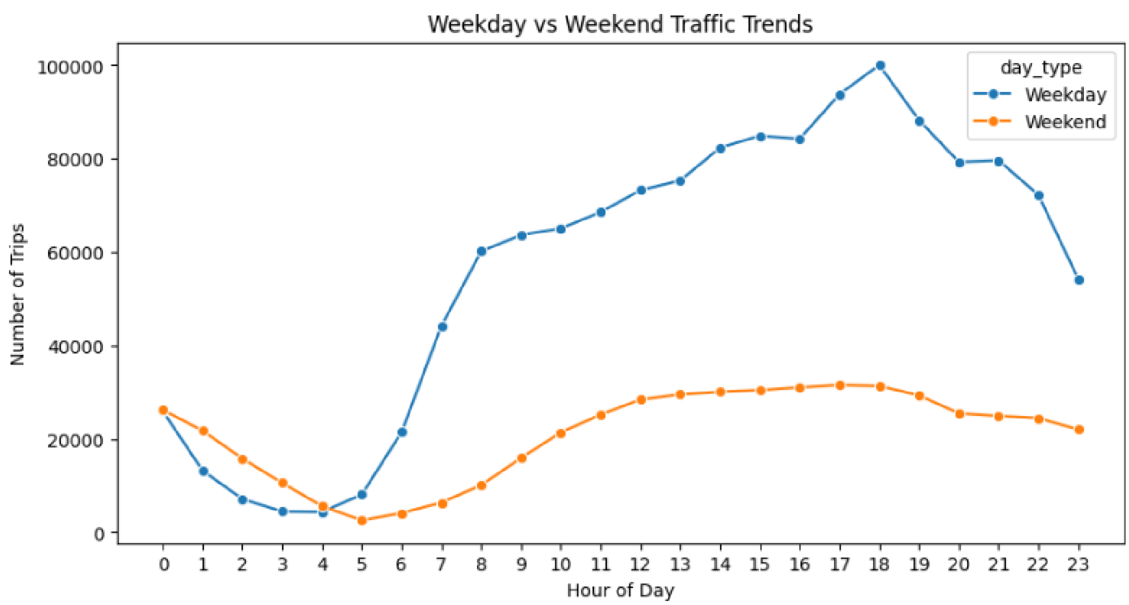
```
   pickup_hour   count  count_scaled
0           18  131416     2628320.0
1           17  125354     2507080.0
2           19  117592     2351840.0
3           15  115349     2306980.0
4           16  115301     2306020.0
```



3.2.4. Compare hourly traffic on weekdays and weekends

Trend to be compared for avg weekdays vs avg weekends:

```
# Compare traffic trends for the week days and weekends
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour
df['pickup_day'] = df['tpep_pickup_datetime'].dt.dayofweek
df['day_type'] = df['pickup_day'].apply(lambda x: 'Weekend' if x >= 5 else 'Weekday')

# Count trips per hour for each type
df_hourly = df.groupby(['day_type', 'pickup_hour']).size().reset_index(name='count')

# Plot
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 5))
sns.lineplot(data=df_hourly, x='pickup_hour', y='count', hue='day_type', marker='o')
plt.title("Weekday vs Weekend Traffic Trends")
plt.xlabel("Hour of Day")
plt.ylabel("Number of Trips")
plt.xticks(range(0, 24))
plt.show()
```



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

```
# Find top 10 pickup and dropoff zones
#df['PULocationID'] #I need top 10 location ids whose hourly pickups are at max #we already have a column pick_hour
# Count hourly pickups
pickup_counts = df.groupby(['PULocationID', 'pickup_hour']).size().reset_index(name='count')

# Total pickups per zone (regardless of hour)
top_pickup_zones = pickup_counts.groupby('PULocationID')['count'].sum().nlargest(10).index

# Filter only top pickup zones
pickup_top10 = pickup_counts[pickup_counts['PULocationID'].isin(top_pickup_zones)]
high_hr_PU = pickup_top10['PULocationID'].unique()
high_hr_PU
```

```
array([132, 138, 142, 161, 162, 170, 186, 230, 236, 237])
```

