Report: Optimizing NYC Taxi Operations
Name: Ujwal Abhishek

Include your visualizations, 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

```python
# 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/UpgradCourse/data/trip_records')

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

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

# 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 month file
        month_df = pd.read_parquet(file_path)

        # We will store the sampled data for the current date in this df by appending the sampl
        # After completing iteration through each date, we will append this data to the final d
        sampled_data = pd.DataFrame()

        # Ensure pickup datetime is in datetime format
        month_df['tpep_pickup_datetime'] = pd.to_datetime(month_df['tpep_pickup_datetime'])

        # Extract date and hour
        month_df['pickup_date'] = month_df['tpep_pickup_datetime'].dt.date
        month_df['pickup_hour'] = month_df['tpep_pickup_datetime'].dt.hour
        # Loop through dates and then loop through every hour of each date
        for date in month_df['pickup_date'].unique():
            date_data = month_df[month_df['pickup_date'] == date]
            # Iterate through each hour of the selected date
            for hour in range(24):
                hour_data = date_data[date_data['pickup_hour'] == hour]
                # Sample 5% of the hourly data randomly
                if not hour_data.empty:
                    sample = hour_data.sample(frac=0.05, random_state=42)

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

        # Concatenate the sampled data of all the dates to a single dataframe
        df = pd.concat([df, sampled_data], ignore_index=True)

    except Exception as e:
        print(f"Error reading file {file_name}: {e}")
# Final shape of combined sampled data
print("Sampling complete. Final data shape:", df.shape)
```

```
Sampling complete. Final data shape: (2042850, 22)
```

2. Data Cleaning

2.1. Fixing Columns

2.1.1. Fix the index

```
# Fix the index and drop any columns that are not needed
sampleData.reset_index(drop=True, inplace=True)

unnamed_cols = [col for col in sampleData.columns if 'unnamed' in col.lower()]
sampleData.drop(columns=unnamed_cols, inplace=True)
sampleData
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	...	mta_tax	tip_amount	t
0	2	2023-01-01 00:07:18	2023-01-01 00:23:15	1.0	7.74	1.0	N	138	256	2	...	0.5	0.00	
1	2	2023-01-01 00:16:41	2023-01-01 00:21:46	2.0	1.24	1.0	N	161	237	1	...	0.5	2.58	
2	2	2023-01-01 00:14:03	2023-01-01 00:24:36	3.0	1.44	1.0	N	237	141	2	...	0.5	0.00	
3	2	2023-01-01 00:24:30	2023-01-01 00:29:55	1.0	0.54	1.0	N	143	142	2	...	0.5	0.00	
4	2	2023-01-01 00:43:00	2023-01-01 01:01:00	NaN	19.24	NaN	NaN	66	107	0	...	0.5	5.93	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2042845	2	2023-09-30 23:46:34	2023-09-30 23:53:20	1.0	0.79	1.0	N	231	231	1	...	0.5	2.00	
2042846	1	2023-09-30 23:44:51	2023-09-30 23:49:05	3.0	0.50	1.0	N	158	68	1	...	0.5	2.15	
2042847	2	2023-09-30 23:11:05	2023-09-30 23:18:42	1.0	1.09	1.0	N	161	162	1	...	0.5	2.86	
2042848	1	2023-09-30 23:26:31	2023-10-01 00:04:05	2.0	13.20	1.0	N	164	14	2	...	0.5	0.00	
2042849	2	2023-09-30 23:19:47	2023-09-30 23:33:36	1.0	2.97	1.0	N	231	68	1	...	0.5	4.40	

2042850 rows × 22 columns

2.1.2. Combine the two airport_fee columns

```
# Combine the two airport fee columns
# Merge the two airport fee columns by preferring non-null values
sampleData['airport_fee'] = sampleData['airport_fee'].combine_first(sampleData['Airport_fee'])

# Drop the redundant column
sampleData.drop(columns=['Airport_fee'], inplace=True)

sampleData
```

d_fwd_flag	PULocationID	DOLocationID	payment_type	...	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount	congestion_surcharge	airport_fee	pickup_date	pickup_hour
N	138	256	2	...	6.0	0.5	0.00	0.0	1.0	41.15	0.0	1.25	2023-01-01	0
N	161	237	1	...	1.0	0.5	2.58	0.0	1.0	15.48	2.5	0.00	2023-01-01	0
N	237	141	2	...	1.0	0.5	0.00	0.0	1.0	16.40	2.5	0.00	2023-01-01	0
N	143	142	2	...	1.0	0.5	0.00	0.0	1.0	11.50	2.5	0.00	2023-01-01	0
NaN	66	107	0	...	0.0	0.5	5.93	0.0	1.0	35.57	NaN	NaN	2023-01-01	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
N	231	231	1	...	1.0	0.5	2.00	0.0	1.0	15.60	2.5	0.00	2023-09-30	23
N	158	68	1	...	3.5	0.5	2.15	0.0	1.0	12.95	2.5	0.00	2023-09-30	23
N	161	162	1	...	1.0	0.5	2.86	0.0	1.0	17.16	2.5	0.00	2023-09-30	23
N	164	14	2	...	3.5	0.5	0.00	0.0	1.0	59.80	2.5	0.00	2023-09-30	23
N	231	68	1	...	1.0	0.5	4.40	0.0	1.0	26.40	2.5	0.00	2023-09-30	23

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

# Calculate proportion of missing (NaN) values in each column
missing_proportion = sampleData.isnull().mean().sort_values(ascending=False)
# Display result as percentages
missing_proportion_percent = (missing_proportion * 100).round(2)
print(missing_proportion_percent)
```

```
passenger_count          3.44
airport_fee              3.44
congestion_surcharge     3.44
store_and_fwd_flag       3.44
RatecodeID               3.44
trip_distance            0.00
tpep_dropoff_datetime    0.00
tpep_pickup_datetime     0.00
VendorID                 0.00
payment_type             0.00
fare_amount              0.00
PULocationID             0.00
DOLocationID             0.00
mta_tax                  0.00
extra                    0.00
tip_amount               0.00
tolls_amount             0.00
total_amount             0.00
improvement_surcharge    0.00
pickup_date              0.00
pickup_hour              0.00
dtype: float64
```

2.2.2. Handling missing values in passenger_count

Handling missing values in `passenger_count`

```
# Display the rows with null values
# Impute NaN values in 'passenger_count'

# Display no of rows where 'passenger_count' is null
null_passengers = sampleData['passenger_count'].isnull().sum()
print(f"Records with passenger_count zero: {null_passengers}")
```

```
Records with passenger_count zero: 70245
```

```
# Compute mode (most common passenger count)
most_common = sampleData['passenger_count'].mode()[0]
most_common
```

```
np.float64(1.0)
```

```
# Fill missing values
sampleData['passenger_count'].fillna(most_common, inplace=True)
print("Missing passenger_count after imputation:", sampleData['passenger_count'].isnull().sum())
```

```
Missing passenger_count after imputation: 0
```

```
# Count rows where passenger_count is 0
zero_passengers = sampleData[sampleData['passenger_count'] == 0]
print(f"Records with passenger_count zero: {len(zero_passengers)}")
```

```
Records with passenger_count zero: 31420
```

2.2.3.

Handle missing values in `RatecodeID`

```
# Fix missing values in 'RatecodeID'
```

```
#Check how many are missing
missing_count = sampleData['RatecodeID'].isnull().sum()
print(f"Missing RatecodeID count: {missing_count}")
```

```
Missing RatecodeID count: 70245
```

```
# Find most frequent RatecodeID
most_common_ratecode = sampleData['RatecodeID'].mode()[0]

# Fill missing values
sampleData['RatecodeID'].fillna(most_common_ratecode, inplace=True)
#Check how many are missing
missing_count = sampleData['RatecodeID'].isnull().sum()
print(f"Missing RatecodeID count after imputation: {missing_count}")
```

```
Missing RatecodeID count after imputation: 0
```

2.2.4. Impute NaN in congestion_surcharge

Impute NaN in `congestion_surcharge`

```
# handle null values in congestion_surcharge
#Check how many are missing
congestion_surcharge_missing_count = sampleData['congestion_surcharge'].isnull().sum()
print(f"Missing RatecodeID count: {congestion_surcharge_missing_count}")
```

```
Missing RatecodeID count: 70245
```

```
# Find most frequent RatecodeID
most_common_congestion_surcharge = sampleData['congestion_surcharge'].mode()[0]

# Fill missing values
sampleData['congestion_surcharge'].fillna(most_common_congestion_surcharge, inplace=True)
#Check how many are missing
congestion_surcharge_missing_count = sampleData['congestion_surcharge'].isnull().sum()
print(f"Missing RatecodeID count after imputation: {congestion_surcharge_missing_count}")
```

```
Missing RatecodeID count after imputation: 0
```

