Report: Optimising NYC Taxi Operations

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

1. Data Preparation

1.1. Loading the dataset

1.1.1. Sample the data and combine the files

```
[28] 1 df.head(5)
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	...	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_
1164	2	2023-01-01 00:10:30	2023-01-01 00:11:49	1.0	0.49	1.0	N	239	238	1	...	0.5	1.88	0.00	1.0	
1868	2	2023-01-01 00:49:02	2023-01-01 00:55:15	1.0	0.75	1.0	N	45	148	2	...	0.5	0.00	0.00	1.0	
3106	1	2023-01-01 00:47:17	2023-01-01 01:07:01	2.0	2.90	1.0	N	142	170	1	...	0.5	4.80	0.00	1.0	
808	2	2023-01-01 00:06:02	2023-01-01 00:31:38	1.0	2.50	1.0	N	43	161	1	...	0.5	7.25	0.00	1.0	
169	2	2023-01-01 00:02:19	2023-01-01 00:30:49	1.0	20.37	2.0	N	132	140	1	...	0.5	12.00	6.55	1.0	

5 rows × 22 columns

2. Data Cleaning

2.1. Fixing Columns

2.1.1. Fix the index

```
1 # Fix the index and drop any columns that are not needed
2 df=df.reset_index().drop('index',axis=1)
3 df
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	...	mta_tax	tip_amount	tolls_amount	improvement_surcharge	tot
0	2	2023-01-01 00:10:30	2023-01-01 00:11:49	1.0	0.49	1.0	N	239	238	1	...	0.5	1.88	0.00	1.0	
1	2	2023-01-01 00:49:02	2023-01-01 00:55:15	1.0	0.75	1.0	N	45	148	2	...	0.5	0.00	0.00	1.0	
2	1	2023-01-01 00:47:17	2023-01-01 01:07:01	2.0	2.90	1.0	N	142	170	1	...	0.5	4.80	0.00	1.0	
3	2	2023-01-01 00:06:02	2023-01-01 00:31:38	1.0	2.50	1.0	N	43	161	1	...	0.5	7.25	0.00	1.0	
4	2	2023-01-01 00:02:19	2023-01-01 00:30:49	1.0	20.37	2.0	N	132	140	1	...	0.5	12.00	6.55	1.0	

...

2.1.2. Combine the two airport_fee columns

```python
1 # Combine the two airport fee columns
2 df['airport_fee']=df['airport_fee'].fillna(df['Airport_fee'])
3
```

```python
1 df.head(5)
```

ount	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	...	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount	congestion_surcharge	airport_fee	date	hour	Airport_fee
1.0	0.49	1.0	N	239	238	1	...	0.5	1.88	0.00	1.0	11.28	2.5	0.0	2023-01-01	0	NaN
1.0	0.75	1.0	N	45	148	2	...	0.5	0.00	0.00	1.0	12.90	2.5	0.0	2023-01-01	0	NaN
2.0	2.90	1.0	N	142	170	1	...	0.5	4.80	0.00	1.0	28.90	2.5	0.0	2023-01-01	0	NaN
1.0	2.50	1.0	N	43	161	1	...	0.5	7.25	0.00	1.0	36.25	2.5	0.0	2023-01-01	0	NaN
1.0	20.37	2.0	N	132	140	1	...	0.5	12.00	6.55	1.0	92.55	2.5	0.0	2023-01-01	0	NaN

2.2. Handling Missing Values

2.2.1. Find the proportion of missing values in each column

```python
1 # Find the proportion of missing values in each column
2 df.isna().sum()/len(df)*100
```

	0
VendorID	0.000000
tpep_pickup_datetime	0.000000
tpep_dropoff_datetime	0.000000
passenger_count	3.294860
trip_distance	0.000000
RatecodeID	3.294860
store_and_fwd_flag	3.294860
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	3.294860
airport_fee	3.294860
date	0.000000
hour	0.000000
Airport_fee	11.121979

2.2.2. Handling missing values in passenger_count

```python
9]    1 df['passenger_count'].median()
```

```
1.0
```

```python
      1 df['passenger_count'].fillna(df['passenger_count'].median(),inplace=True)
```

```python
1]    1 df['passenger_count'].isna().sum()
```

```
0
```

2.2.3. Handle missing values in RatecodeID

Handle missing values in `RatecodeID`

[55]
```
# Fix missing values in 'RatecodeID'
df['RatecodeID'].isna().sum()
```
```
9466
```

