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

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocatio
0	1	2023-04-01 00:58:33	2023-04-01 01:07:03	4.0	1.10	1.0	Y	249	
1	1	2023-04-01 00:10:28	2023-04-01 00:27:17	1.0	3.00	1.0	N	230	
2	2	2023-04-01 00:54:11	2023-04-01 01:00:52	1.0	1.53	1.0	N	100	
3	1	2023-04-01 00:53:11	2023-04-01 01:04:05	2.0	1.50	1.0	N	90	
4	2	2023-04-01 00:38:39	2023-04-01 00:52:06	4.0	1.60	1.0	N	211	

5 rows × 22 columns

2. Data Cleaning

2.1. Fixing Columns

2.1.1. Fix the index

```
dfn.reset_index(drop=True, inplace=True)
dfn.head()
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	I
0	1	2023-04-01 00:58:33	2023-04-01 01:07:03	4.0	1.10	1.0	Y	249	
1	1	2023-04-01 00:10:28	2023-04-01 00:27:17	1.0	3.00	1.0	N	230	
2	2	2023-04-01 00:54:11	2023-04-01 01:00:52	1.0	1.53	1.0	N	100	
3	1	2023-04-01 00:53:11	2023-04-01 01:04:05	2.0	1.50	1.0	N	90	
4	2	2023-04-01 00:38:39	2023-04-01 00:52:06	4.0	1.60	1.0	N	211	

5 rows × 22 columns

2.1.2.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 236180 entries, 0 to 236179
Data columns (total 21 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   VendorID               236180 non-null  int64
 1   tpep_pickup_datetime   236180 non-null  datetime64[us]
 2   tpep_dropoff_datetime  236180 non-null  datetime64[us]
 3   passenger_count        229994 non-null  float64
 4   trip_distance          236180 non-null  float64
 5   RatecodeID             229994 non-null  float64
 6   store_and_fwd_flag     229994 non-null  object
 7   PULocationID           236180 non-null  int64
 8   DOLocationID           236180 non-null  int64
 9   payment_type           236180 non-null  int64
 10  fare_amount            236180 non-null  float64
 11  extra                  236180 non-null  float64
 12  mta_tax                236180 non-null  float64
 13  tip_amount             236180 non-null  float64
 14  tolls_amount           236180 non-null  float64
 15  improvement_surcharge  236180 non-null  float64
 16  total_amount           236180 non-null  float64
 17  congestion_surcharge   229994 non-null  float64
 18  date                   236180 non-null  object
 19  hour                   236180 non-null  int32
 20  airport_fee            229994 non-null  float64
dtypes: datetime64[us](2), float64(12), int32(1), int64(4), object(2)
memory usage: 36.9+ MB
```

2.2. Handling Missing Values

2.2.1. Find the proportion of missing values in each column

fare_per_mile_per_passenger	0.000157
fare_per_mile	0.000148
mta_tax	0.000043
improvement_surcharge	0.000043
congestion_surcharge	0.000035
tip_percentage	0.000035
trip_distance	0.000000
tpep_dropoff_datetime	0.000000
tpep_pickup_datetime	0.000000
VendorID	0.000000
passenger_count	0.000000
DOLocationID	0.000000
RatecodeID	0.000000

extra	0.000000
fare_amount	0.000000
tolls_amount	0.000000
tip_amount	0.000000
total_amount	0.000000
payment_type	0.000000
PULocationID	0.000000
store_and_fwd_flag	0.000000
airport_fee	0.000000
hour	0.000000
date	0.000000
pickup_hour	0.000000
trip_duration_min	0.000000
pickup_month	0.000000
day_type	0.000000
day_of_week	0.000000
weekday_name	0.000000
distance_tier	0.000000

2.2.2. Handling missing values in passenger_count

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DO
14	2	2023-04-01 00:19:21	2023-04-01 00:37:50	NaN	3.34	NaN	None	234	
15	2	2023-04-01 00:00:13	2023-04-01 00:28:39	NaN	5.03	NaN	None	158	
48	2	2023-04-01 01:20:09	2023-04-01 01:27:51	NaN	1.25	NaN	None	164	
78	2	2023-04-01 02:28:39	2023-04-01 02:47:10	NaN	4.65	NaN	None	261	
88	2	2023-04-01 02:32:25	2023-04-01 02:35:23	NaN	0.67	NaN	None	238	
...	...	...	...	...	...	...	...	...	
235996	2	2023-08-31 18:11:47	2023-08-31 18:58:46	NaN	5.66	NaN	None	225	
236038	1	2023-08-31 20:45:15	2023-08-31 21:28:06	NaN	0.00	NaN	None	50	
236081	1	2023-08-31 21:49:39	2023-08-31 21:56:15	NaN	1.20	NaN	None	249	
236142	1	2023-08-31 22:51:37	2023-08-31 23:01:25	NaN	1.20	NaN	None	90	

PULocationID	DOLocationID	payment_type	...	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount	congestion_surcharge
234	141	0	...	0.0	0.5	4.71	0.00	1.0	28.28	NaN
158	236	0	...	0.0	0.5	1.67	0.00	1.0	35.02	NaN
164	90	0	...	0.0	0.5	2.63	0.00	1.0	20.14	NaN
261	48	0	...	0.0	0.5	5.72	0.00	1.0	34.32	NaN
238	151	0	...	0.0	0.5	3.00	0.00	1.0	18.01	NaN
...	...	...	...	...	...	...	...	...	...	...
225	145	0	...	0.0	0.5	0.00	0.00	1.0	37.75	NaN
50	226	0	...	1.0	0.5	0.00	6.94	1.0	45.94	NaN
249	79	0	...	1.0	0.5	1.36	0.00	1.0	14.96	NaN
90	170	0	...	1.0	0.5	2.36	0.00	1.0	18.06	NaN

2.2.3. Handle missing values in RatecodeID

```
dfn['RatecodeID'].isnull().sum()
mode_value = df['RatecodeID'].mode()[0]
dfn['RatecodeID'].fillna(mode_value, inplace=True)
```

```
<ipython-input-218-e1624da4c821>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment usi
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values alwa

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(va


  dfn['RatecodeID'].fillna(mode_value, inplace=True)
```

2.2.4. Impute NaN in congestion_surcharge

```
<ipython-input-47-f69f56d13a4e>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment us
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values alw

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(v

  dfn['congestion_surcharge'].fillna(mode_value, inplace=True)
```

	congestion_surcharge
0	2.5
1	2.5
2	2.5
3	2.5
4	2.5
...	...
236175	2.5
236176	2.5

2.3. Handling Outliers and Standardising Values

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

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID
count	229986.000000	229986	229986	229986.000000	229986.000000	229986.000000	229986.000000	229986.000000
mean	1.725975	2023-04-26 12:59:56.471967	2023-04-26 13:16:37.055394	1.369049	0.033285	1.643074	165.272364	163.828450
min	1.000000	2023-01-01 00:04:34	2023-01-01 00:09:40	0.000000	0.000000	1.000000	1.000000	1.000000
25%	1.000000	2023-03-09 03:52:07	2023-03-09 04:36:23	1.000000	0.010355	1.000000	132.000000	113.000000
50%	2.000000	2023-04-14 18:38:15	2023-04-14 18:52:49	1.000000	0.017258	1.000000	162.000000	162.000000
75%	2.000000	2023-06-20 09:06:55	2023-06-20 09:24:46	1.000000	0.032598	1.000000	233.000000	234.000000
max	2.000000	2023-08-31 23:54:22	2023-09-01 00:26:50	7.000000	1.000000	99.000000	265.000000	265.000000
std	0.446023	NaN	NaN	0.896837	0.043669	7.452020	63.708239	69.902952