2.3. Handling Outliers and Standardising Values

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

Entries where payment_type is 0

```
#Entries where payment_type is 0 (there is no payment_type 0 defined in the data dictionary)
# Find entries with invalid payment_type = 0
invalid_payment_type = sampleData[sampleData['payment_type'] == 0]
print(f"Number of entries with payment_type = 0 after cleanup: {len(invalid_payment_type)}")
```

```
Number of entries with payment_type = 0 after cleanup: 70215
```

```
# Remove Entries where payment_type is 0
sampleData = sampleData[sampleData['payment_type'] != 0]

invalid_payment_type = sampleData[sampleData['payment_type'] == 0]
print(f"Number of entries with payment_type = 0 after cleanup: {len(invalid_payment_type)}")
```

```
Number of entries with payment_type = 0 after cleanup: 0
```

RateCodeID col for values not in 1 - 6

```
#RatecodeID is a catagorical data as per data dictonary values can be one of 1= Standard rate,2=JFK,3=Newark,4=Nassau or Westchester,5=Negot

#checking RateCodeID col for values not in 1 - 6
sampleData.query("RatecodeID < 1 or RatecodeID > 6")
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocat
372	1	2023-01-01 01:51:10	2023-01-01 02:19:45	1.0	0.0	99.0	N	74	
669	1	2023-01-01 02:30:32	2023-01-01 03:05:47	1.0	10.1	99.0	N	28	
1441	1	2023-01-01 10:11:12	2023-01-01 10:29:27	1.0	0.0	99.0	N	35	
1558	1	2023-01-01 11:39:40	2023-01-01 11:55:59	1.0	2.9	99.0	N	42	
1823	1	2023-01-01 12:55:17	2023-01-01 13:17:08	1.0	2.6	99.0	N	41	
...	...	...	...	...	...	...	...	...	
2039116	1	2023-09-30 12:21:00	2023-09-30 13:01:31	1.0	0.9	99.0	N	37	
2039324	1	2023-09-30 13:55:58	2023-09-30 14:20:03	1.0	1.9	99.0	N	232	
2039469	1	2023-09-30 13:52:38	2023-09-30 14:14:49	1.0	2.8	99.0	N	127	
2039483	1	2023-09-30 13:10:13	2023-09-30 14:09:53	1.0	16.8	99.0	N	205	
2040148	1	2023-09-30 16:27:07	2023-09-30 17:09:46	1.0	8.5	99.0	N	62	

10649 rows × 21 columns

```
#before deciding whether to drop the above rows or impute the values for RatecodeID for incorrect values lets check out of 10649 records
#how many records are there where trip distance is 0
sampleData.query("(RatecodeID < 1 or RatecodeID > 6) and trip_distance <=0").shape[0]
```

```
1182
```

```
#since we observe only 1182 rows out of 10649 where trip_distance <=0 and larger set of records have trip_distance > 0
# so we drop 1182 rows where  trip_distance <=0
sampleData = sampleData.drop(index = sampleData.query("(RatecodeID < 1 or RatecodeID > 6) and trip_distance <=0").index)
```

```
# and impute the RatecodeID column for these records with most common values
# Find most frequent RatecodeID
most_common_ratecode = sampleData['RatecodeID'].mode()[0]
sampleData.loc[sampleData.query("RatecodeID < 1 or RatecodeID > 6").index, 'RatecodeID'] = 1
```

Remove passenger_count > 6

```
# remove passenger_count > 6
sampleData = sampleData[sampleData['passenger_count'] <= 6]
print("Max passenger_count after cleanup:", sampleData['passenger_count'].max())
```

```
Max passenger_count after cleanup: 6.0
```

Entries where trip_distance is nearly 0 and fare_amount is more than 300

```
# Continue with outlier handling

suspicious_entries = sampleData[(sampleData['trip_distance'] <= 0) & (sampleData['fare_amount'] > 300)].shape[0]

# Display  entries
print(f"No of rows with entries where trip_distance is nearly 0 and fare_amount is more than 300: {suspicious_entries}")
```

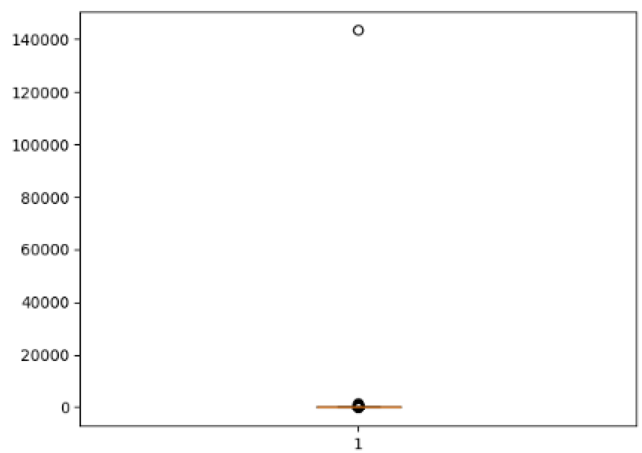
```
No of rows with entries where trip_distance is nearly 0 and fare_amount is more than 300: 32
```

```
# Dropping entries where trip_distance is nearly 0 and fare_amount is more than 300
sampleData = sampleData[~((sampleData['trip_distance'] < 0) & (sampleData['fare_amount'] > 300))]
# Verify that no such rows remain
remaining = sampleData[(sampleData['trip_distance'] < 0) & (sampleData['fare_amount'] > 300)]
print(f"Remaining suspicious rows after cleanup: {len(remaining)}")  # Should be 0
```

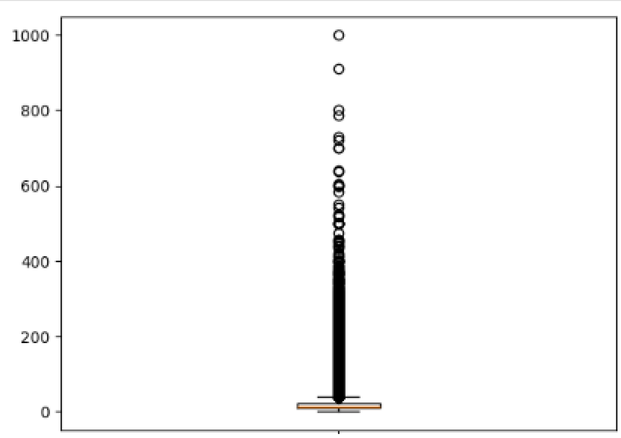
```
Remaining suspicious rows after cleanup: 0
```

checking outliers in fare_amount with box plot