[56]
```
df['RatecodeID'].median()
```
```
1.0
```

```
df['RatecodeID'].fillna(df['RatecodeID'].median(),inplace=True)
```

[58]
```
df['RatecodeID'].isna().sum()
```
```
0
```

2.2.4. Impute NaN in congestion_surcharge

2.2.4 [3 marks]

Impute NaN in `congestion_surcharge`

```
# handle null values in congestion_surcharge
df['congestion_surcharge'].isnull().sum()
```
```
9466
```

[60]
```
df['congestion_surcharge'].median()
```
```
2.5
```

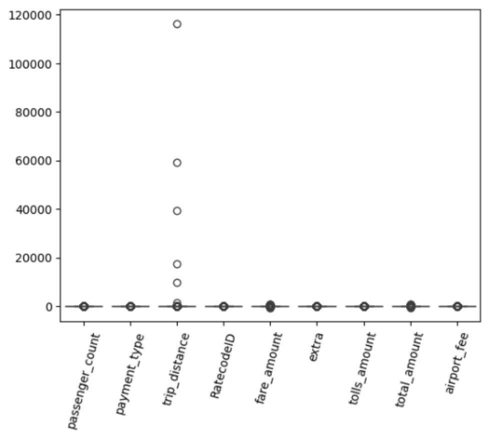
[61]
```
df['congestion_surcharge'].fillna(df['congestion_surcharge'].median(),inplace= True)
```

[62]
```
df['congestion_surcharge'].isnull().sum()
```
```
0
```

2.3. Handling Outliers and Standardising Values

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

```
1 sns.boxplot(data=df[['passenger_count','payment_type','trip_distance','RatecodeID','fare_amount','extra','tolls_amount','total_amount','
2 plt.xticks(rotation=75)
3 plt.show()
```



3. Exploratory Data Analysis

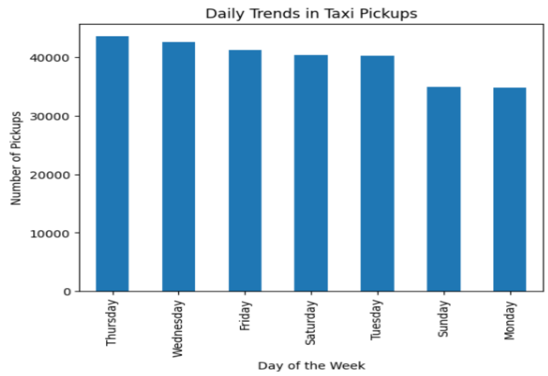
3.1. General EDA: Finding Patterns and Trends

3.1.1. Classify variables into categorical and numerical

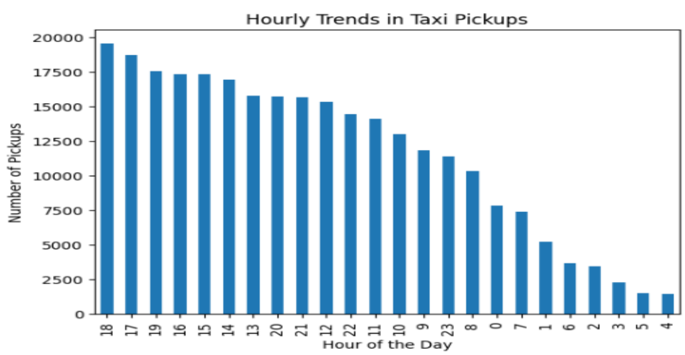
```
**3.1.1** <font color = red>[3 marks]</font> <br>
Categorise the varaibles into Numerical or Categorical.
* `VendorID`:Categorical
* `tpep_pickup_datetime`:Numerical
* `tpep_dropoff_datetime`:Numerical
* `passenger_count`:Numerical
* `trip_distance`:Numerical
* `RatecodeID`:Categorical
* `PULocationID`:Categorical
* `DOLocationID`:Categorical
* `payment_type`:Categorical
* `pickup_hour`:Numerical
* `trip_duration`:Numerical


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


* `fare_amount:Numerical`
* `extra`:Numerical
* `mta_tax`:Numerical
* `tip_amount`:Numerical
* `tolls_amount`:Numerical
* `improvement_surcharge`:Numerical
* `total_amount`:Numerical
* `congestion_surcharge`:Numerical
* `airport_fee`:Numerical
```