Top 10 zones that has high hourly pickups: [132, 138, 142, 161, 162, 170, 186, 230, 236, 237]

Top 10 zones that has hourly drops: [68, 141, 142, 161, 162, 170, 230, 236, 237, 239]

```
# Find top 10 pickup and dropoff zones
#df['PULocationID'] #I need top 10 location ids whose hourly pickups are at max #we already have a column pick_hour
# Count hourly pickups
do_counts = df.groupby(['DOLocationID', 'pickup_hour']).size().reset_index(name='count')

# Total pickups per zone (regardless of hour)
do_counts_zones = do_counts.groupby('DOLocationID')['count'].sum().nlargest(10).index

# Filter only top pickup zones
do_top10 = do_counts[do_counts['DOLocationID'].isin(do_counts_zones)]
high_hr_DO = do_top10['DOLocationID'].unique()
high_hr_DO
```

```
array([ 68, 141, 142, 161, 162, 170, 230, 236, 237, 239])
```

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

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

Pickup_Counts = df['PULocationID'].value_counts()
dropoff_counts = df['DOLocationID'].value_counts()

zones_merged = pd.merge(Pickup_Counts, dropoff_counts, left_on='PULocationID',
    right_on='DOLocationID',
    how='outer')
zones_merged.columns = ['Pickup_Counts', 'dropoff_counts']
zones_merged = zones_merged.fillna(0)
zones_merged['pickup_dropoff_ratio'] = zones_merged['Pickup_Counts'] / zones_merged['dropoff_counts']
zones_merged.sort_values('pickup_dropoff_ratio', ascending = False, inplace = True)
print('10 highest ratio')
print(zones_merged.head(10))
print('10 lowest ratio')
print(zones_merged.tail(10))
```

```
10 highest ratio
     Pickup_Counts  dropoff_counts  pickup_dropoff_ratio
194            2.0             0.0                   inf
69          8246.0           893.0              9.234043
127        95145.0         21056.0              4.518665
133        63939.0         24066.0              2.656819
181        63663.0         40601.0              1.568016
109        24699.0         17927.0              1.377754
42         31074.0         22664.0              1.371073
244        41265.0         30987.0              1.331687
157        66092.0         53008.0              1.246831
99         30392.0         25553.0              1.189371
10 lowest ratio
     Pickup_Counts  dropoff_counts  pickup_dropoff_ratio
151            1.0            26.0              0.038462
240            1.0            31.0              0.032258
246            1.0            34.0              0.029412
26             1.0            38.0              0.026316
0             49.0          5566.0              0.008803
171            0.0            13.0              0.000000
105            0.0            26.0              0.000000
216            0.0            35.0              0.000000
98             0.0             3.0              0.000000
29             0.0            18.0              0.000000
```

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

```python
# 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
df.columns
df_night = df[(df['pickup_hour'] >= 23) | (df['pickup_hour'] <= 5)]
df_night_1 = df_night.groupby('PULocationID').size().reset_index(name='count')
df_night_1.sort_values('count', ascending = False, inplace = True)
t1opu = df_night_1.head(10)['PULocationID'].tolist()
l10pu = df_night_1.tail(10)['PULocationID'].tolist()

df_night_2 = df_night.groupby('DOLocationID').size().reset_index(name='count')
df_night_2.sort_values('count', ascending = False, inplace = True)
t1odo = df_night_2.head(10)['DOLocationID'].tolist()
l10do = df_night_2.tail(10)['DOLocationID'].tolist()

print(f"Top 10 PU: {t1opu}")
print(f"Low 10 PU: {l10pu}")
print(f"Top 10 DO: {t1odo}")
print(f"Low 10 DO: {l10do}")
```