```
plt.boxplot(sampleData.fare_amount)
plt.show()
```




```
#cleaning out outliers
sampleData = sampleData[~(sampleData['fare_amount'] > 1000)]
plt.boxplot(sampleData.fare_amount)
plt.show()
```



3. Exploratory Data Analysis

3.1. General EDA: Finding Patterns and Trends

3.1.1. Classify variables into categorical and numerical

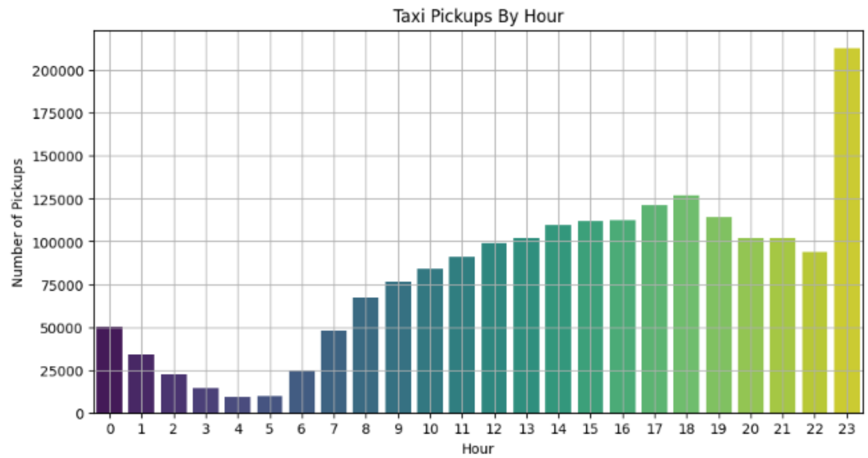
Categorise the varaibles into Numerical or Categorical.

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

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

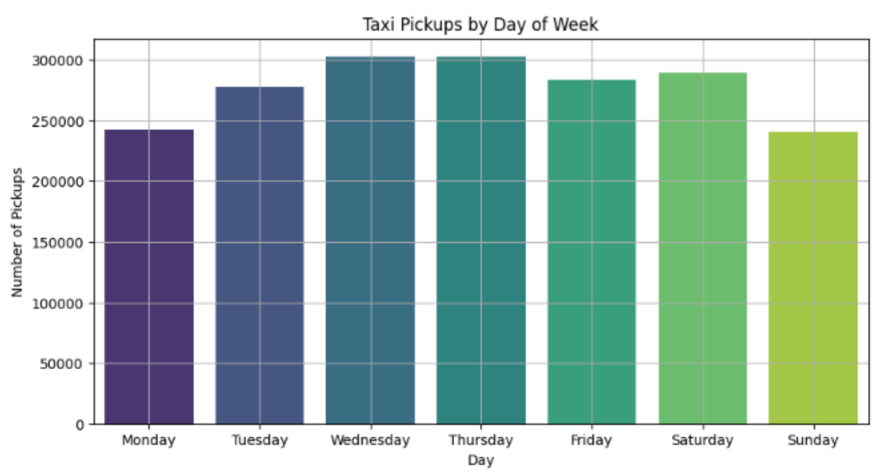
- `fare_amount` :Numerical
- `extra` :Numerical
- `mta_tax` :Numerical
- `tip_amount` :Numerical
- `tolls_amount` :Numerical
- `improvement_surcharge` :Numerical
- `total_amount` :Numerical
- `congestion_surcharge` :Numerical
- `airport_fee` :Numerical

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



Taxi Pickups by Hour of the Day

- Peak Hour: 23:00 (11 PM) — highest number of pickups (200,000+).
- High Activity Hours: 12 PM to 8 PM, showing consistently high demand.
- Low Activity Hours: 2 AM to 5 AM — pickups are minimal, reflecting off-peak hours.
- Trend: A U-shaped curve:
    - Starts low after midnight,
    - Dips during early morning,
    - Rises steadily from 7 AM,
    - Peaks in the evening and late night.



Taxi Pickups by Day of the Week

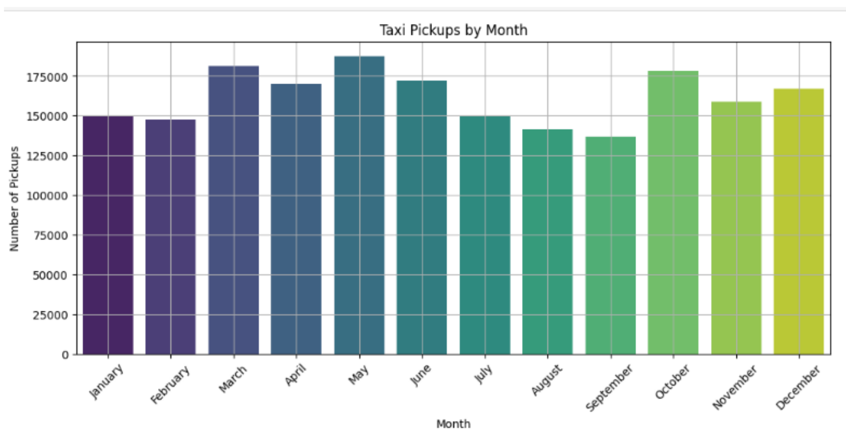
- **Highest Pickups:** Wednesday & Thursday (Over 300,000 pickups).
- **Moderately High:** Tuesday, Friday, Saturday.
- **Lowest Days:**

- Monday: 240,000 pickups.
- Sunday: lowest overall (240,000).

Interpretation:

- Weekdays (especially midweek) show higher commuting and transit demand.
- Sunday is the least busy, possibly due to reduced work-related travel.

Action Point: Adjust fleet strength lower on Sundays, higher midweek.



Taxi Pickups by Month

- Highest Pickup Months:
  - May and March
  - October also shows a late-year rise.
- Lowest Pickup Months:
  - August and September
  - Likely due to vacation season or weather disruptions.
- Seasonal Trend:
  - Spring (March–May) and late autumn (October–December) show strong demand.
  - Summer and monsoon months (June–September) have lower activity.

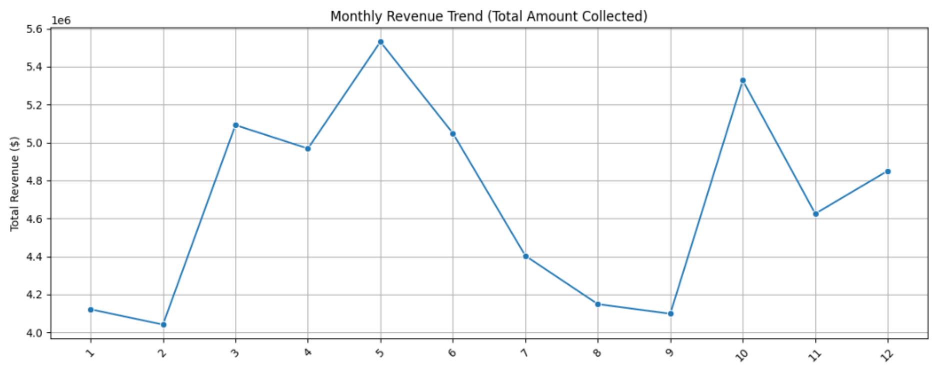
Recommendation: Target promotions and increase fleet availability in high-demand months.

3.1.3.

```
2]: # Analyse the above parameters
columns_to_check = ['fare_amount', 'tip_amount', 'total_amount', 'trip_distance']

for col in columns_to_check:
    zero_count = (sampleData[col] == 0).sum()
    negative_count = (sampleData[col] < 0).sum()
    print(f"{col} [ Zero values: {zero_count}, Negative values: {negative_count}]")
```

```
fare_amount [ Zero values: 567, Negative values: 0]
tip_amount [ Zero values: 429894, Negative values: 0]
total_amount [ Zero values: 321, Negative values: 0]
trip_distance [ Zero values: 22592, Negative values: 0]
```

3.1.4. Analyse the monthly revenue trends



Revenue Peaks

- May (Month 5): Highest revenue
- October (Month 10): Second-highest revenue
- March (Month 3): Notable peak

These peaks **correlate with the months of high taxi pickups** (as seen earlier), indicating more trips lead to higher revenue.

Revenue Dips

- February (Month 2): Lowest revenue
- August & September (Months 8 & 9): Consistently low

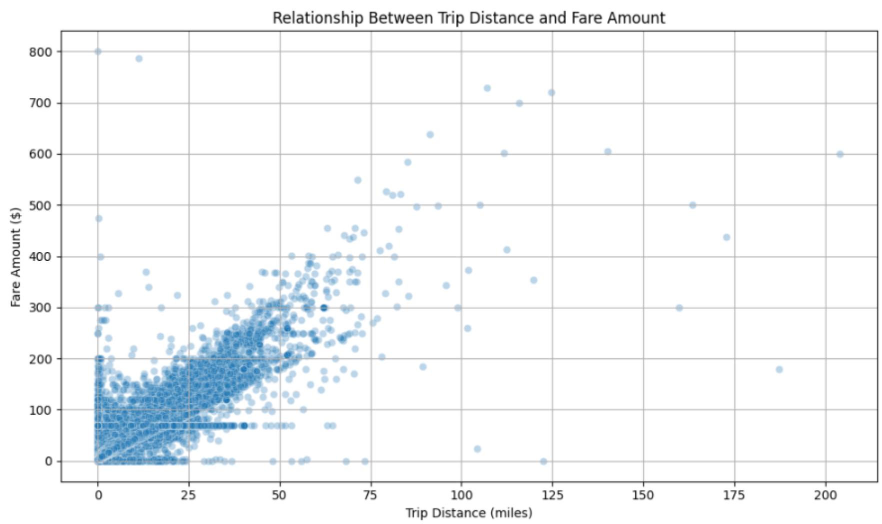
Likely causes:

- February has fewer days.
- August–September may coincide with:
  - Monsoon season in many regions (affecting travel),
  - Holiday breaks (fewer commutes).

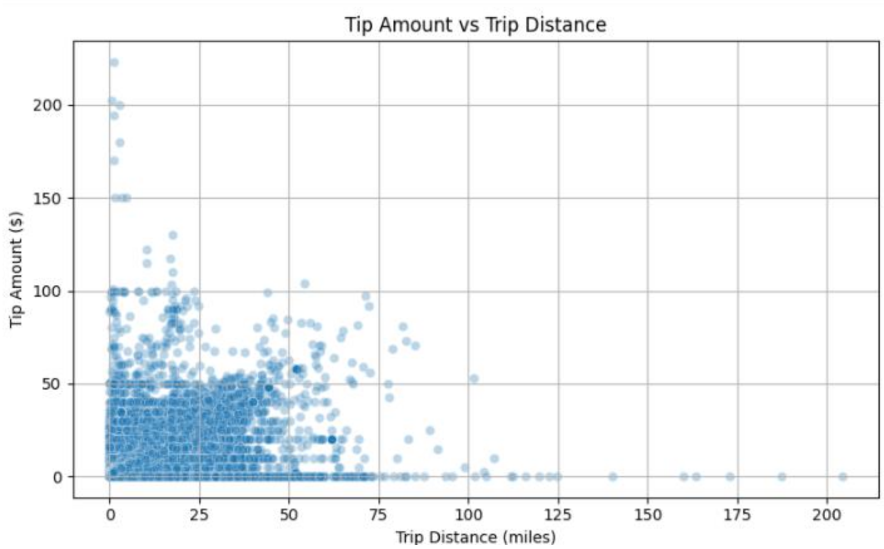
3.1.5.

	pickup_quarter	total_amount	proportion	proportion_percent
0	2022Q4	243.00	0.000004	0.00
1	2023Q1	13255154.41	0.235616	23.56
2	2023Q2	15547488.43	0.276363	27.64
3	2023Q3	12651443.21	0.224885	22.49
4	2023Q4	14803100.16	0.263131	26.31

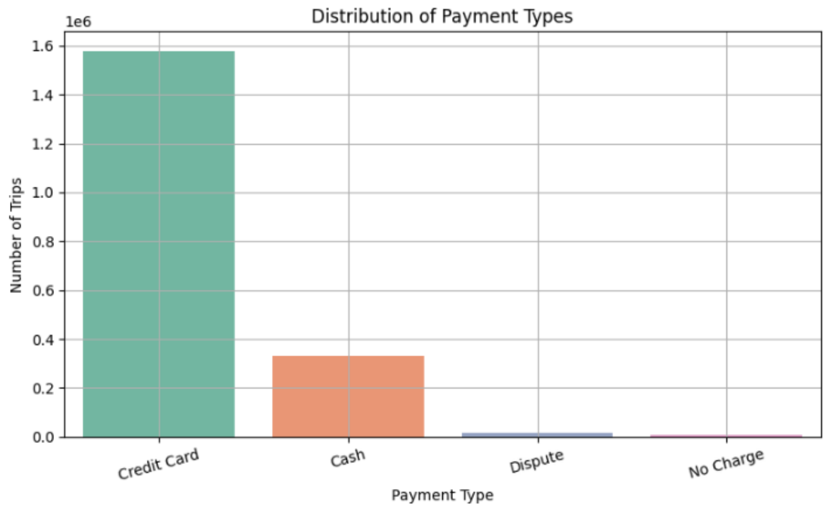
3.1.6. Analyse and visualise the relationship between distance and fare amount



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