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

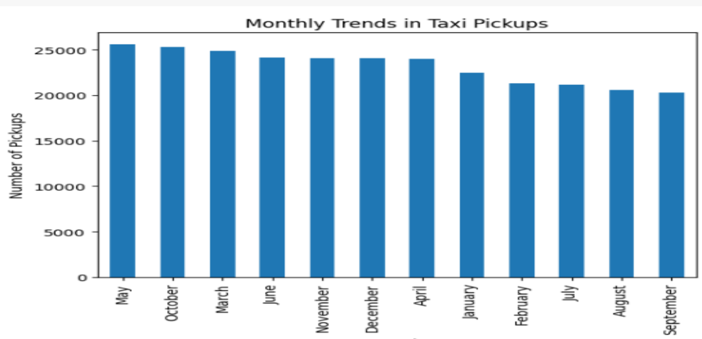
```python
# Find and show the daily trends in taxi pickups (days of the week)
df['p_weekdays'].value_counts().plot(kind='bar')
plt.title('Daily Trends in Taxi Pickups')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Pickups')
plt.show()

```




```python
# Find and show the hourly trends in taxi pickups
df['hour'].value_counts().plot(kind='bar')
plt.title('Hourly Trends in Taxi Pickups')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Pickups')
plt.show()

```




```python
# Show the monthly trends in pickups
df['p_month'].value_counts().plot(kind='bar')
plt.title('Monthly Trends in Taxi Pickups')
plt.xlabel('Month')
plt.ylabel('Number of Pickups')
plt.show()
```



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

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

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	...	improvement_surcharge	total_amount
0	2	2023-01-01 00:10:30	2023-01-01 00:11:49	1.0	0.49	1.0	N	239	238	1	...	1.0	11.28
2	1	2023-01-01 00:47:17	2023-01-01 01:07:01	2.0	2.90	1.0	N	142	170	1	...	1.0	28.90
3	2	2023-01-01 00:06:02	2023-01-01 00:31:38	1.0	2.50	1.0	N	43	161	1	...	1.0	36.25
4	2	2023-01-01 00:02:19	2023-01-01 00:30:49	1.0	20.37	2.0	N	132	140	1	...	1.0	92.55
7	2	2023-01-01 00:13:54	2023-01-01 00:21:26	1.0	2.03	1.0	N	142	75	1	...	1.0	18.84

3.1.4. Analyse the monthly revenue trends

Analyse the monthly revenue (`total_amount`) trend

```
[102]
# Group data by month and analyse monthly revenue
df.groupby('p_month')['total_amount'].sum().sort_values(ascending=False)
```

p_month	total_amount
October	599928.43
May	586593.79
June	555576.87
November	554538.87
March	549929.63
December	544568.74
April	541064.43
January	484348.61
September	474802.18
July	466185.51
February	460103.92
August	453896.60

dtype: float64

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

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

[103]

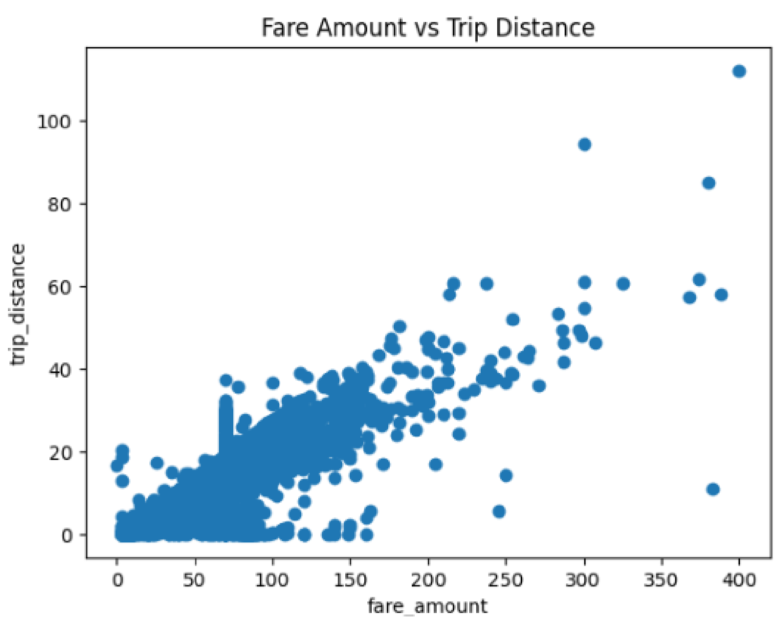
```
# Calculate proportion of each quarter
df.groupby('Quater')['total_amount'].sum().sort_values(ascending=False)/df['total_amount'].sum()*100
```