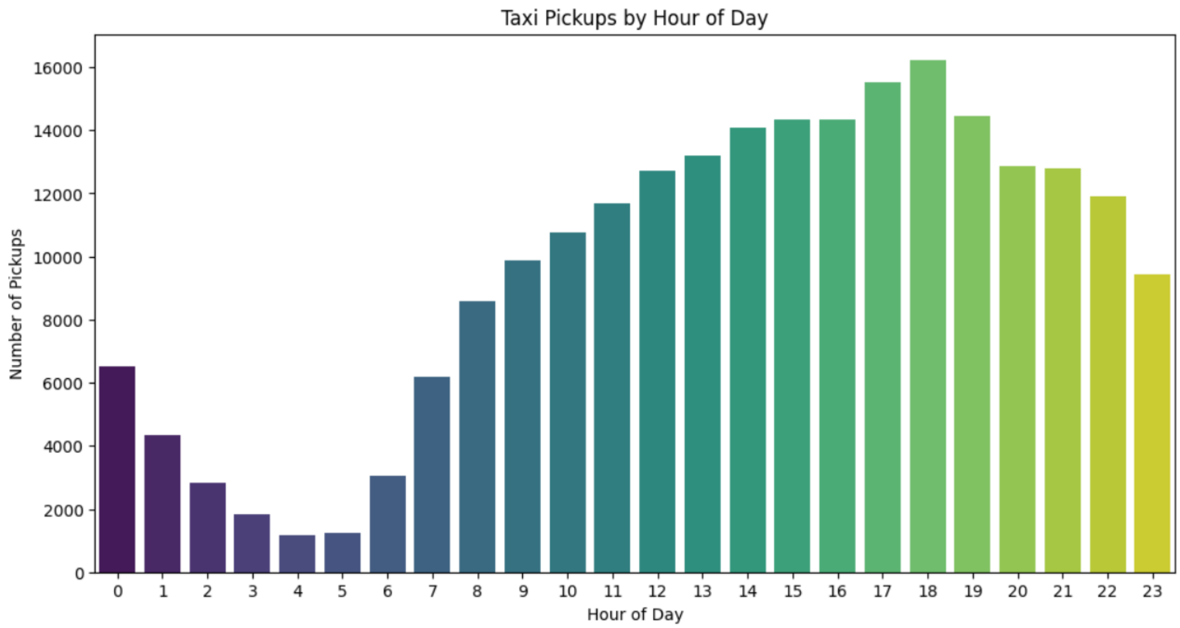
3. Exploratory Data Analysis

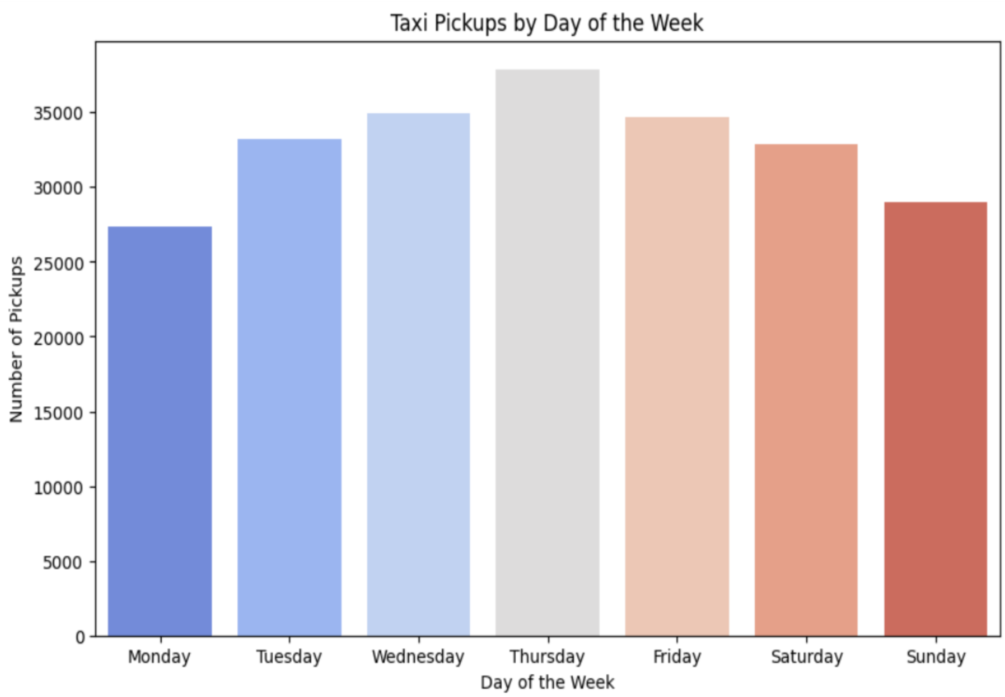
3.1. General EDA: Finding Patterns and Trends

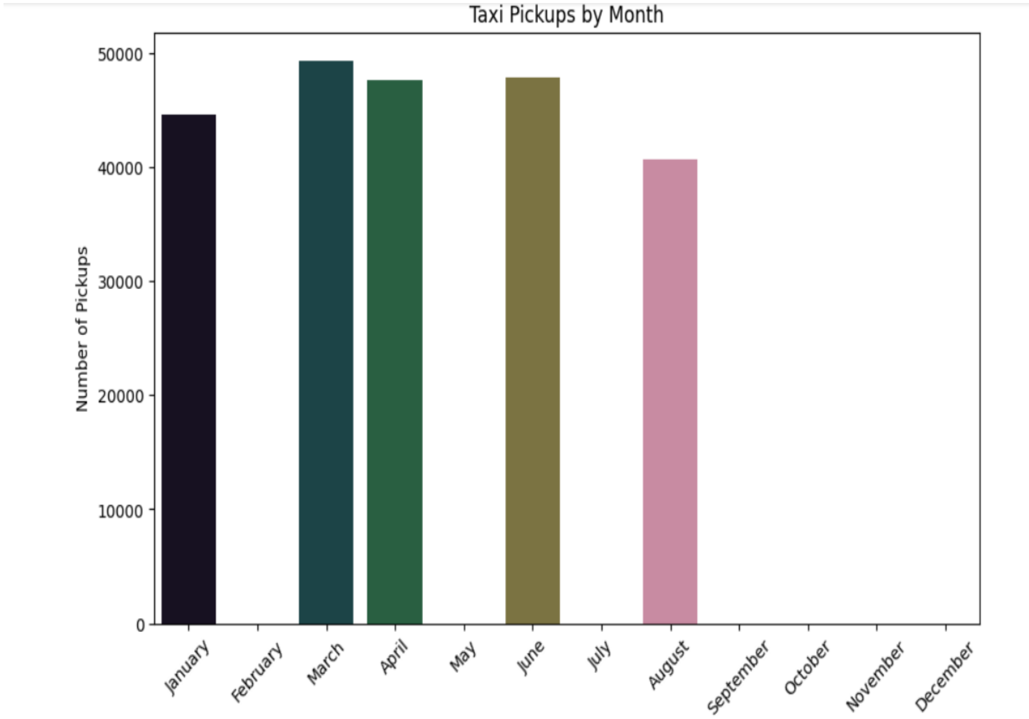
3.1.1. Classify variables into categorical and numerical

```
Categorical columns: ['store_and_fwd_flag', 'day_of_week', 'pickup_month', 'weekday_name', 'day_type', 'distance_tier', 'distance_category']
Numerical columns: ['VendorID', 'passenger_count', 'trip_distance', 'RatecodeID', 'PULocationID', 'DOLocationID', 'payment_type', 'fare_amount',
```

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


Taxi Pickups by Hour of Day
Number of Pickups
Hour of Day
0
2000
4000
6000
8000
10000
12000
14000
16000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