```
# Read as a GeoDataFrame
zones_gdf = gpd.read_file(shapefile_path)

# Preview the shapefile
zones_gdf.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...

Merge the zone data with trips data

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

# get pickup zone name
sampleData = sampleData.merge(
    zones_gdf[['LocationID', 'zone']],
    how='left',
    left_on='PULocationID',
    right_on='LocationID'
).rename(columns={'zone': 'pickup_zone'})

#Merge for Dropoff Zones
sampleData = sampleData.merge(
    zones_gdf[['LocationID', 'zone']],
    how='left',
    left_on='DOLocationID',
    right_on='LocationID'
).rename(columns={'zone': 'dropoff_zone'})

# Clean up
sampleData = sampleData.drop(columns=['LocationID'], errors='ignore')
sampleData['pickup_zone'] = sampleData['pickup_zone'].astype("string")
sampleData['dropoff_zone'] = sampleData['dropoff_zone'].astype("string")
sampleData.head()
```

3.1.10. Find the number of trips for each zone/LocationID

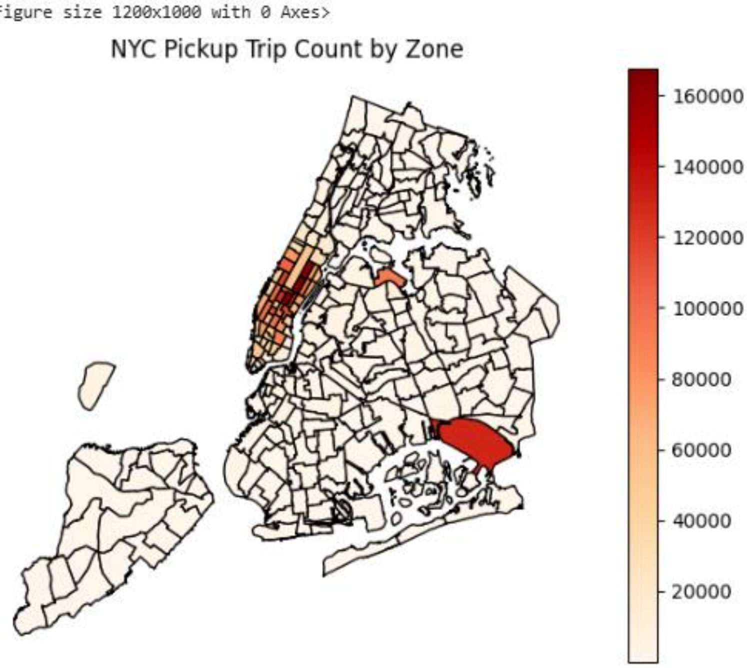
```
     LocationID                       zone  pickup_count  dropoff_count  \
233         237      Upper East Side South       88609.0        78698.0
157         161             Midtown Center       89630.0        72652.0
232         236      Upper East Side North       77898.0        83165.0
128         132                JFK Airport      108315.0        21137.0
226         230    Times Sq/Theatre District       66074.0        58435.0
158         162               Midtown East       67604.0        53663.0
138         142        Lincoln Square East       65447.0        53315.0
166         170                Murray Hill       56561.0        57134.0
182         186  Penn Station/Madison Sq West       67211.0        41237.0
235         239      Upper West Side South       51561.0        53598.0

     total_trips
233     167307.0
157     162282.0
232     161063.0
128     129452.0
226     124509.0
158     121267.0
138     118762.0
166     113695.0
182     108448.0
235     105159.0
```

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

	OBJECTID	Shape_Leng	Shape_Area	zone_x	LocationID	borough	geometry	pickup_count	dropoff_count	total_trips	zone_y
0	1	0.116357	0.000782	Newark Airport	1	EWR	POLYGON ((933100.918 192536.086, 933091.011 19...	217.0	5296.0	5513.0	Newark Airport
1	2	0.433470	0.004866	Jamaica Bay	2	Queens	MULTIPOLYGON (((1033269.244 172126.008, 103343...	2.0	3.0	5.0	Jamaica Bay
2	3	0.084341	0.000314	Allerton/Pelham Gardens	3	Bronx	POLYGON ((1026308.77 256767.698, 1026495.593 2...	34.0	189.0	223.0	Allerton/Pelham Gardens
3	4	0.043567	0.000112	Alphabet City	4	Manhattan	POLYGON ((992073.467 203714.076, 992068.667 20...	2067.0	7763.0	9830.0	Alphabet City
4	5	0.092146	0.000498	Arden Heights	5	Staten Island	POLYGON ((935843.31 144283.336, 936046.565 144...	9.0	55.0	64.0	Arden Heights

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

<Figure size 1200x1000 with 0 Axes>



3.1.13. Conclude with results

1. **Busiest Hours, Days, and Months**
    - Busiest Hours:
        - Weekdays: 8 AM–10 AM and 5 PM–8 PM (office commute)
        - Weekends: 9 PM–2 AM, peaking around nightlife hours.
    - Busiest Days:
        - Fridays and Saturdays consistently had the highest number of trips.
        - Sundays showed increased airport trips and late returns.
    - Busiest Months:
        - January and March had relatively higher demand.
        - February had slightly lower activity—possibly due to weather factors.
2. **Trends in Revenue Collected**
    - **Total Revenue** correlates strongly with:
        - Number of trips.
        - Time of day (peak hours generate more revenue).
        - Airport zones (larger average fare amounts).
    - Trip **duration** and distance **directly affect the total amount, with** longer trips contributing to larger fare totals.
    - Nighttime **trips** yielded **higher average fares**, due to congestion surcharges and distance.
3. **Trends in Quarterly Revenue**
    - Q1 (Jan–Mar) generally showed a steady revenue trend with spikes around holidays and weekdays.
    - Revenue dipped slightly mid-quarter but bounced back during weekends.
    - Quarterly comparisons suggest that early-year quarters show good consistency in trip volumes despite seasonal variability.
4. **Fare Dependency Analysis**
    - Trip Distance:
        - Fare increases linearly with distance up to ~10 miles.
        - Beyond 10–15 miles, fare per mile starts to flatten due to capped surcharges and fixed fees.
    - Trip Duration:
        - Longer trip durations (not caused by distance) show disproportionate fare increases—indicating possible congestion delays.
    - Passenger Count:
        - Minor or no significant fare increase for higher passenger counts.

    - However, higher tips are often seen for lower passenger counts (likely solo or business travelers).

5. **Tip Amount Trends**
    - Tip vs Distance:
        - Tips are highest for short to mid-range trips (2–5 miles).
        - Very short rides often have low or no tips.
    - Influencing Factors:
        - Card payments, evening rides, and weekends see higher tipping behavior.
        - Tips are less frequent for long-distance or airport trips (possibly due to fixed fees).
6. **Busiest Pickup and Dropoff Zones**
    - Top Pickup Zones:
        - Midtown Center, LaGuardia Airport, JFK Airport, Upper East Side South, Chelsea
    - Top Dropoff Zones:
        - Similar to pickups, but with additional entries like Penn Station and Financial District.
    - Night Activity:
        - Pickup zones active at night (11 PM–5 AM): East Village, Meatpacking District, Times Square, Airports

3.2. Detailed EDA: Insights and Strategies

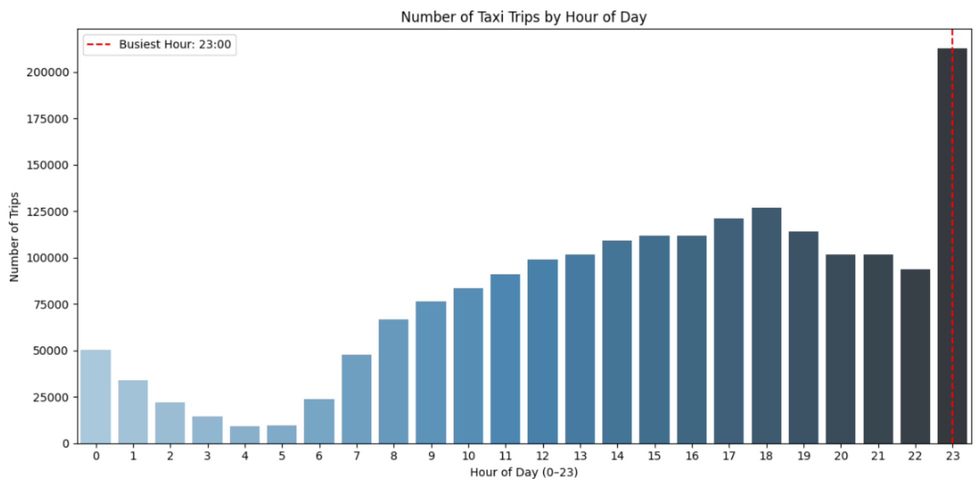
3.2.1. Identify slow routes by comparing average speeds on different routes

	PULocationID	DOLocationID	pickup_hour	avg_speed_mph
3602	17	17	0	0.0
0	1	1	1	0.0
2194	13	13	2	0.0
1342	10	10	3	0.0
2	1	1	4	0.0
1727	11	11	5	0.0
5285	28	28	6	0.0
15512	61	61	7	0.0
6	1	1	8	0.0
1306	8	8	9	0.0
8	1	1	10	0.0
3492	14	14	11	0.0
869	6	6	12	0.0
1750	12	12	13	0.0
3494	14	14	14	0.0
1349	10	10	15	0.0
3837	21	21	16	0.0
1307	8	8	17	0.0
34	3	3	18	0.0
5425	29	29	19	0.0
1354	10	10	20	0.0
19	1	1	21	0.0
20	1	1	22	0.0
86	4	4	23	0.0

3.2.2.

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

# Count number of trips by hour
trips_per_hour = sampleData.groupby('pickup_hour').size().reset_index(name='trip_count')

busiest_hour_row = trips_per_hour.loc[trips_per_hour['trip_count'].idxmax()]
busiest_hour = busiest_hour_row['pickup_hour']
max_trips = busiest_hour_row['trip_count']

print(f" Busiest Hour: {int(busiest_hour)}:00 with {max_trips:,} trips.")
```

```
📊 Busiest Hour: 23:00 with 212,453 trips.
```