Quater	total_amount
4	27.091220
2	26.839273
1	23.828003
3	22.241504

dtype: float64

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

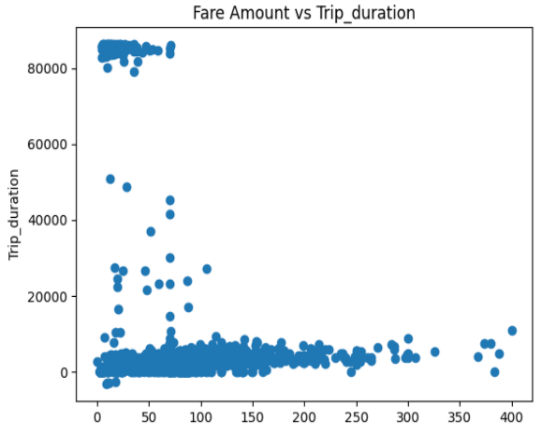
```
# Show how trip fare is affected by distance
plt.scatter(df['fare_amount'], df['trip_distance'])
plt.xlabel('fare_amount')
plt.ylabel('trip_distance')
plt.title('Fare Amount vs Trip Distance')
plt.show()
```



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

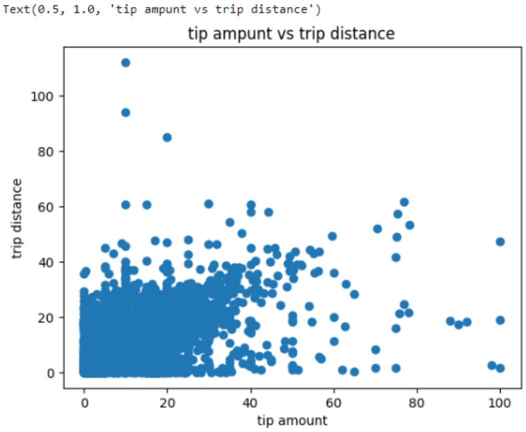
```python
plt.scatter(df['fare_amount'], df['trip_duration'])
plt.xlabel('Fare Amount')
plt.ylabel('Trip_duration')
plt.title('Fare Amount vs Trip_duration')
```

```
Text(0.5, 1.0, 'Fare Amount vs Trip_duration')
```



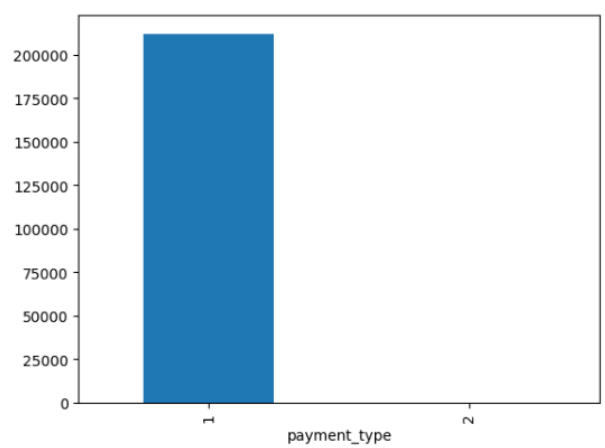

```python
# Show relationship between tip and trip distance
plt.scatter(df['tip_amount'],df['trip_distance'])
plt.xlabel('tip amount')
plt.ylabel('trip distance')
plt.title('tip ampunt vs trip distance')
```

```
Text(0.5, 1.0, 'tip ampunt vs trip distance')
```



3.1.8. Analyse the distribution of different payment types

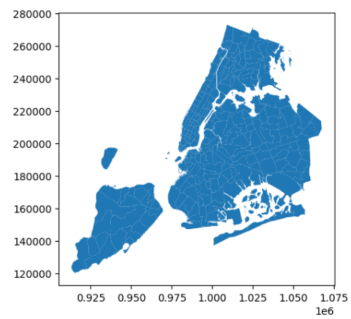
```python
# Analyse the distribution of different payment types (payment_type).
#sns.countplot(data=df['payment_type'])
df['payment_type'].value_counts().plot(kind='bar')
plt.show()

```



3.1.9. Load the taxi zones shapefile and display it

```
2  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

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