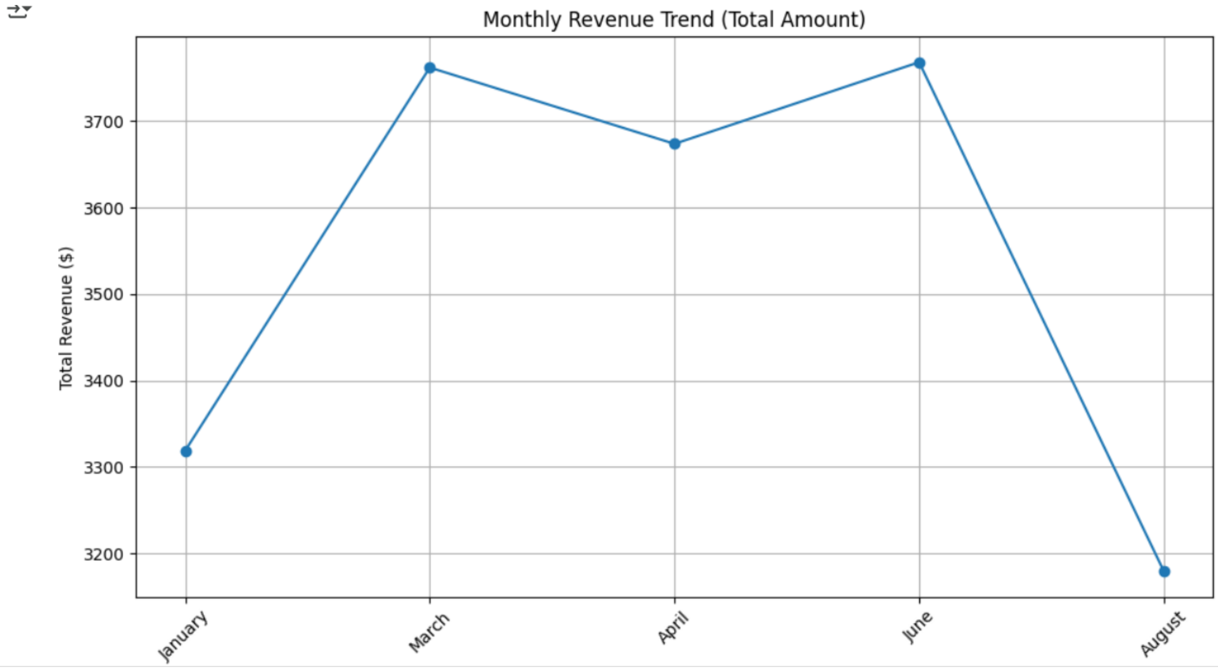

Taxi Pickups by Day of the Week
Number of Pickups
Day of the Week
0
5000
10000
15000
20000
25000
30000
35000
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday



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

```
fare_amount — Zero values: 78, Negative values: 0
tip_amount — Zero values: 52254, Negative values: 0
total_amount — Zero values: 8, Negative values: 0
trip_distance — Zero values: 2804, Negative values: 0
```

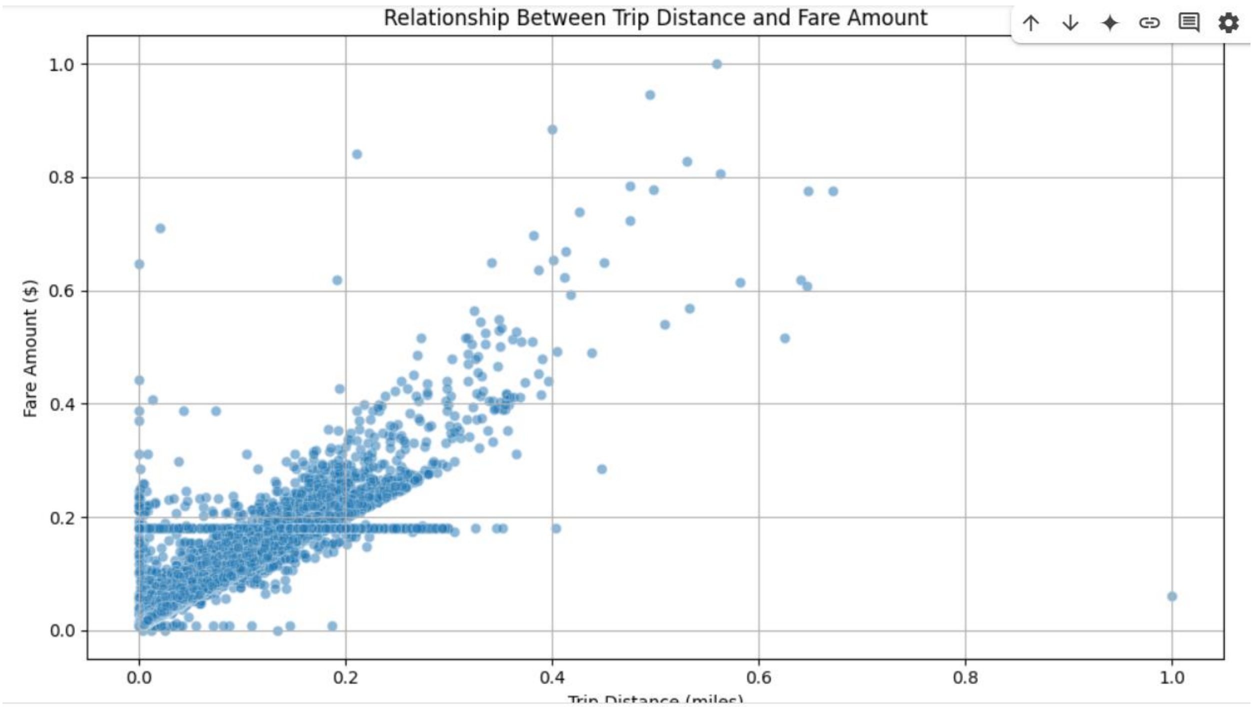
3.1.4. Analyse the monthly revenue trends



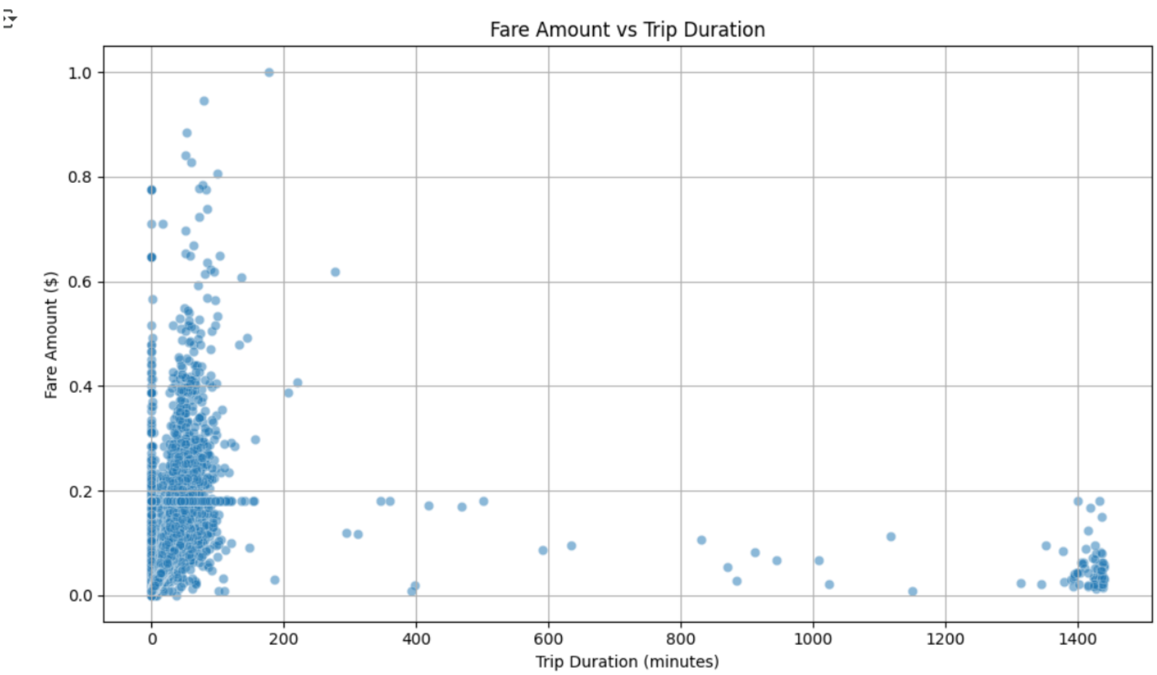
3.1.5. Find the proportion of each quarter's revenue in the yearly revenue

```
  quarter  total_amount
0      Q1   7080.556656
1      Q2   7441.700237
2      Q3   3179.390478
  quarter  total_amount  revenue_proportion
0      Q1   7080.556656            0.399994
1      Q2   7441.700237            0.420396
2      Q3   3179.390478            0.179610
```