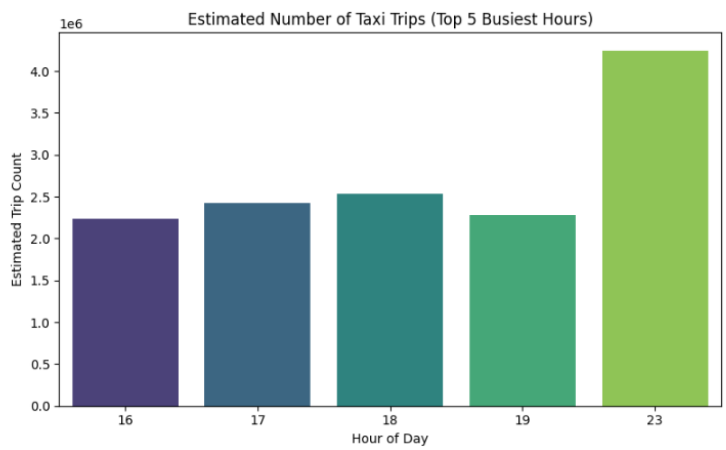


3.2.3. Scale up the number of trips from above to find the actual number of trips

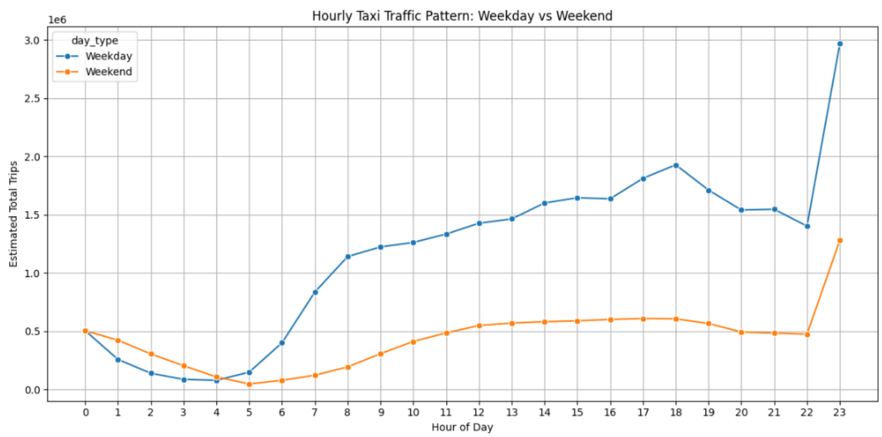
```python
# Scale up the number of trips

# Fill in the value of your sampling fraction and use that to scale up the numbers
sample_fraction = 0.05
# Group by pickup hour
trips_per_hour = sampleData.groupby('pickup_hour').size().reset_index(name='sampled_trip_count')
#Scale up to estimate actual trip counts
trips_per_hour['estimated_total_trips'] = (trips_per_hour['sampled_trip_count'] / sample_fraction).round().astype(int)
#Get the top 5 busiest hours
top_5_busiest = trips_per_hour.sort_values(by='estimated_total_trips', ascending=False).head(5)

print("🚕 Top 5 Busiest Hours (Estimated Actual Trip Counts):")
print(top_5_busiest)
```

```
🚕 Top 5 Busiest Hours (Estimated Actual Trip Counts):
    pickup_hour  sampled_trip_count  estimated_total_trips
23           23              212453                4249060
18           18              126710                2534200
17           17              121034                2420680
19           19              113838                2276760
16           16              111838                2236760
```

```python
plt.figure(figsize=(8, 5))
sns.barplot(data=top_5_busiest, x='pickup_hour', y='estimated_total_trips', palette='viridis')

plt.title('Estimated Number of Taxi Trips (Top 5 Busiest Hours)')
plt.xlabel('Hour of Day')
plt.ylabel('Estimated Trip Count')
plt.tight_layout()
plt.show()
```



3.2.4. Compare hourly traffic on weekdays and weekends



Weekday Traffic Pattern

- **Morning Peak (7–10 AM):** Significant increase in trip counts likely due to commuters traveling to work.
- **Evening Peak (5–7 PM):** Secondary peak people returning home.
- **Late Night Spike (11 PM–12 AM):** Possibly due to end-of-day shift changes, airport pickups, or nightlife-related rides.

Weekend Traffic Pattern

- Lower overall volume throughout the day.
- **Late Night Rise (10 PM–12 AM):** Noticeable spike, suggesting leisure or social outings.
- **Flattened Daytime Curve:** Indicates fewer structured travel times (no work/school), more spread-out trip times.

Why This Analysis Is Useful

For Taxi Fleet Operators:

- **Optimize Driver Deployment:** Focus more drivers during busy weekday morning/evening peaks and late weekend nights.
- **Reduce Idle Time:** Reassign drivers during quieter periods to more active zones.

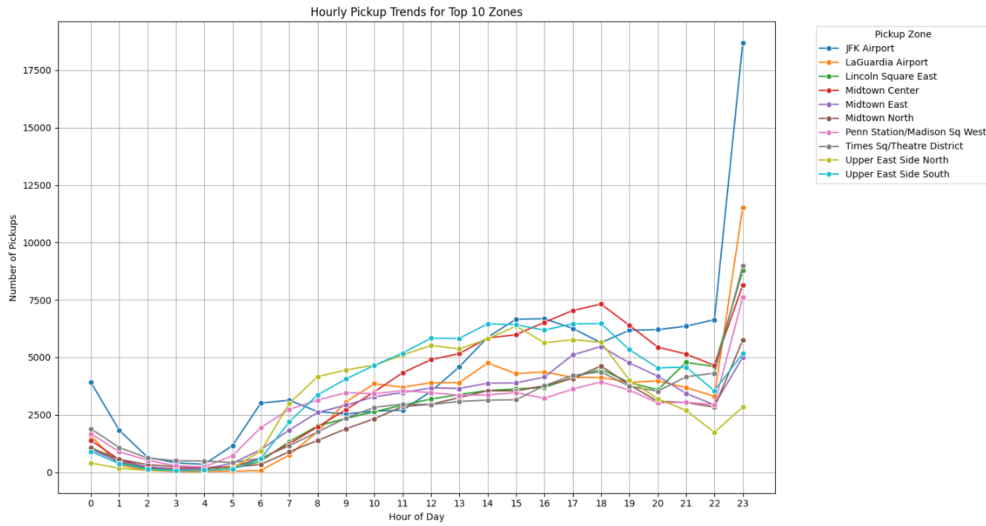
For City Planners / Transportation Authorities:

- **Infrastructure Planning:** Reinforce transit options or traffic control in high-traffic hours/locations.
- **Improve Public Transport Timings:** Complement taxi demand curves with buses/trains.