```
Top 10 PU: [79, 132, 249, 48, 148, 114, 230, 186, 164, 68]
Low 10 PU: [81, 44, 31, 3, 9, 192, 128, 185, 171, 153]
Top 10 DO: [79, 48, 170, 68, 107, 141, 263, 249, 230, 239]
Low 10 DO: [111, 253, 30, 172, 59, 187, 240, 44, 176, 99]
```

Top 10 zones with high traffic during night hours: [79, 132, 249, 48, 148, 114, 230, 186, 164, 68]

3.2.8. Find the revenue share for nighttime and daytime hours

```python
# Filter for night hours (11 PM to 5 AM)
revenue_night = df_night['total_amount'].sum()
print(f"Night revenue: {revenue_night}")

df_day = df[~((df['pickup_hour'] >= 23) | (df['pickup_hour'] <= 5))]  # Opposite of night hours
revenue_day = df_day['total_amount'].sum()
print(f"Day revenue: {revenue_day}")
```

```
Night revenue: 6526865.180000001
Day revenue: 47095377.33999998
```

Night revenue: 6526865.180000001

Day revenue: 47095377.33999998

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

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

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

# Avoid division by zero or NaN
df = df.replace([float('inf'), -float('inf')], pd.NA).dropna(subset=['fare_per_mile_per_passenger'])

# 2. Group by passenger count and calculate average
fare_analysis = (
    df.groupby('passenger_count')['fare_per_mile_per_passenger']
    .mean()
    .reset_index()
    .sort_values(by='fare_per_mile_per_passenger', ascending=False)
)

print(fare_analysis)
```

```
   passenger_count  fare_per_mile_per_passenger
0              1.0                    10.786594
1              2.0                     6.432401
3              4.0                     4.363227
2              3.0                     3.908099
4              5.0                     1.709614
5              6.0                     1.350744
```

	passenger_count	fare_per_mile_per_passenger
0	1.0	10.786594
1	2.0	6.432401
3	4.0	4.363227
2	3.0	3.908099
4	5.0	1.709614
5	6.0	1.350744

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

```
Average Fare per Mile by Hour:
    hour  fare_per_mile
0      0      10.372547
1      1      11.205206
2      2       9.875312
3      3      10.802813
4      4      13.234854
5      5      13.894556
6      6      10.987407
7      7      10.154647
8      8      10.308118
9      9      10.466256
10    10      10.739128
11    11      10.939761
12    12      11.696945
13    13      11.939735
14    14      11.546389
15    15      12.498474
16    16      13.817502
17    17      11.966156
18    18      11.536954
19    19      11.446739
20    20       9.560997
21    21       9.485441
22    22      10.126020
23    23      10.711384
/nAverage Fare per Mile by Day of Week:
  day_of_week  fare_per_mile
0      Monday      10.928148
1     Tuesday      11.319571
2   Wednesday      11.041087
3    Thursday      11.174598
4      Friday      10.905138
5    Saturday      10.760392
6      Sunday      12.506695
```

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

```python
# Compare fare per mile for different vendors
df_vendor = df.loc[:,['VendorID','fare_per_mile']]
avg_fare_vendor = df_vendor.groupby('VendorID')['fare_per_mile'].mean().reset_index()
avg_fare_vendor
```

	VendorID	fare_per_mile
0	1	8.125524
1	2	12.302985
2	6	6.282738

There are 3 vendors with Vendor ID 1, 2 and 6. Their fare_per_mile is calculated as in the fig.