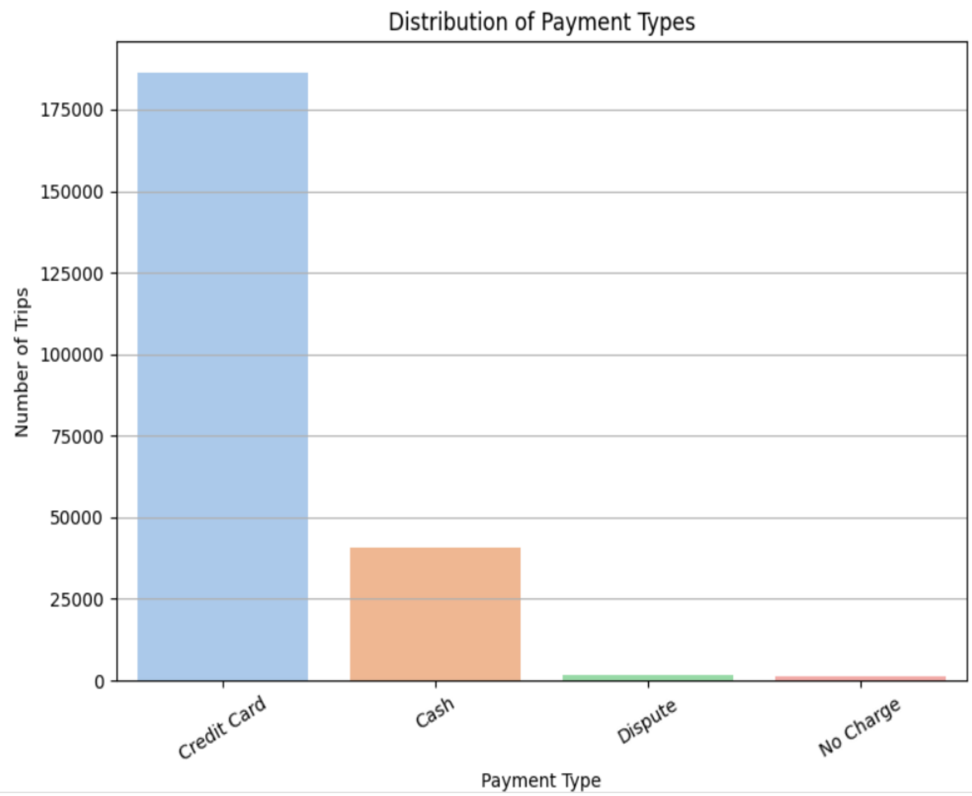
3.1.6. Analyse and visualise the relationship between distance and fare amount



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



3.1.8. Analyse the distribution of different payment types



3.1.9. Load the taxi zones shapefile and display it

	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...

3.1.10. Merge the zone data with trips data

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

```
print(trips_with_zones.head())
print(dropoff_counts.head())
```

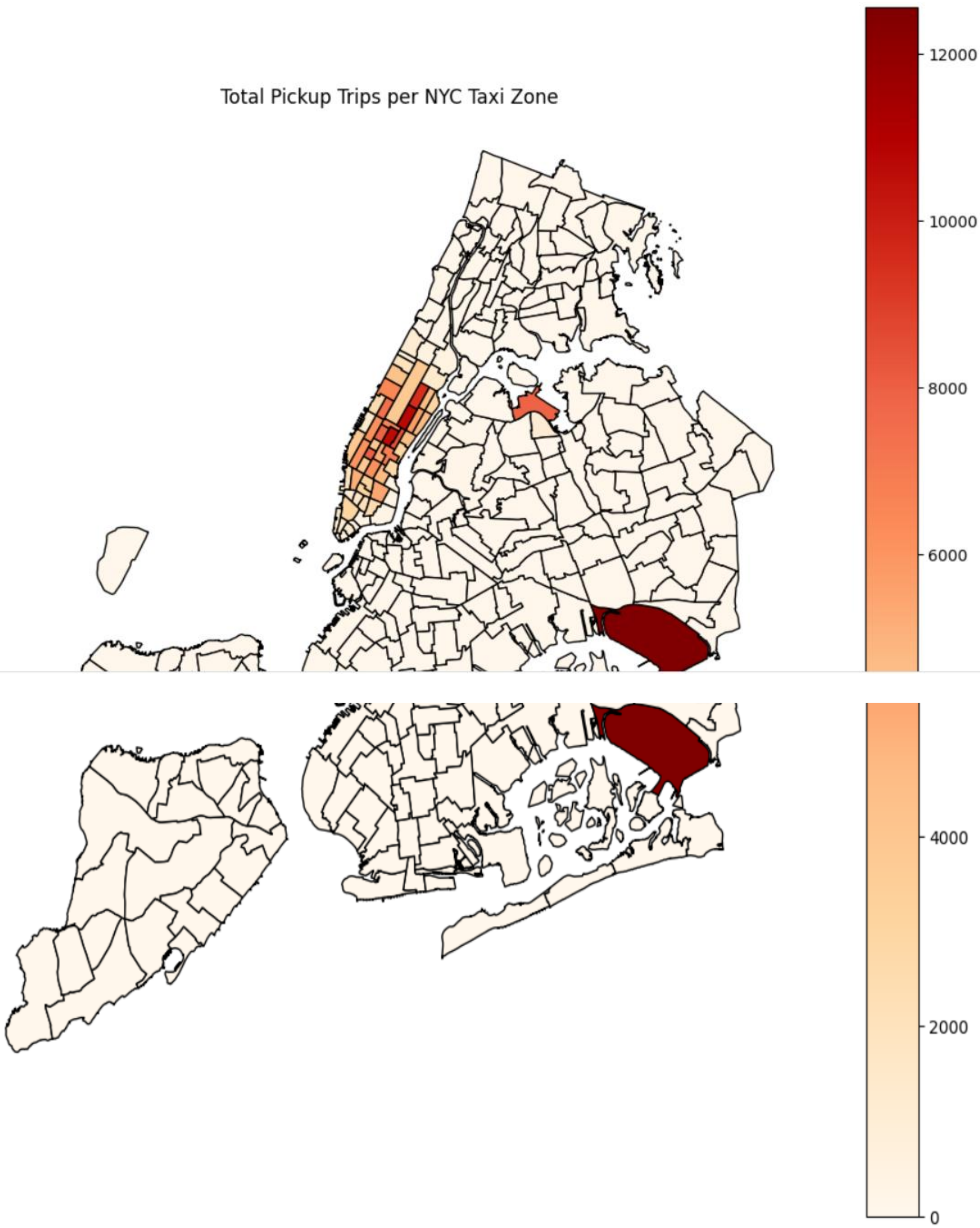
```
   VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
0         1  2023-04-01 00:58:33   2023-04-01 01:07:03              4.0
1         1  2023-04-01 00:10:28   2023-04-01 00:27:17              1.0
2         2  2023-04-01 00:54:11   2023-04-01 01:00:52              1.0
3         1  2023-04-01 00:53:11   2023-04-01 01:04:05              2.0
4         2  2023-04-01 00:38:39   2023-04-01 00:52:06              4.0

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID  DOLocationID  \
0       0.010547         1.0                  Y           249           125
1       0.028763         1.0                  N           230           114
2       0.014669         1.0                  N           100           113
3       0.014382         1.0                  N            90           164
4       0.015340         1.0                  N           211           211

   payment_type  ...  LocationID    borough  \
0             1  ...       249.0  Manhattan
1             2  ...       230.0  Manhattan
2             1  ...       100.0  Manhattan
3             1  ...        90.0  Manhattan
4             1  ...       211.0  Manhattan

                                            geometry  OBJECTID_dropoff  \
0  POLYGON ((983555.319 204876.901, 983469.158 20...             125.0
1  POLYGON ((988786.877 214532.094, 988650.277 21...             114.0
2  POLYGON ((987770.527 212686.678, 987638.873 21...             113.0
3  POLYGON ((985265.129 208165.863, 985125.733 20...             164.0
4  POLYGON ((983827.65 201526.658, 983727.737 201...             211.0
```

