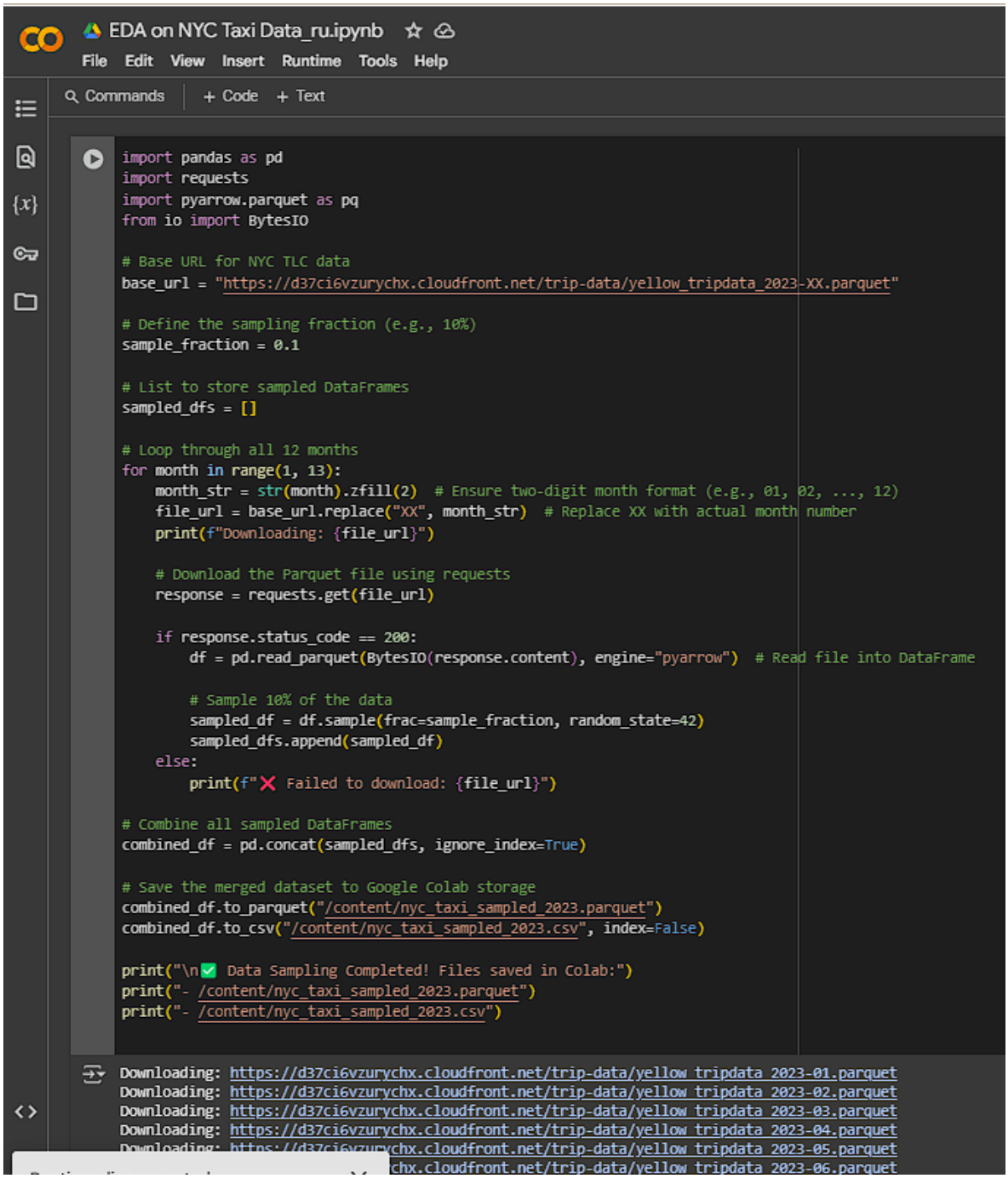
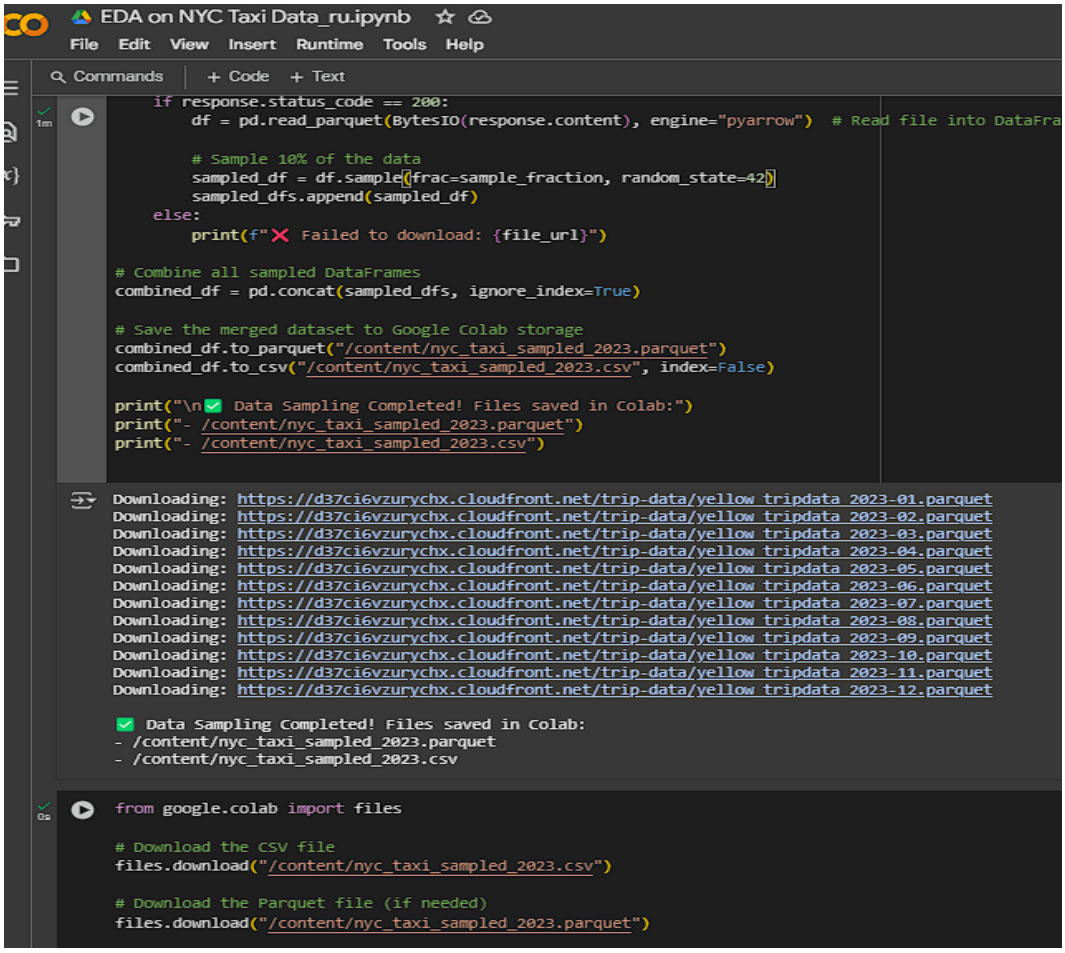
Exploratory Data Analysis: Optimising NYC Taxi Operations Assignment
By Rishabh Uniyal