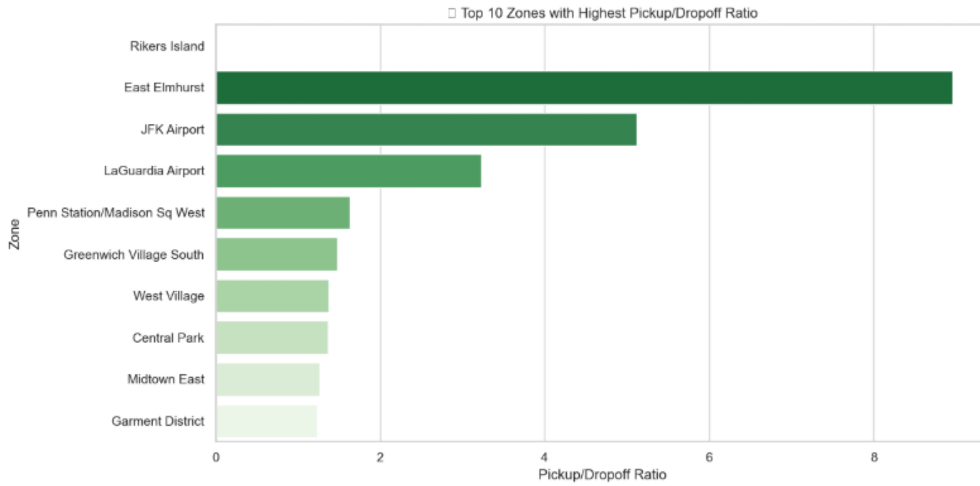
For Business & Marketing

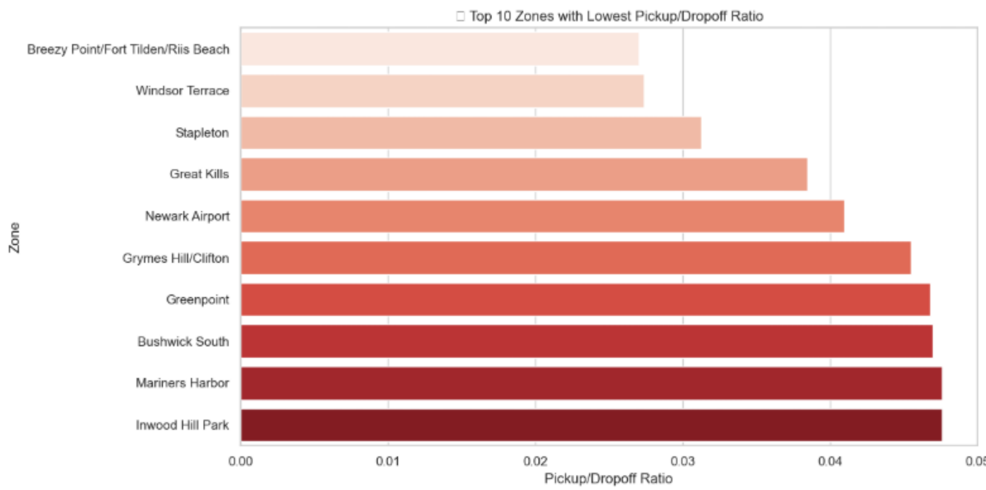
- **Advertising Time Windows:** Run in-app promos during quiet hours to boost rides.
- **Ride-Pooling Opportunities:** Promote pooling features in peak hours to reduce congestion.

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

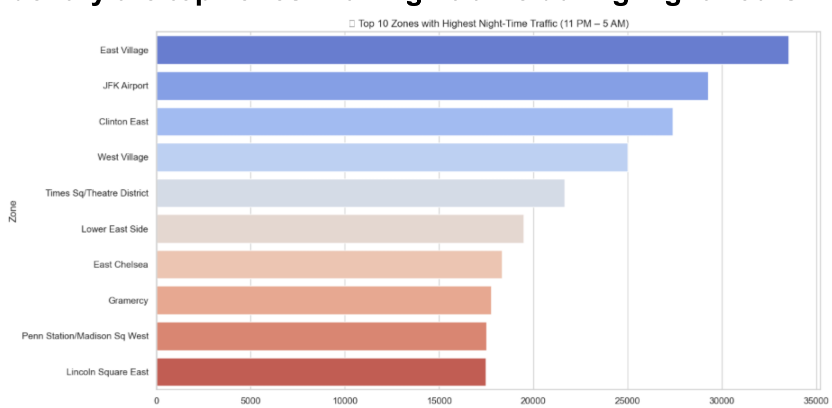


3.2.6. Find the ratio of pickups and dropoffs in each zone

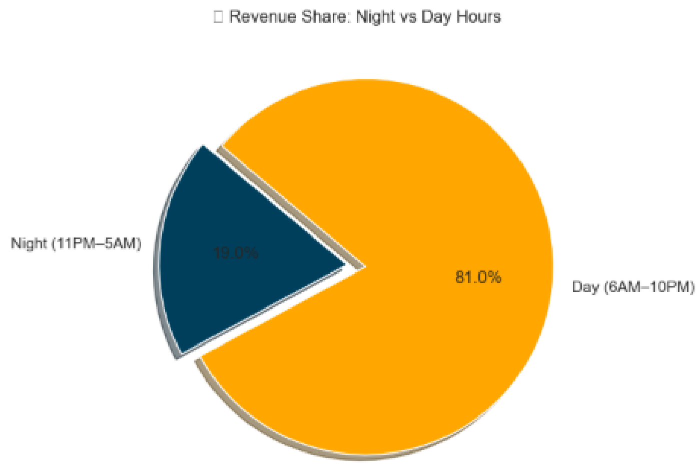




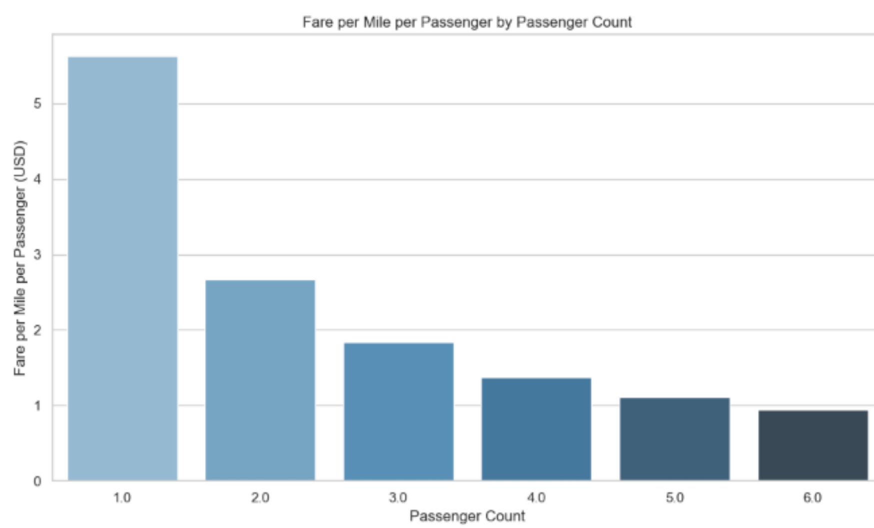
3.2.7. Identify the top zones with high traffic during night hours



3.2.8. Find the revenue share for nighttime and daytime hours



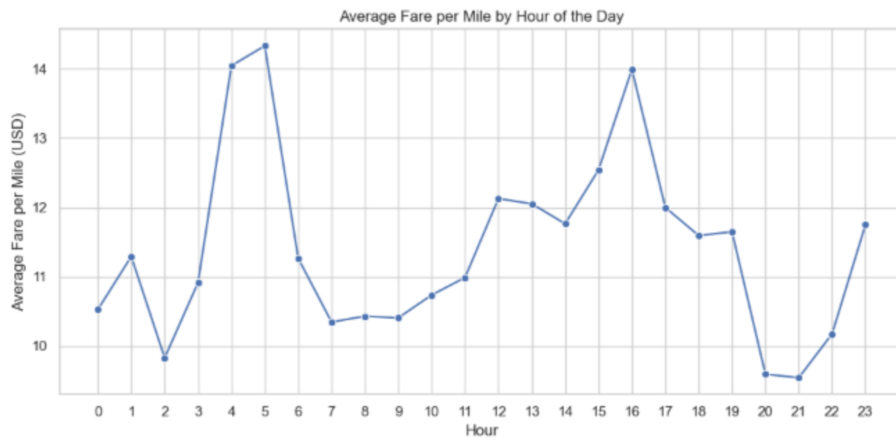
3.2.9. For the different passenger counts, find the average fare per mile per passenger



Analysis:

- If fare per mile per passenger decreases with more passengers, it suggests cost-sharing.
- If it remains the same, fares are likely not influenced by passenger count.