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



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

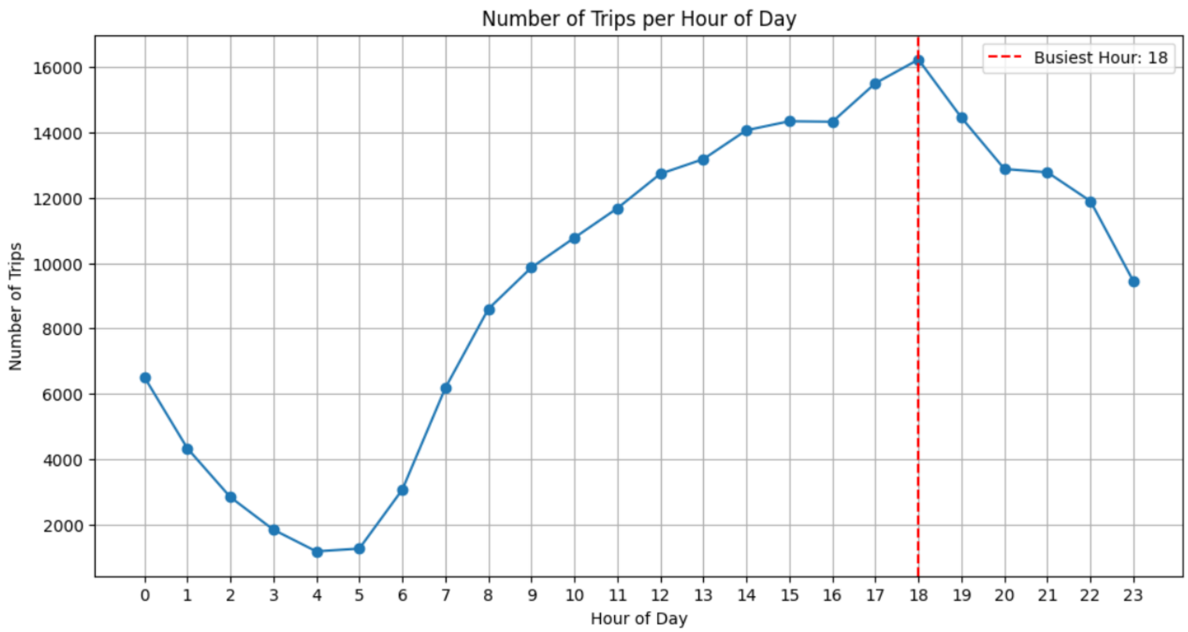
3.1.14. Conclude with results

3.2. Detailed EDA: Insights and Strategies

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

	pickup_hour	route	avg_speed_mph
0	0	231-231	0.000449
1	1	142-142	0.007266
2	2	229-137	0.000778
3	3	148-148	0.002186
4	4	230-51	0.006882
5	5	230-230	0.003500
6	6	185-168	0.007910
7	7	231-236	0.005888
8	8	162-163	0.002383
9	9	151-163	0.002199
10	10	234-231	0.000872
11	11	220-236	0.000929
12	12	239-164	0.002732
13	13	113-113	0.000157
14	14	162-238	0.005655
15	15	134-265	0.000708
16	16	41-41	0.000636
17	17	151-24	0.000872
18	18	234-256	0.001300
19	19	158-158	0.001731
20	20	161-132	0.001845
21	21	164-100	0.000650
22	22	263-75	0.001625
23	23	230-48	0.001543

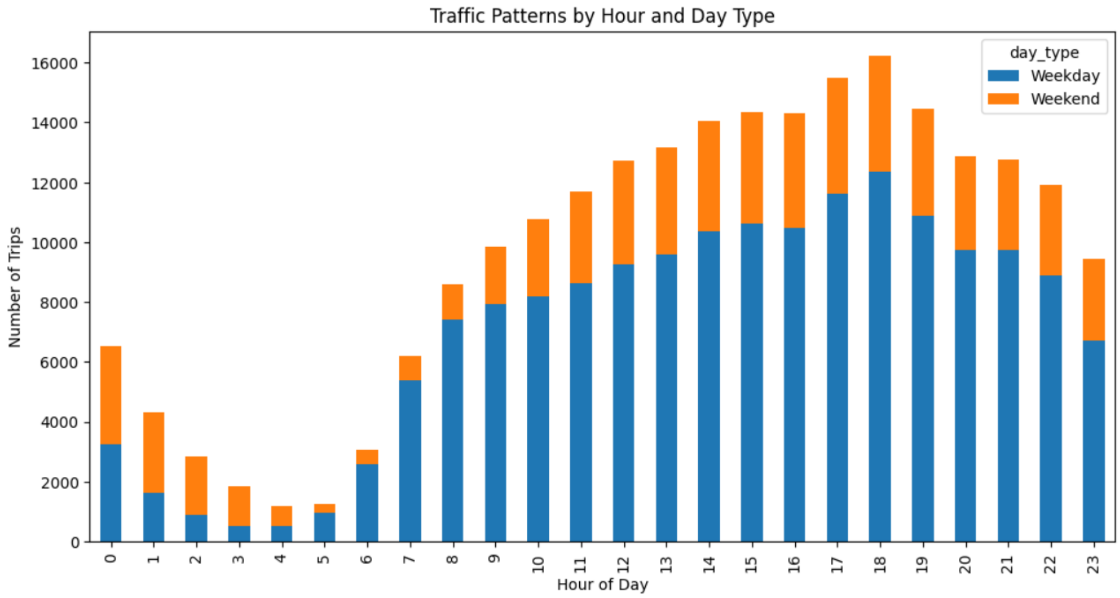
3.2.2. Calculate the hourly number of trips and identify the busy hours



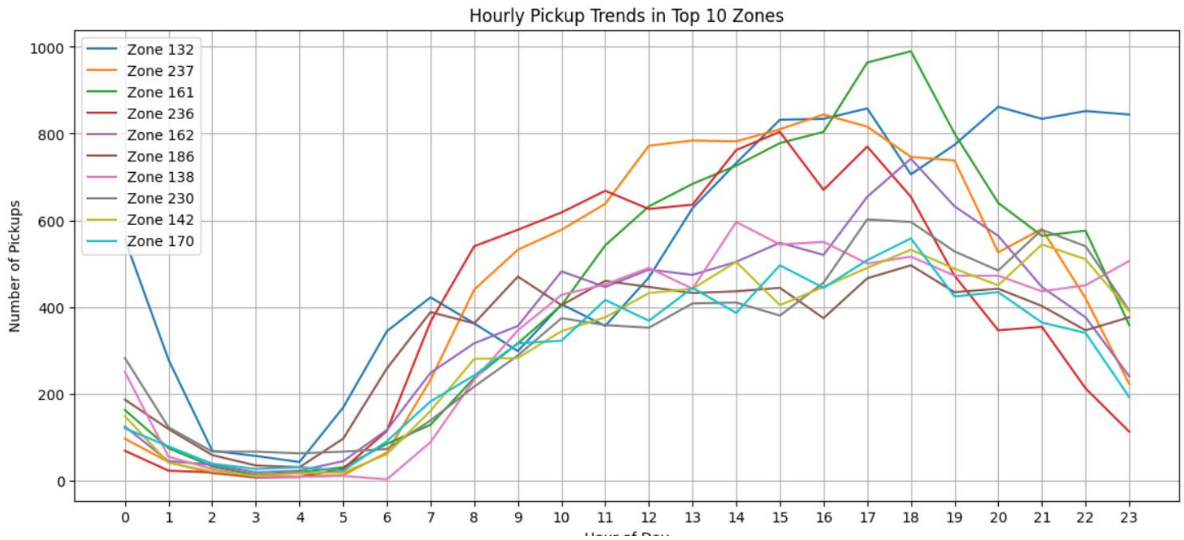
3.2.3. Scale up the number of trips from above to find the actual number of trips