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

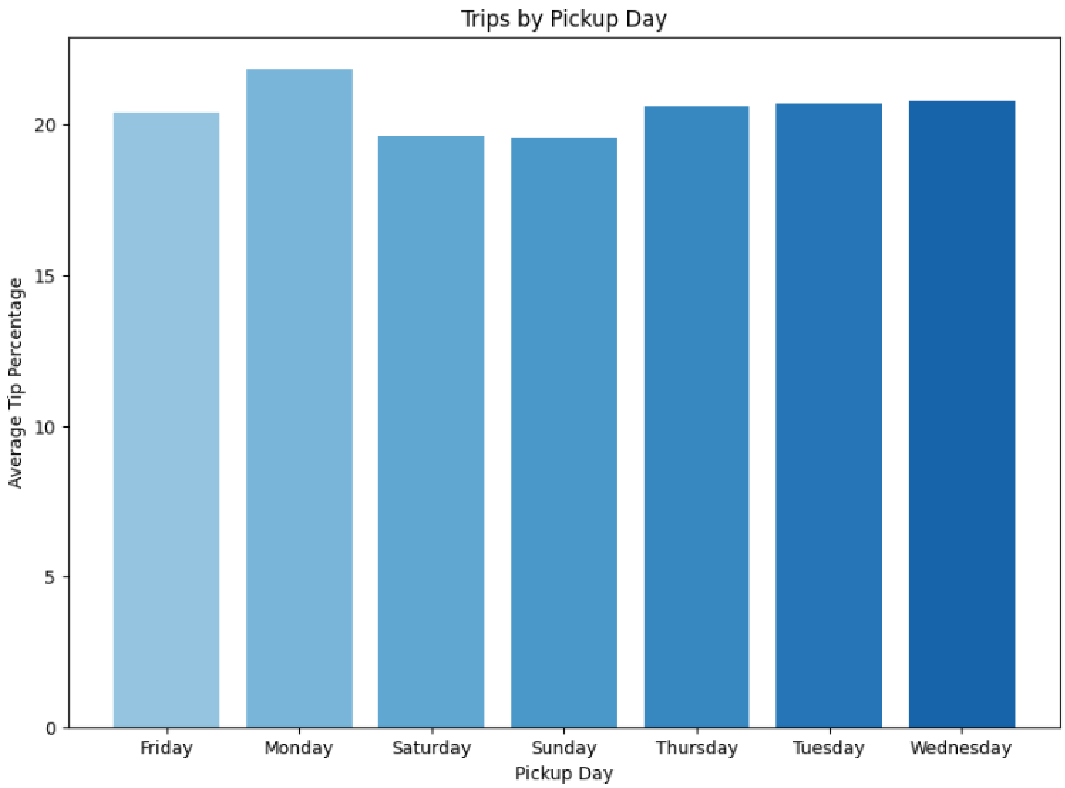
Mentioned that the tiers are in the following manner =>

```python
# Defining distance tiers
def distance_tier(d):
    if d <= 2:
        return 'Up to 2 miles'
    elif d <= 5:
        return '2 to 5 miles'
    else:
        return 'More than 5 miles'

df['distance_tier'] = df['trip_distance'].apply(distance_tier)

# Group by VendorID and distance tier
tier_analysis = (
    df.groupby(['VendorID', 'distance_tier'])['fare_per_mile']
    .mean()
    .reset_index()
    .sort_values(['distance_tier', 'fare_per_mile'], ascending=[True, False])
)

print(tier_analysis.sort_values(by=['distance_tier', 'VendorID']))
```

```
   VendorID       distance_tier  fare_per_mile
0         1        2 to 5 miles       6.382426
3         2        2 to 5 miles       6.538461
6         6        2 to 5 miles       8.107119
1         1   More than 5 miles       4.426758
4         2   More than 5 miles       4.490539
7         6   More than 5 miles       4.382026
2         1       Up to 2 miles       9.911689
5         2       Up to 2 miles      17.940902
8         6       Up to 2 miles      32.422471
```

3.2.13. Analyse the tip percentages

From the chart attached we clearly see that on an average maximum percent of tips are rewarded on Mondays.