```
[115]
1 # Merge zones and trip records using locationID and PULocationID
2 df=pd.merge(df,zones,how='left',left_on='PULocationID',right_on='LocationID')
3 df.head()
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	...	Quater	p_weekdays	trip_duration	OBJECTID
0	2	2023-01-01 00:10:30	2023-01-01 00:11:49	1.0	0.49	1.0	N	239	238	1	...	1	Sunday	79.0	239.0
1	1	2023-01-01 00:47:17	2023-01-01 01:07:01	2.0	2.90	1.0	N	142	170	1	...	1	Sunday	1184.0	142.0
2	2	2023-01-01 00:06:02	2023-01-01 00:31:38	1.0	2.50	1.0	N	43	161	1	...	1	Sunday	1536.0	43.0
3	2	2023-01-01 00:02:19	2023-01-01 00:30:49	1.0	20.37	2.0	N	132	140	1	...	1	Sunday	1710.0	132.0
4	2	2023-01-01 00:13:54	2023-01-01 00:21:26	1.0	2.03	1.0	N	142	75	1	...	1	Sunday	452.0	142.0

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

```python
# Group data by location and calculate the number of trips
df.groupby('PULocationID')['trip_distance'].count()
```

PULocationID	trip_distance
1	8
4	209
7	39
8	1
10	58
...	...
261	1022
262	2928
263	4142
264	1830
265	28

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

```python
# Merge trip counts back to the zones GeoDataFrame
numberof_trip=df.groupby('PULocationID')['trip_distance'].count()
```

```python
numberof_trip=numberof_trip.to_frame().reset_index()
```

```python
numberof_trip.rename(columns={'trip_distance':'trip_count'},inplace=True)
numberof_trip
```

	PULocationID	trip_count
0	1	8
1	4	209
2	7	39
3	8	1
4	10	58
...	...	...
158	261	1022
159	262	2928
160	263	4142
161	264	1830
162	265	28

163 rows × 2 columns

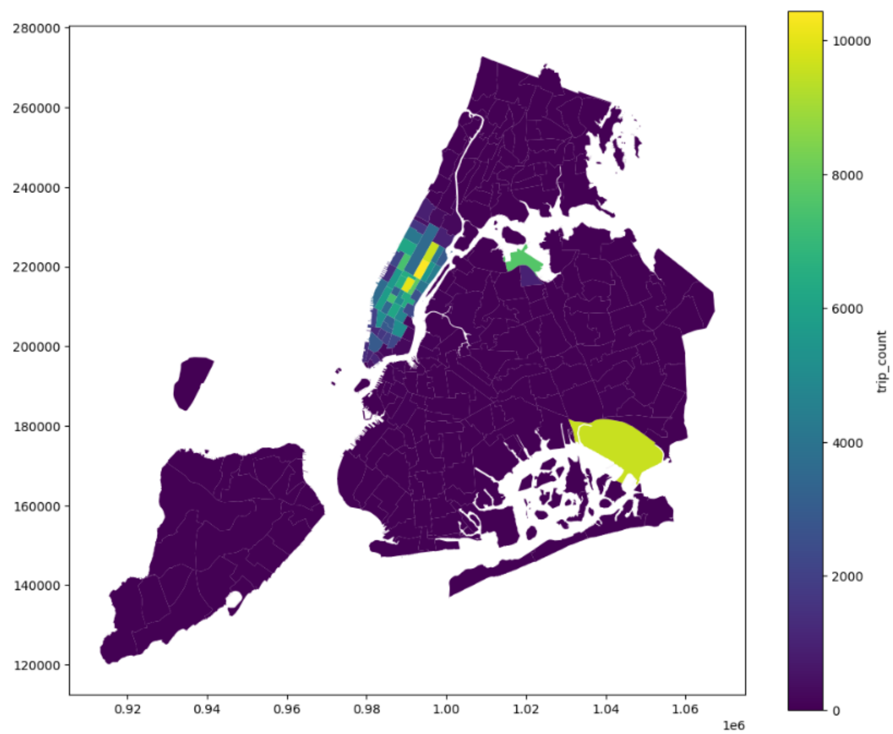
```python
zones = pd.merge(zones, numberof_trip, how='left', left_on='LocationID', right_on='PULocationID')
```

```python
zones
```