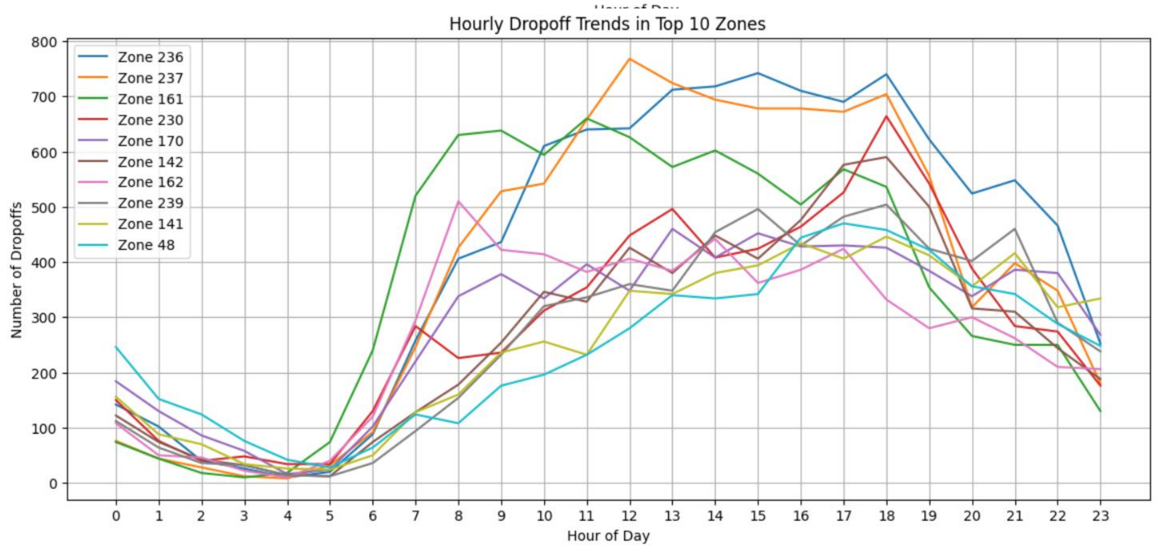
```
Estimated actual number of trips in the 5 busiest hours:
pickup_hour
18    324640
17    310160
19    289080
15    286880
16    286560
Name: count, dtype: int64
```

3.2.4. Compare hourly traffic on weekdays and weekends



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





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

Top 10 Pickup/Dropoff Ratios:

	pickup_trip_count	dropoff_trip_count	pickup_dropoff_ratio
118	7386.0	4	1846.500000
108	12562.0	8	1570.250000
144	6840.0	6	1140.000000
92	3106.0	6	517.666667
102	1020.0	2	510.000000
137	6482.0	16	405.125000
225	2372.0	6	395.333333
42	1412.0	4	353.000000
116	4672.0	22	212.363636
156	7958.0	40	198.950000

```
Bottom 10 Pickup/Dropoff Ratios:
     pickup_trip_count  dropoff_trip_count  pickup_dropoff_ratio
232                0.0                  90                   0.0
233                0.0                  40                   0.0
234                0.0                3732                   0.0
231                0.0                4134                   0.0
228                0.0                 368                   0.0
229                0.0                 702                   0.0
230                0.0                   4                   0.0
227                0.0                  32                   0.0
240                0.0                 570                   0.0
241                0.0                 540                   0.0
```

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

```
Top 10 Pickup Zones (Night Hours 11PM–5AM):
PULocationID
132    2012
79     1986
249    1632
48     1266
148    1248
114    1086
230    1056
186     898
138     862
164     724
Name: count, dtype: int64

Top 10 Dropoff Zones (Night Hours 11PM–5AM):
DOLocationID
79     1064
48      916
170     766
68      748
141     732
107     724
263     674
229     596
236     592
249     590
```

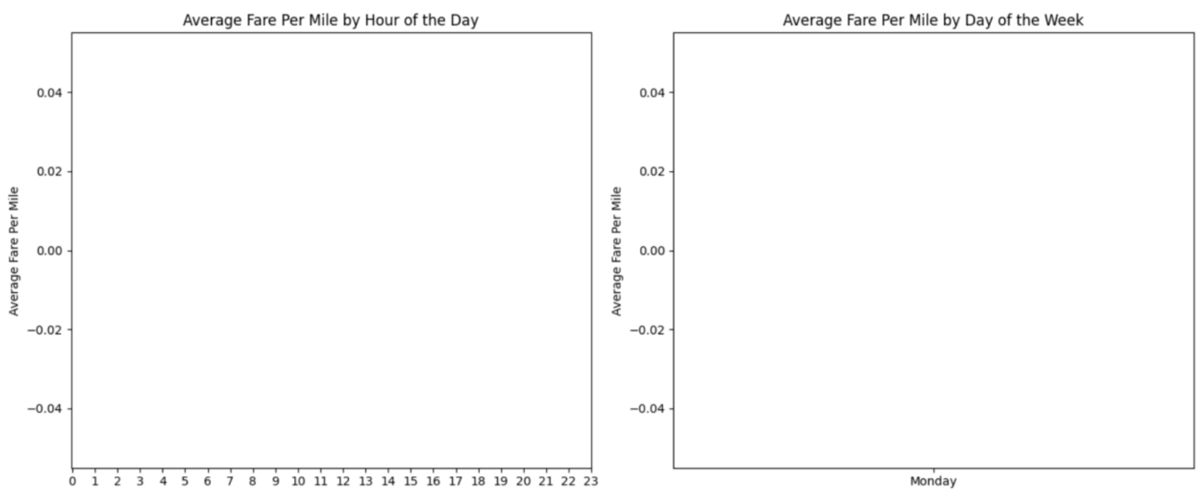
3.2.8. Find the revenue share for nighttime and daytime hours

```
Nighttime Revenue Share: 12.37%
Daytime Revenue Share: 87.63%
```

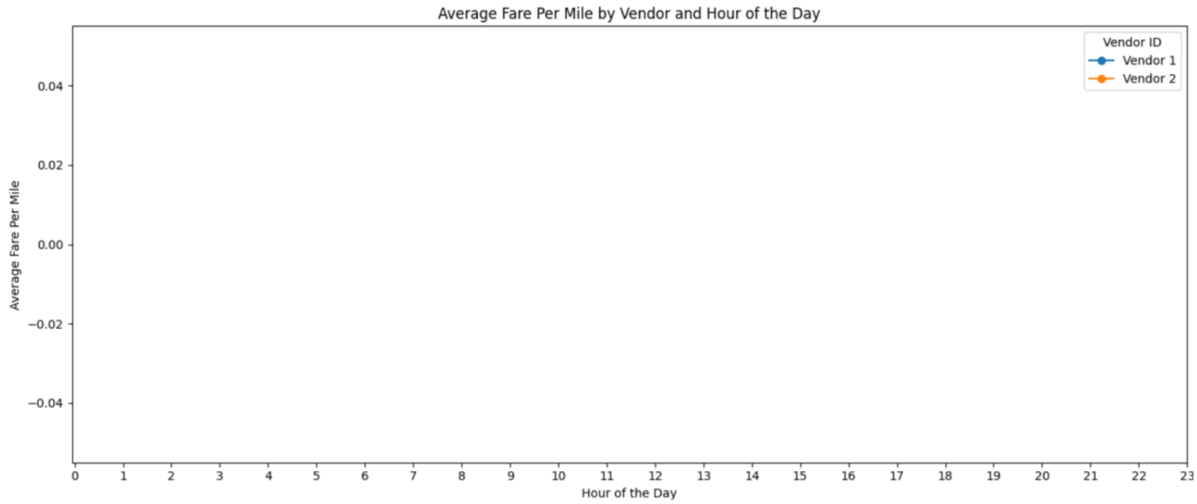
3.2.9. For the different passenger counts, find the average fare per mile per passenger

```
   passenger_count  fare_per_mile_per_passenger
0              1.0                          inf
1              2.0                          inf
2              3.0                          inf
3              4.0                          inf
4              5.0                          inf
5              6.0                          inf
6              7.0                          inf
```