3.2.14. Analyse the trends in passenger count

```python
# Compare trips with tip percentage < 10% to trips with tip percentage > 25%


low_tips = df[df['tip_percentage'] < 10]
high_tips = df[df['tip_percentage'] > 25]

# Compare averages for key features
comparison = pd.DataFrame({
    'Low Tip % (<10)': [
        low_tips['trip_distance'].mean(),
        low_tips['passenger_count'].mean(),
        low_tips['fare_amount'].mean(),
        low_tips['tip_percentage'].mean()
    ],
    'High Tip % (>25)': [
        high_tips['trip_distance'].mean(),
        high_tips['passenger_count'].mean(),
        high_tips['fare_amount'].mean(),
        high_tips['tip_percentage'].mean()
    ]
}, index=['Avg Trip Distance', 'Avg Passenger Count', 'Avg Fare Amount', 'Avg Tip Percentage'])

print(comparison)
```

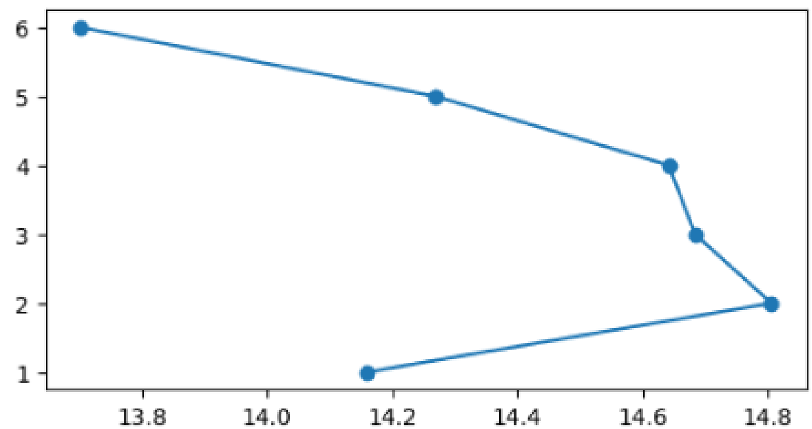
```
                     Low Tip % (<10)  High Tip % (>25)
Avg Trip Distance           3.938333          2.310612
Avg Passenger Count         1.417324          1.358722
Avg Fare Amount            21.665446         14.440492
Avg Tip Percentage          1.083931         32.508941
```

3.2.15. Analyse the variation of passenger counts across zones

As we clearly see different average of hours show population pickups.

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

#df.columns
df_passenger_hr = df.groupby('passenger_count')['pickup_hour'].mean().reset_index()
df_passenger_hr.columns = ['passenger_count', 'pickup_hour']