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry	PULocationID	trip_count
0	1	0.116357	0.000782	Newark Airport	1	EWR	POLYGON ((933100.918 192536.086, 933091.011 19...	1.0	8.0
1	2	0.433470	0.004866	Jamaica Bay	2	Queens	MULTIPOLYGON (((1033269.244 172126.008, 103343...	NaN	NaN
2	3	0.084341	0.000314	Allerton/Pelham Gardens	3	Bronx	POLYGON ((1026308.77 256767.698, 1026495.593 2...	NaN	NaN
3	4	0.043567	0.000112	Alphabet City	4	Manhattan	POLYGON ((992073.467 203714.076, 992068.667 20...	4.0	209.0
4	5	0.092146	0.000498	Arden Heights	5	Staten Island	POLYGON ((935843.31 144283.336, 936046.565 144...	NaN	NaN
...	...	...	...	...	...	...	...	...	...
258	259	0.126750	0.000395	Woodlawn/Wakefield	259	Bronx	POLYGON ((1025414.782 270986.139, 1025138.624 ...	NaN	NaN
259	260	0.133514	0.000422	Woodside	260	Queens	POLYGON ((1011466.966 216463.005, 1011545.889 ...	260.0	15.0
260	261	0.027120	0.000034	World Trade Center	261	Manhattan	POLYGON ((980555.204 196138.486, 980570.792 19...	261.0	1022.0
261	262	0.049064	0.000122	Yorkville East	262	Manhattan	MULTIPOLYGON (((999804.795 224498.527, 999824....	262.0	2928.0
262	263	0.037017	0.000066	Yorkville West	263	Manhattan	POLYGON ((997493.323 220912.386, 997355.264 22...	263.0	4142.0

263 rows × 9 columns

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



3.1.14. Conclude with results

-yellow coloured zone is the busy zone,better to incerase the number of taxis

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

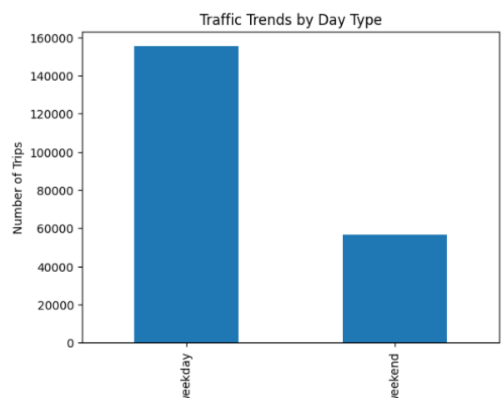
3.2.3. Scale up the number of trips from above to find the actual number of trips

```
1 route_speed[['zone','borough','LocationID','speed']].sort_values(by='speed', ascending=True).head(10)
```

	zone	borough	LocationID	speed
10923	Garment District	Manhattan	100.0	0.058128
51116	Yorkville East	Manhattan	262.0	0.077999
2	Newark Airport	EWR	1.0	0.087167
41436	UN/Turtle Bay South	Manhattan	233.0	0.090693
26069	Lower East Side	Manhattan	148.0	0.107049
15443	Hudson Sq	Manhattan	125.0	0.108294
31253	Midtown North	Manhattan	163.0	0.110633
40214	TriBeCa/Civic Center	Manhattan	231.0	0.135797
34315	Murray Hill	Manhattan	170.0	0.137430
14283	Greenwich Village South	Manhattan	114.0	0.180266

3.2.4. Compare hourly traffic on weekdays and weekends

```python
traffic_trends = df.groupby('day_type')['trip_distance'].count()
traffic_trends.plot(kind='bar')
plt.title('Traffic Trends by Day Type')
plt.xlabel('Day Type')
plt.ylabel('Number of Trips')
plt.show()
```



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

```python
print(top_pickup)
print(top_dropoff)
```

```
   LocationID  pickup_count                      zone
0         237         10441     Upper East Side South
1         161         10069            Midtown Center
2         236          9615     Upper East Side North
3         132          9580               JFK Airport
4         162          7798              Midtown East
5         138          7643         LaGuardia Airport
6         142          7383       Lincoln Square East
7         186          7317  Penn Station/Madison Sq West
8         230          6500     Times Sq/Theatre District
9         170          6381               Murray Hill
   LocationID  dropoff_count                      zone
0         236          10003    Upper East Side North
1         237           9552    Upper East Side South
2         161           8443           Midtown Center
3         170           6524              Murray Hill
4         239           6384    Upper West Side South
5         142           6269      Lincoln Square East
6         162           6064             Midtown East
7         141           5993           Lenox Hill West
8         230           5941  Times Sq/Theatre District
9         163           5425            Midtown North
```