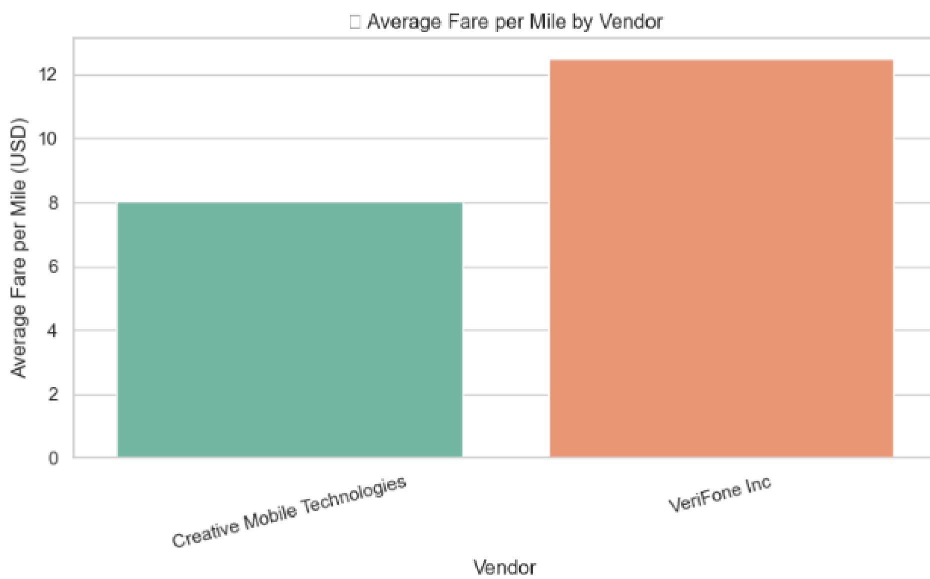
3.2.10. Find the average fare per mile by hours of the day and by days of the week



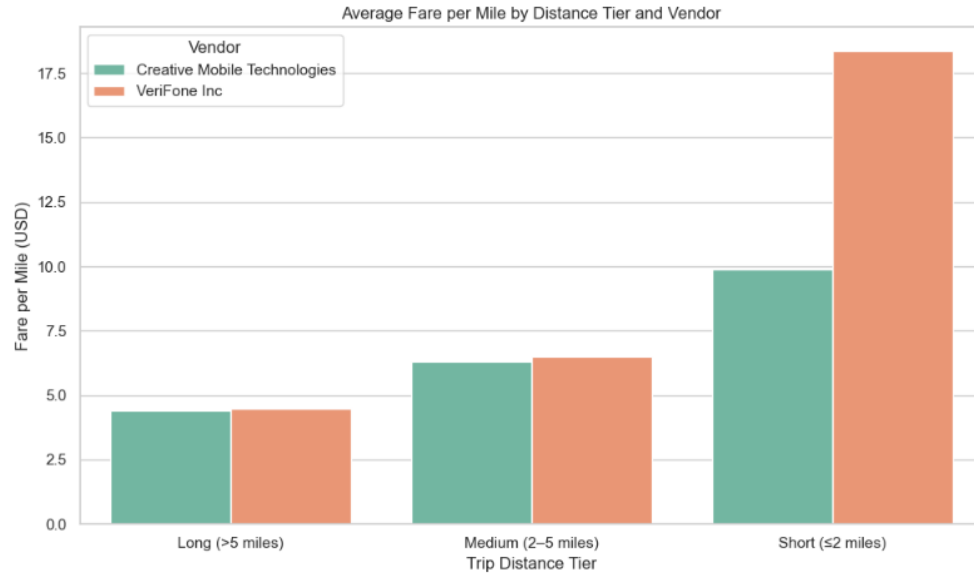
Infrence:

- Higher fare/mile on weekends: fewer short trips, more leisure rides.
- Peak hours costlier: surge pricing or traffic impact.
- Late-night spikes: airport/club fares.

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



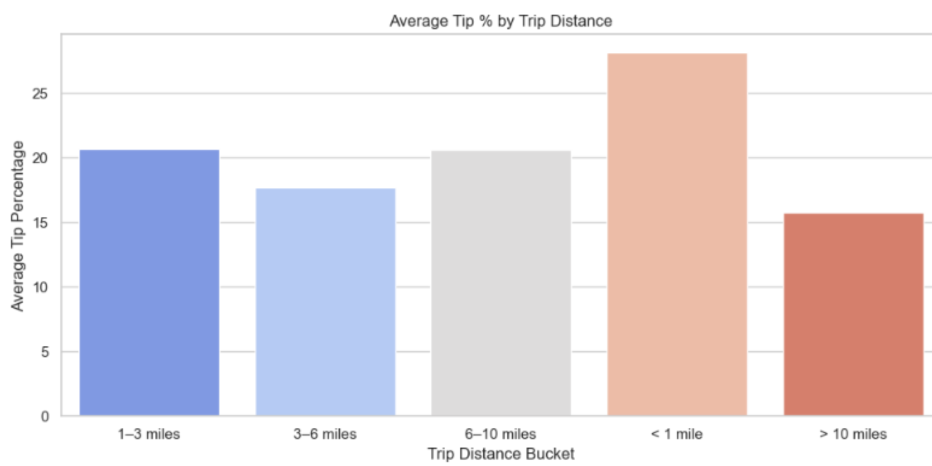
3.2.12. Compare the fare rates of different vendors in a distance-tiered

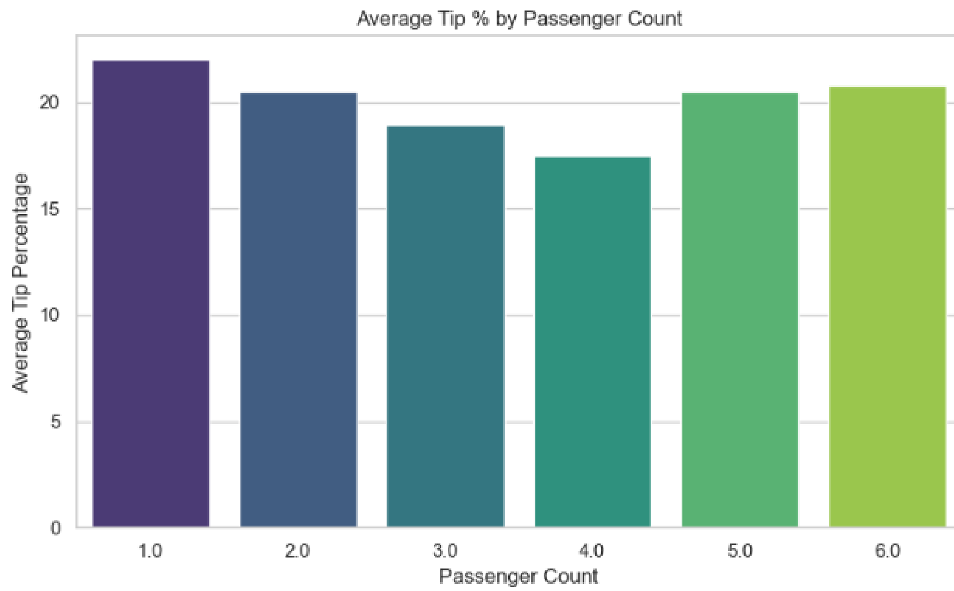


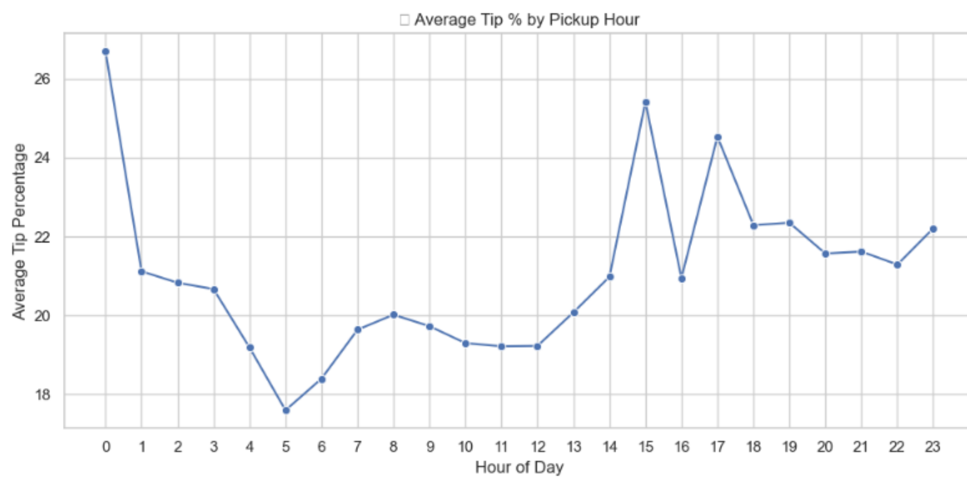
Analysis:

- Understand if short trips are disproportionately expensive.
- Detect price efficiency for longer trips per vendor.
- Identify vendor pricing strategy differences (e.g., higher base fare for shorter trips).