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



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



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

3.2.13. Analyse the tip percentages

```
Average Tip Percentage by Distance Category:
   distance_category  tip_percentage
0  Short (<=2 miles)       28.354196
```

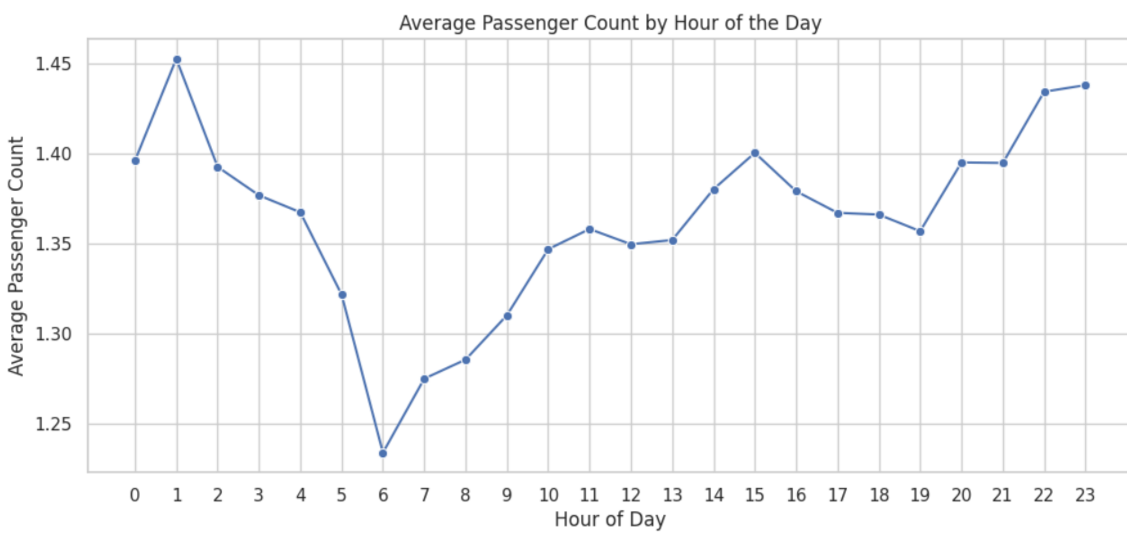
Low Tip Trips (<10%)

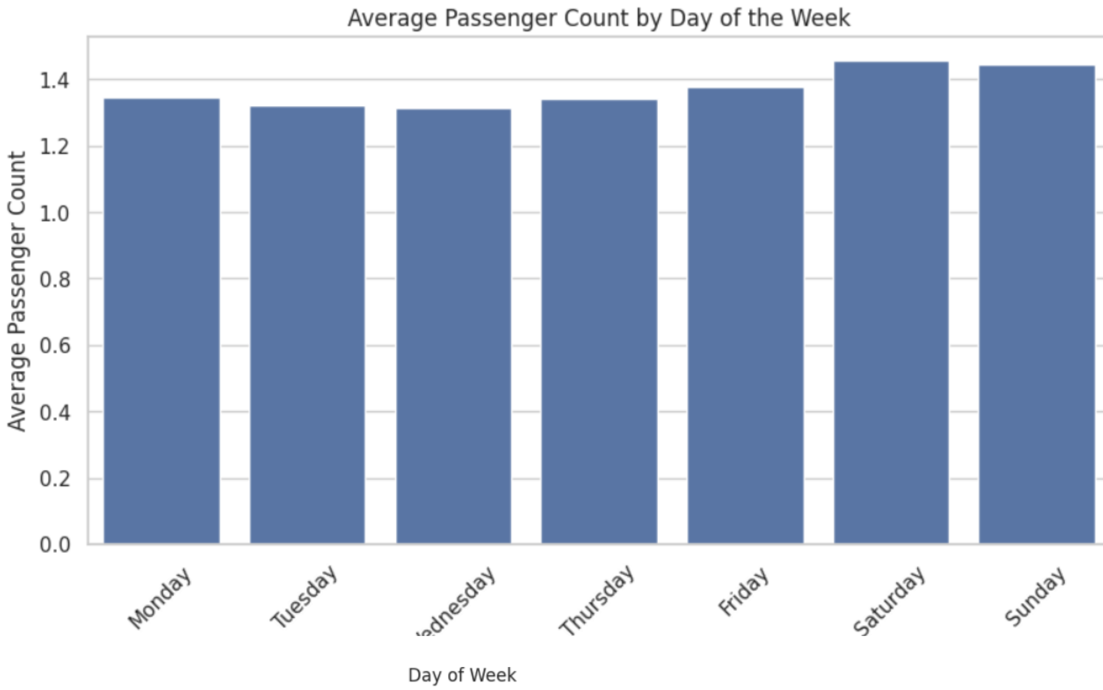
	trip_distance	total_amount	passenger_count	pickup_hour
count	54530.000000	54530.000000	54530.000000	54530.000000
mean	0.035175	0.071194	1.414597	13.883000
std	0.048021	0.052981	0.934295	5.655385
min	0.000000	0.005921	0.000000	0.000000
25%	0.009204	0.040976	1.000000	10.000000
50%	0.016395	0.052582	1.000000	14.000000
75%	0.035570	0.075793	2.000000	18.000000
max	0.672100	0.960090	7.000000	23.000000

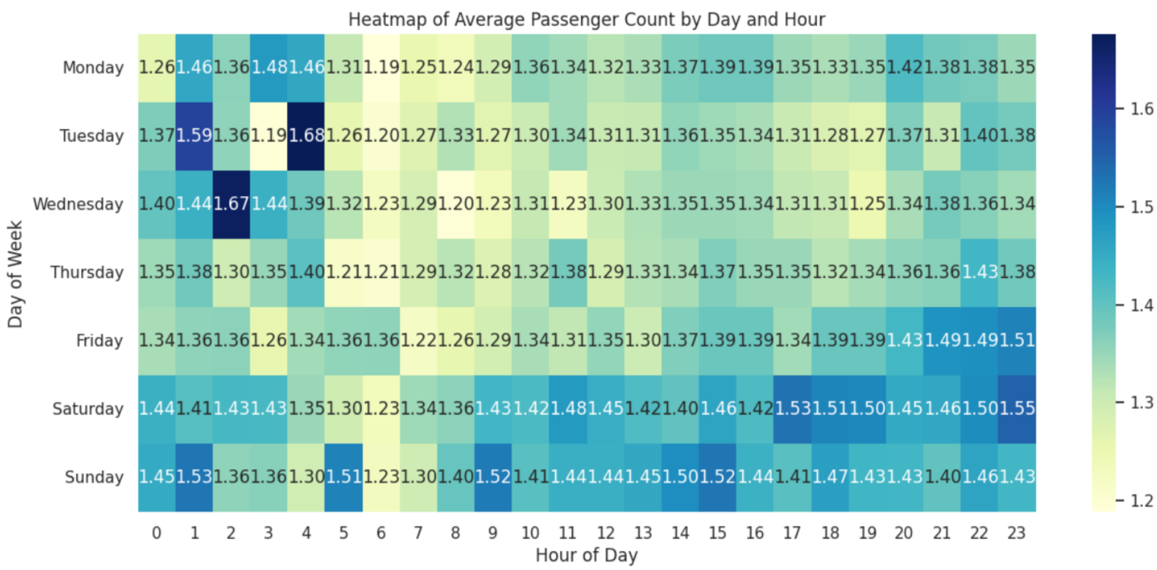
High Tip Trips (>25%)

	trip_distance	total_amount	passenger_count	pickup_hour
count	148898.000000	148898.000000	148898.000000	148898.000000
mean	0.033010	0.080204	1.358688	14.413155
std	0.042451	0.054938	0.886347	5.794978
min	0.000000	0.022264	0.000000	0.000000
25%	0.010547	0.049266	1.000000	11.000000
50%	0.017258	0.061203	1.000000	15.000000
75%	0.032119	0.083089	1.000000	19.000000
max	0.647267	1.000000	6.000000	23.000000