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

```python
ratio=pickup_count['no_of_pickup']/dropoff_count['no_of_dropoff']
top10_r=ratio.sort_values(ascending=False).head(10)
bottom10_r=ratio.sort_values(ascending=True).head(10)
print(top10_r)
print(bottom10_r)
```

```
3     1.468424
12    1.251117
4     1.221491
7     1.220924
5     1.219174
6     1.217513
10    1.190299
13    1.178674
9     1.176221
14    1.165902
dtype: float64
129    0.030303
130    0.030303
131    0.031250
132    0.032258
133    0.033333
134    0.033333
135    0.033333
136    0.034483
137    0.035714
138    0.037037
dtype: float64
```

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

```python
# Get top 10 nighttime pickup zones
night_pkup = night_df.groupby('PULocationID')['trip_distance'].count().reset_index()
night_pkup.rename(columns={'trip_distance': 'no_of_pickups'}, inplace=True)
night_pkup = night_pkup.merge(zones[['LocationID', 'zone']], left_on='PULocationID', right_on='LocationID')
print(night_pkup[['zone','PULocationID','no_of_pickups']].nlargest(10, 'no_of_pickups'))

# Get top 10 nighttime dropoff zones
night_drop = night_df.groupby('DOLocationID')['trip_distance'].count().reset_index()
night_drop.rename(columns={'trip_distance': 'no_of_dropoffs'}, inplace=True)
night_drop = night_drop.merge(zones[['LocationID', 'zone']], left_on='DOLocationID', right_on='LocationID')
print(night_drop[['zone', 'no_of_dropoffs','DOLocationID']].nlargest(10, 'no_of_dropoffs'))
```

```
                          zone  PULocationID  no_of_pickups
30                East Village            79           1896
109               West Village           249           1557
50                 JFK Airport           132           1408
18                Clinton East            48           1240
63             Lower East Side           148           1196
44       Greenwich Village South           114           1083
97       Times Sq/Theatre District         230            869
79    Penn Station/Madison Sq West         186            790
74               Midtown South           164            735
41                    Gramercy           107            728
                           zone  no_of_dropoffs  DOLocationID
69                 East Village            1013            79
147                 Murray Hill             776           170
41                 Clinton East             764            48
92                     Gramercy             737           107
120              Lenox Hill West             675           141
224               Yorkville West             672           263
59                 East Chelsea             670            68
213                West Village             587           249
195   Sutton Place/Turtle Bay North          581           229
80                     Flatiron             563            90
```

3.2.8. Find the revenue share for nighttime and daytime hours

```
[292]
night_revenue=night_hour['total_amount'].sum()
night_revenue
```

```
746405.26
```

```
day_revenue=day_hour['total_amount'].sum()
day_revenue
```

```
5525442.629999999
```

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

```
# Step 2: Group by 'passenger_count' and calculate the average fare per mil
avg_fare_per_mile = df.groupby('passenger_count')['fare_per_mile'].mean()

# Display the result
print(avg_fare_per_mile)
```

```
   total_amount  trip_distance  fare_per_mile
0         11.28           0.49      23.020408
1         28.90           2.90       9.965517
2         36.25           2.50      14.500000
3         92.55          20.37       4.543446
4         18.84           2.03       9.280788
passenger_count
1.0    15.475222
2.0    16.807153
3.0    15.263719
4.0    28.835807
5.0    13.002575
6.0    15.024136
Name: fare_per_mile, dtype: float64
```

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
avg_fare_per_mile_hour=df.groupby('hour')['fare_per_mile'].mean()
print(avg_fare_per_mile_hour)
avg_fare_per_mile_day=df.groupby('p_weekdays')['fare_per_mile'].mean()
print(avg_fare_per_mile_day)

```

```
hour
0     17.676508
1     14.175588
2     13.553356
3     11.125841
4     12.100460
5     21.040608
6     11.694427
7     16.215085
8     14.122533
9     14.128000
10    16.579403
11    16.254315
12    14.754149
13    16.627545
14    17.079159
15    18.168978
16    19.761257
17    20.117908
18    16.487922
19    15.260735
20    12.623177
21    13.685330
22    13.151113
23    13.890847
Name: fare_per_mile, dtype: float64
p_weekdays
Friday       16.619339
Monday       15.498030
Saturday     15.329242
Sunday       16.785113
Thursday     16.316436
Tuesday      14.889567
Wednesday    15.735160
Name: fare_per_mile, dtype: float64
```

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