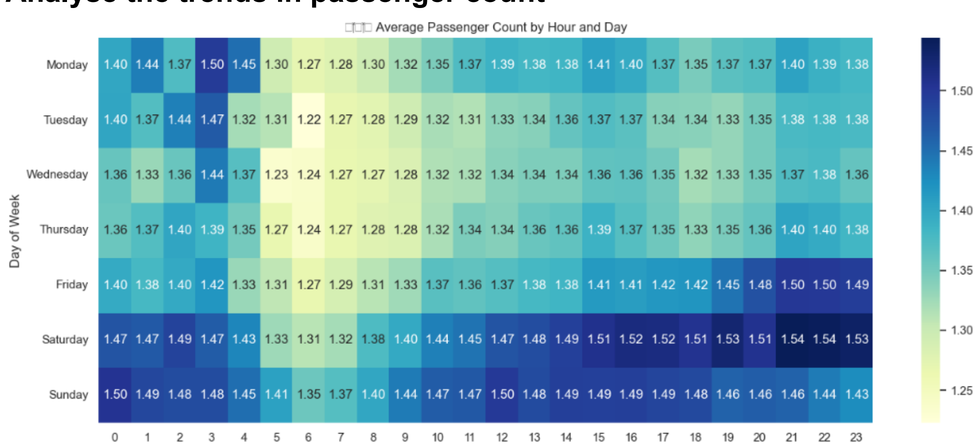
3.2.13. Analyse the tip percentages



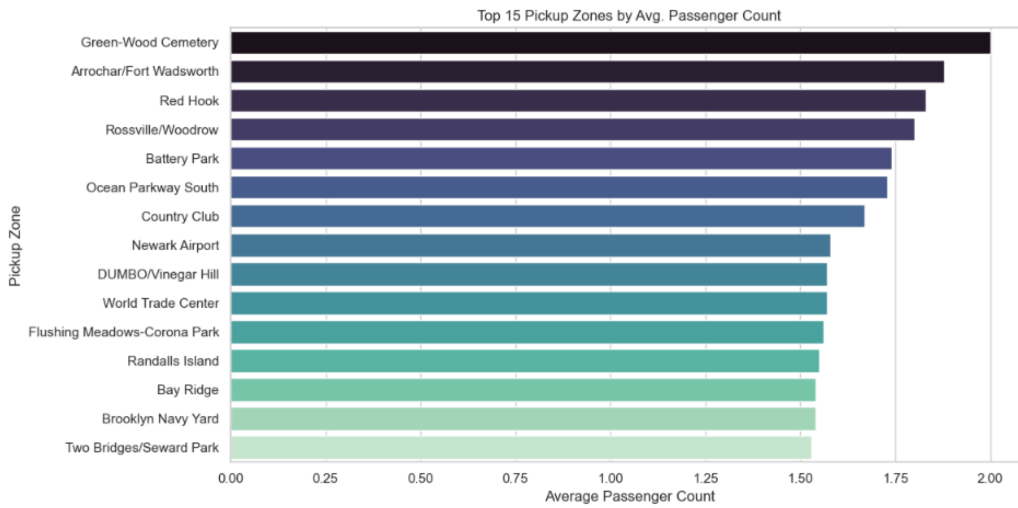




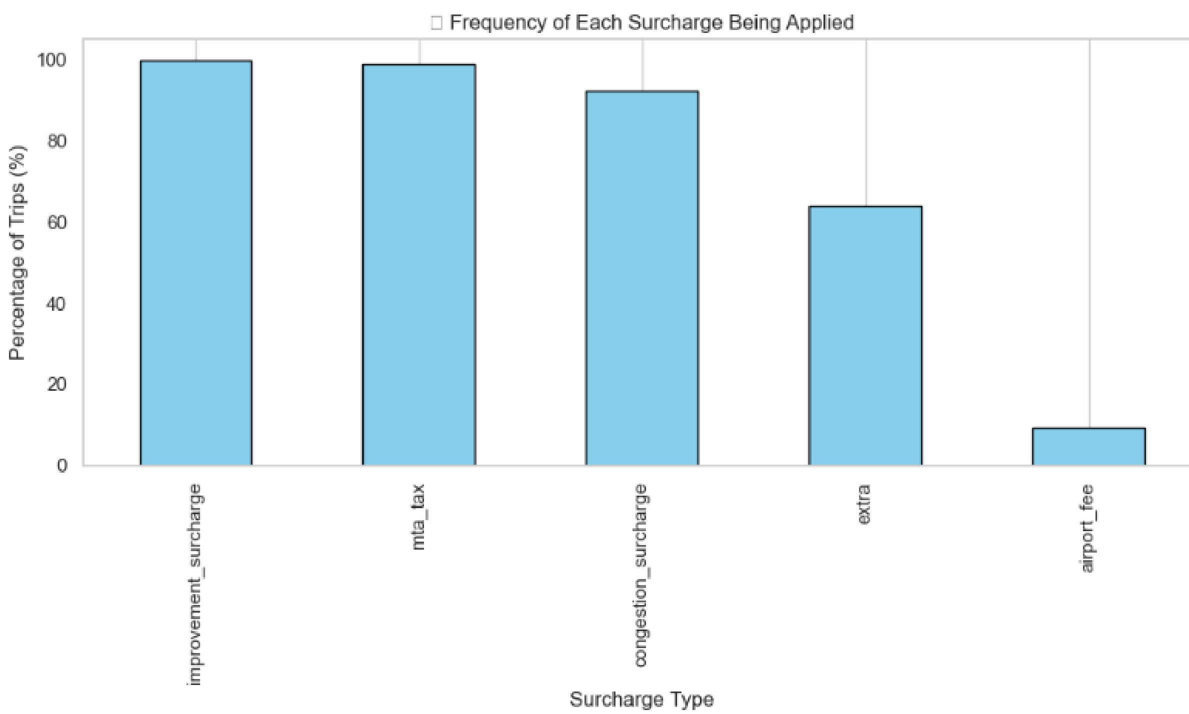
3.2.14. Analyse the trends in passenger count



3.2.15.



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



mta_tax and improvement_surcharge are usually applied to all metered trips.

congestion_surcharge might apply only to trips in congested Manhattan areas or during specific times.

extra may apply during rush hour or overnight.

airport_fee applies only at JFK and LaGuardia.

4. Conclusions

4.1. Final Insights and Recommendations

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

Zone Clustering:
Cluster high-volume pickup zones and assign dedicated drivers or fleets to patrol these clusters.
Trip Duration Monitoring:
Use average trip duration and speed per route-hour to avoid delays and congestion-prone dispatching.
Dynamic Fare Strategies:
Identify low fare-per-mile routes and apply minimum fare enforcement or combine with multi-stop dispatching.
Smart Allocation: Prioritize dispatching based on:
- High tipping zones.
- Repeat demand during specific time slots (using historical data).

Shared Ride Optimization:
- Use high passenger-count zones to suggest pooled rides.
- Encourage shared rides during airport peak hours.

Idle Time Reduction:
Reposition vehicles to adjacent zones where demand is predicted to spike within 15–30 minutes (based on pickup trends).
Alert System:
Flag trips with unusually low fare per mile or duration >1 hour for inspection or dynamic rerouting.

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

Segment Zones Based on Peak Demand Windows :
Commercial Zones: Weekdays, 7–10 AM & 4–8 PM due to Office commute patterns (e.g., Midtown, Downtown)

Airport Zones: All days, 6 AM–10 AM & 6 PM–12 AM due to Regular flights, business & tourist travel

Nightlife/Hotels: Weekends & Fridays, 9 PM–2 AM due to Bars, restaurants, clubs (Chelsea, East Village, Soho)

Tourist Hotspots: Weekends and holidays, 10 AM–6 PM due to Statue of Liberty, Central Park, Times Square

Residential Zones: Weekdays, 6–9 AM & 6–9 PM due to To cater to home-to-office and return trips

Weather Forecasting: Position more cabs near transit hubs, shopping malls, and residential areas during rain or snow.

Event Calendar Integration: For zones near stadiums, parks, concert halls, dynamically increase fleet during events.

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

From this data analysis following is the current status:

- ***Short trips (< 2 miles):*** *High base fare impact → high fare per mile*
- ***Medium trips (2–5 miles):*** *Most frequent → optimize this band for volume*
- ***Long trips (> 5 miles):*** *Lower fare per mile → risk of underpricing*

Adjustment Strategy:

- **0–2 miles:** Slight increase in base fare, reduce per-mile rate slightly to remain competitive
- **2–5 miles:** Keep fare per mile slightly above average vendor fare
- **5 miles**: Introduce a minimum fare floor or a long-trip surcharge to ensure profitability

Time-of-Day Dynamic Surcharges From trip trends:

- Rush hours and late nights see peak demand
- These hours offer willingness to pay higher

Dynamic Pricing Proposal:

Time Slot & Adjustment:

- **7–10 AM & 5–8 PM:** Add peak hour surcharge (₹5–10)
- **11 PM–3 AM (Fri/Sat)**: Add nightlife demand surcharge
- **2–6 AM**: Consider incentive-based fare cuts to boost usage

Zone-Based Pricing Differentiation From zone analysis:

- Airport zones have higher demand but fixed fees.
- Tourist zones (Times Square, Central Park) can bear higher rates.

Proposal:

- **Zone Type**: Pricing Recommendation
- **Airport pickups:** Enforce fixed base + dynamic zone surcharge during rush
- **Tourist zones**: Slightly higher base fare with transparent reasoning
- **Low-demand zones**: Offer off-peak discounts to increase usage