plt.figure(figsize = (6,3))
plt.plot(df_passenger_hr['pickup_hour'], df_passenger_hr['passenger_count'], marker='o')
```

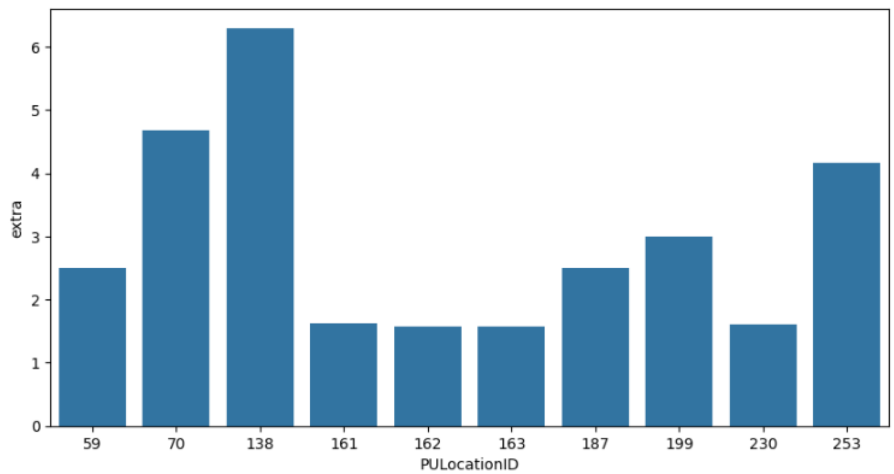
```
[<matplotlib.lines.Line2D at 0x7f2fd0deb850>]
```



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

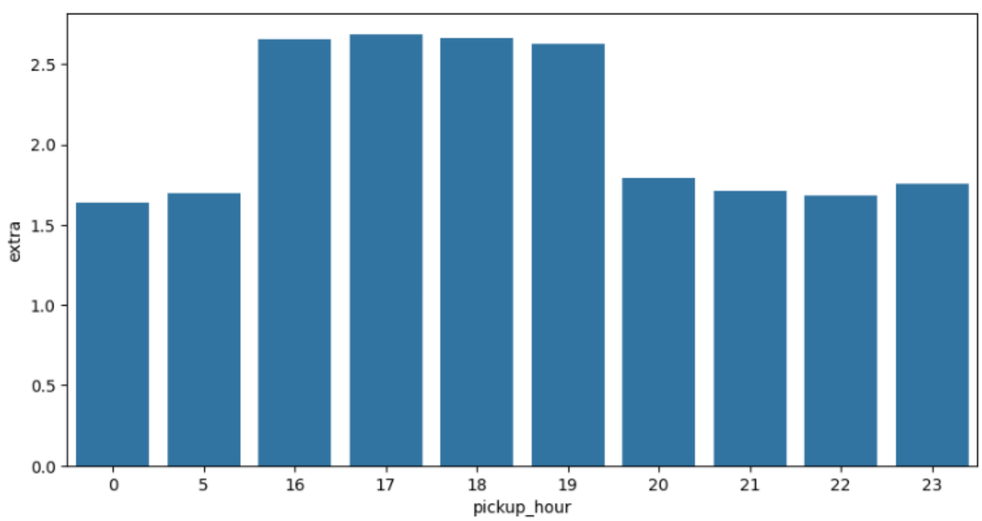
```python
# How often is each surcharge applied?

df.loc[:,['PULocationID','extra']]

#by zones
zone_extra = df.groupby('PULocationID')['extra'].mean().reset_index().sort_values('extra', ascending=False)

#by time
hourly_extra = df.groupby('pickup_hour')['extra'].mean().reset_index().sort_values('extra', ascending=False)

#by both
zone_time_extra = (
    df.groupby(['PULocationID', 'pickup_hour'])['extra']
      .mean()
      .reset_index()
      .sort_values('extra', ascending=False)
)

#top 10 zones
plt.figure(figsize=(10, 5))
sns.barplot(data=zone_extra.head(10), x='PULocationID', y='extra')
```

```
<Axes: xlabel='PULocationID', ylabel='extra'>
```




```python
[38] #top 10 zones
plt.figure(figsize=(10, 5))
sns.barplot(data=hourly_extra.head(10), x='pickup_hour', y='extra')
```

```
<Axes: xlabel='pickup_hour', ylabel='extra'>
```



4. Conclusions

4.1. Final Insights and Recommendations

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

#Based on the patterns, Pickup/dropoff hot spots can be predicted by historical frequency and fare profitability.
#Extra charges and high demand often correlate with certain times (rush hours, late nights, weekends).

#Operational inefficiency would probably be with respect to having lesser cabs positioned in the right zones or spots at the peak hours.
#Zones like 132 needs to have higher number of cabs positioned.

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.

#The high peaks are on Thursdays, and high peak hours are 6pm evenings. More cabs better be positioned to be around 6pm on Thursdays. Moderately higher number of cabs need to be positioned at around 6pm.

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

#Day revenues are generally higher than the night revenues. So prices can be kept keeping profits in mind during the day. More so during the peak hours, i.e. 6pm if we keep the prices upto Rs. 5 higher, there will be significant rise in revenue.

#VendorID 2, in general keeps higher fare amount in avaerage, so the revenue could be highly affected by this vendor, especially if the drive is upto 2 miles.