```
# Compare fare per mile for different vendors
avg_fare_per_mile_vendor=df.groupby('VendorID')['fare_per_mile'].mean()
print(avg_fare_per_mile_vendor)
```

```
VendorID
1    13.676290
2    16.631472
Name: fare_per_mile, dtype: float64
```

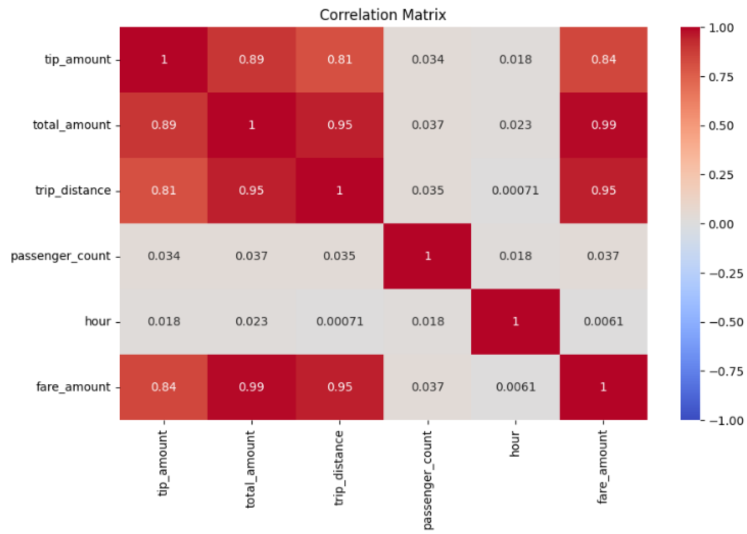
3.2.12. Compare the fare rates of different vendors in a distance-tiered fashion

```
print(avg_fare_per_mile_tier)
```

	distance_tier	VendorID	fare_per_mile
0	2 to 5 miles	1	9.536633
1	2 to 5 miles	2	9.765018
2	More than 5 miles	1	6.410897
3	More than 5 miles	2	6.445353
4	Up to 2 miles	1	17.370212
5	Up to 2 miles	2	23.453033

3.2.13. Analyse the tip percentages

```
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title("Correlation Matrix")
plt.show()
```



3.2.14. Analyse the trends in passenger count

```
# See how passenger count varies across hours and days
pc=df.groupby(['hour','p_weekdays'])['passenger_count'].count()
pc.sort_values(ascending=False)
```

hour	p_weekdays	passenger_count
18	Thursday	2563
	Wednesday	2554
	Tuesday	2444
17	Thursday	2363
19	Wednesday	2306
...	...	...
3	Monday	64
	Wednesday	62
4	Tuesday	60
	Wednesday	57
3	Tuesday	45

168 rows × 1 columns

dtype: int64

3.2.15. Analyse the variation of passenger counts across zones

```
# How does passenger count vary across zones
pc_z=df.groupby('zone')['passenger_count'].count()
pc_z.sort_values(ascending=False)

```

zone	passenger_count
Upper East Side South	10441
Midtown Center	10069
Upper East Side North	9615
JFK Airport	9580
Midtown East	7798
...	...
Cypress Hills	1
Flatlands	1
Brownsville	1
Rosedale	1
Fresh Meadows	1

161 rows × 1 columns

dtype: int64

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

```
often_charge=df.groupby('hour')['extra'].count()
often_charge
```

hour	extra
0	6002
1	3953
2	2620
3	1593
4	919
5	940
6	2584
7	5778
8	8123
9	8982
10	9559
11	10361
12	11241
13	11545
14	12381

4. Conclusions

4.1. Final Insights and Recommendations

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

- Actual busiest hours are 18,17,19,16,15
- Upper East Side South,Midtown Center,Upper East Side North,JFK Airport,Midtown East,this are the most pickuping point
- Garment District-Central Park,Yorkville East-Upper West Side North,Newark Airport,UN/Turtle -Bay South,Central Park are the most traiffc routes

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.

- weekday have more demand comapre to weeends
- JFK Airport,Central Park,   Upper West Side South this are the tope 3 demand zones
- may.oct,march month most of the trips are done

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

- its clear that trips during rush hour evening tend to have higher fares due to increased demand.
- Off-Peak Hours: Conversely, trips during early-morning hours may have lower demand, and pricing adjustments can encourage more drivers to be on the road.
- Frequent Rider Discounts: Provide a discount after a certain number of trips or offer a monthly subscription model for regular riders to increase revenue in quater 3