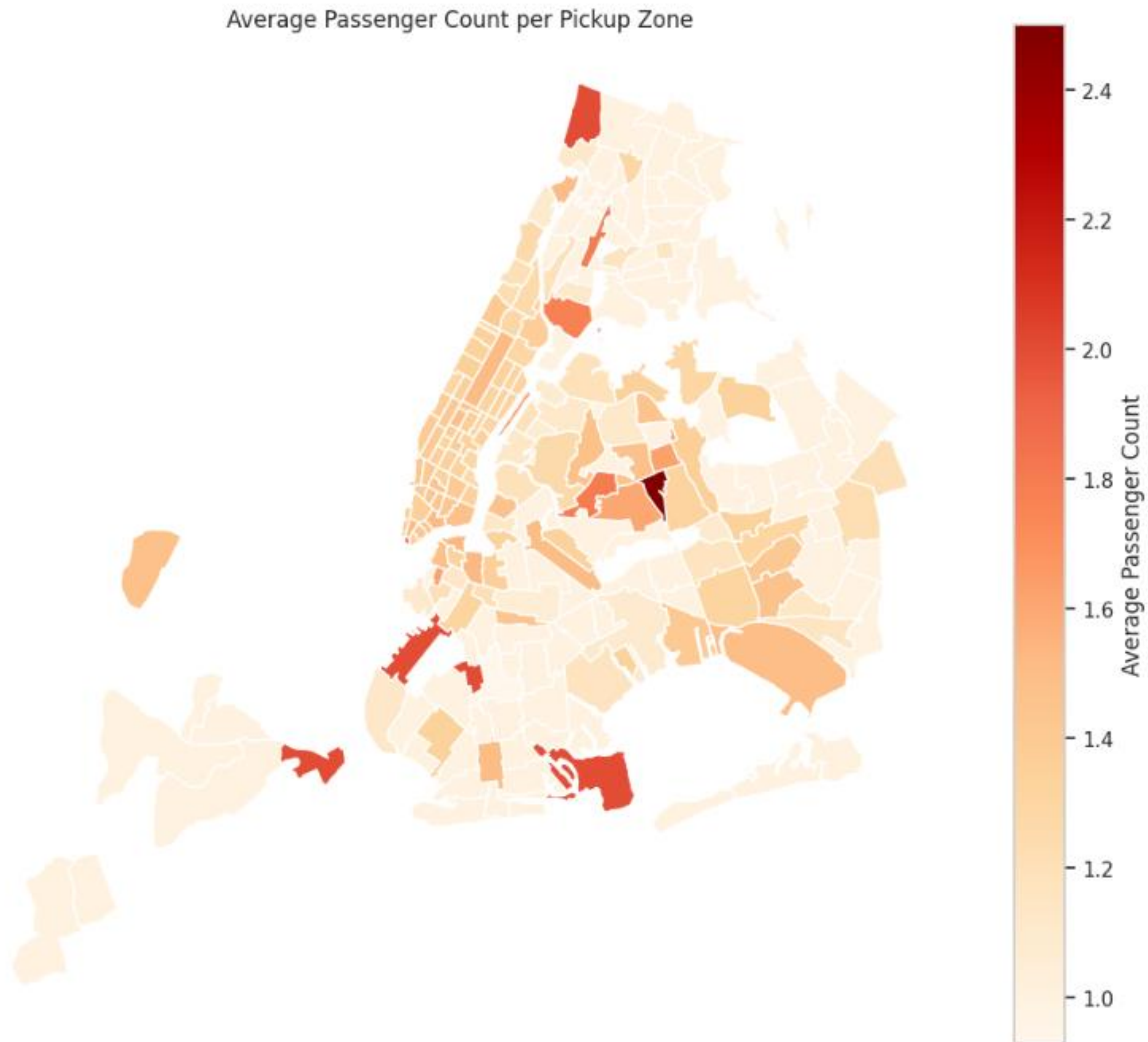
3.2.14. Analyse the trends in passenger count



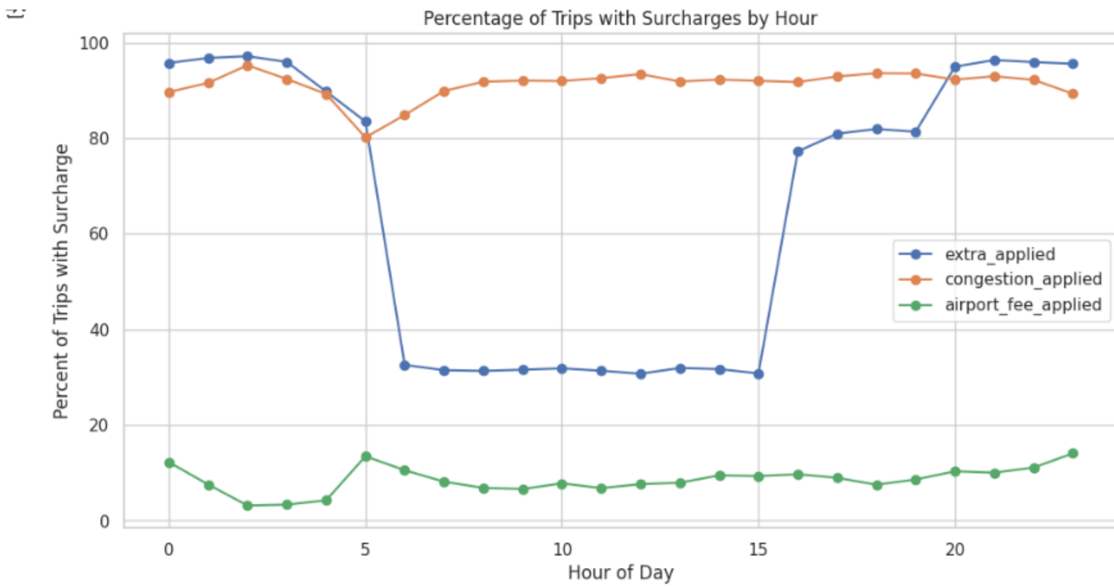




3.2.15. Analyse the variation of passenger counts across zones



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



4. Conclusions

4.1. Final Insights and Recommendations

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

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

```
Morning Rush (7–10 AM): Position cabs in residential areas like Brooklyn and Queens to capture commuter demand.

Evening Rush (4–8 PM): Shift to commercial zones such as Midtown and the Financial District.

Late Night (11 PM–3 AM): Focus on nightlife hotspots, airports, and major transit hubs.

Weekends: Reallocate to leisure and entertainment zones with late-night demand.

Monthly/Seasonal Trends: Increase coverage in tourist areas during holidays and recreational zones in summer.
Predictive Deployment: Use past data to pre-position cabs ahead of expected demand surges.
```

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

```
Use real-time data to adjust pricing dynamically.Monitor vendor-specific trends to stay within competitive benchmarks.
Test pricing changes in select zones/hours before scaling.
```