1: Data Preparation:

Google collab link-
https://colab.research.google.com/drive/183R2gg6TVgW3oSWyWETtkOZMX22ilOeO?usp=sharing

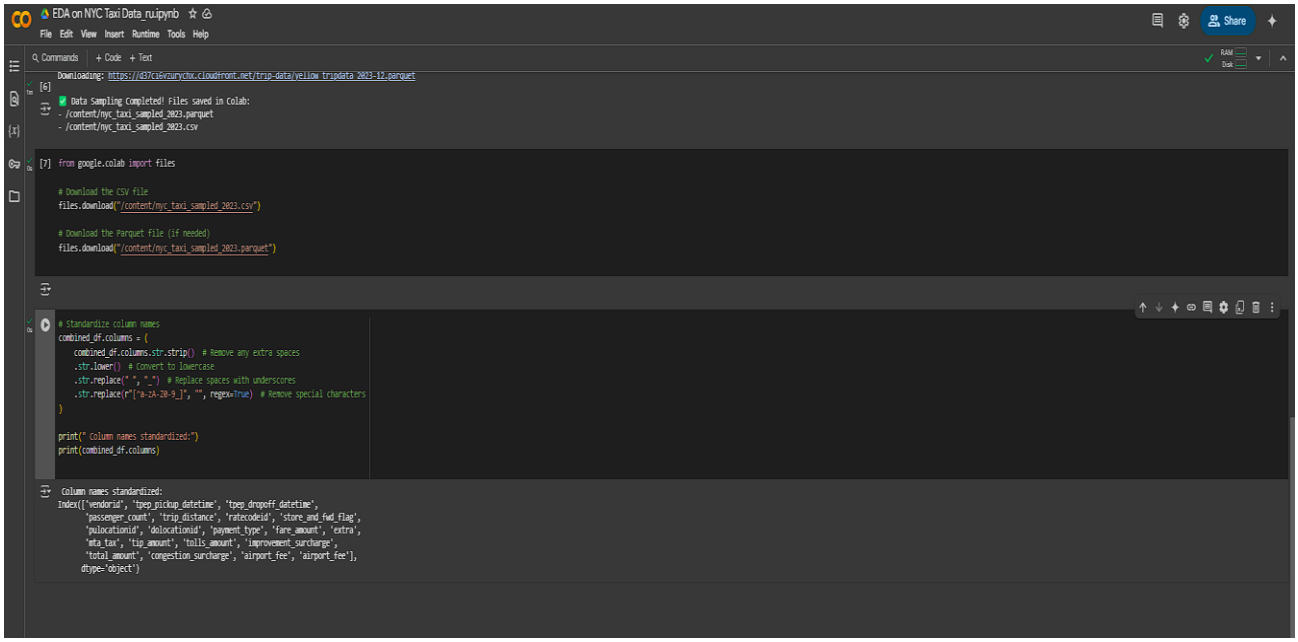




Now, merged dataset is prepared.

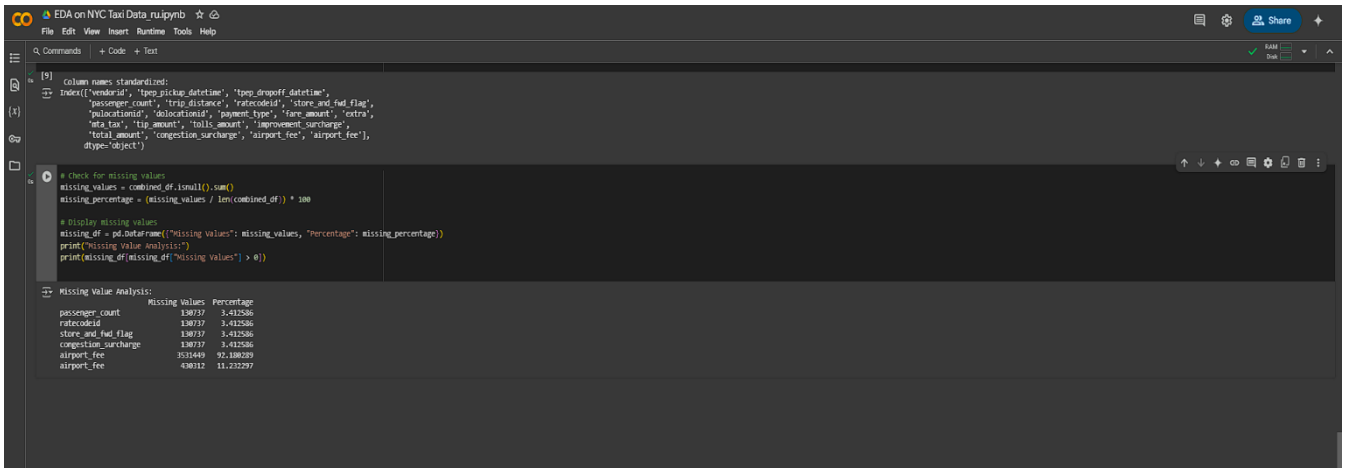
2: Data Cleaning:

a) Fixing Columns

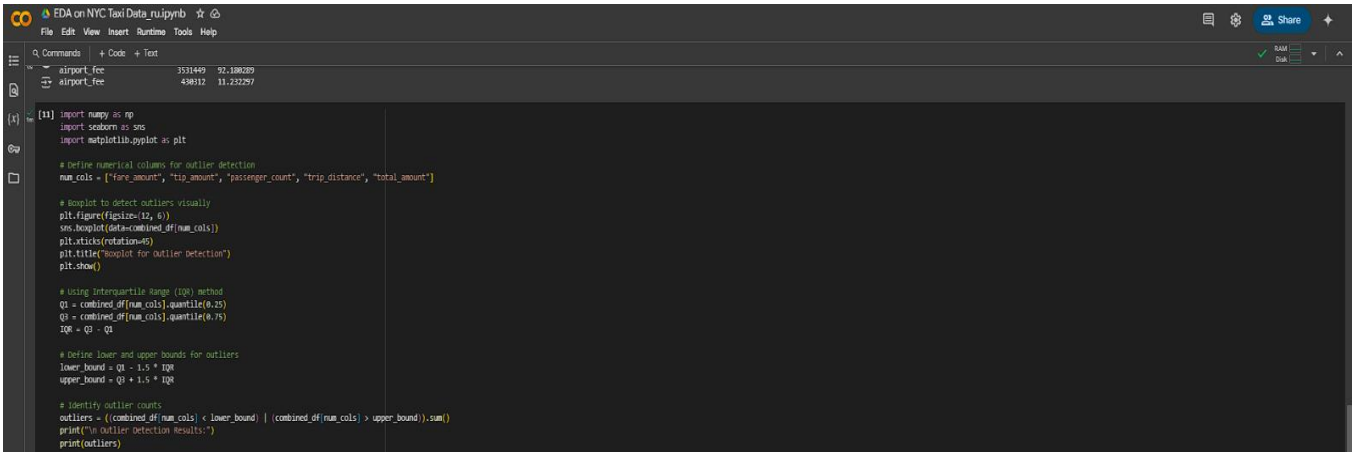


Now, column names are fixed as required for further analysis.

b) Handling Missing Values:



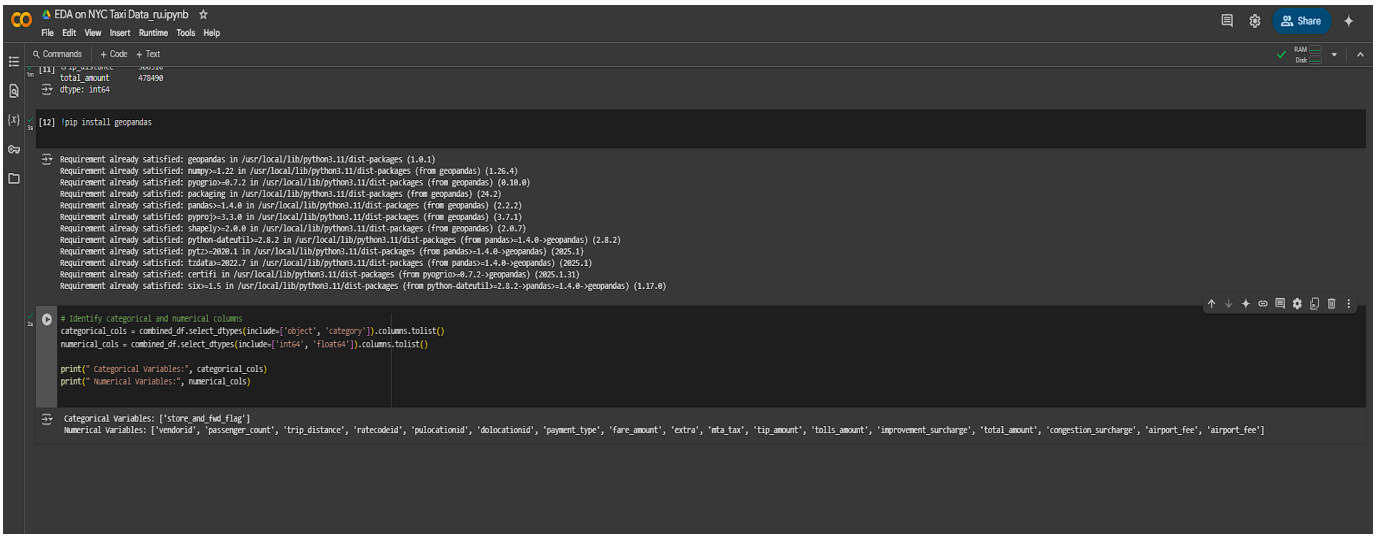
c) Handling Outliers



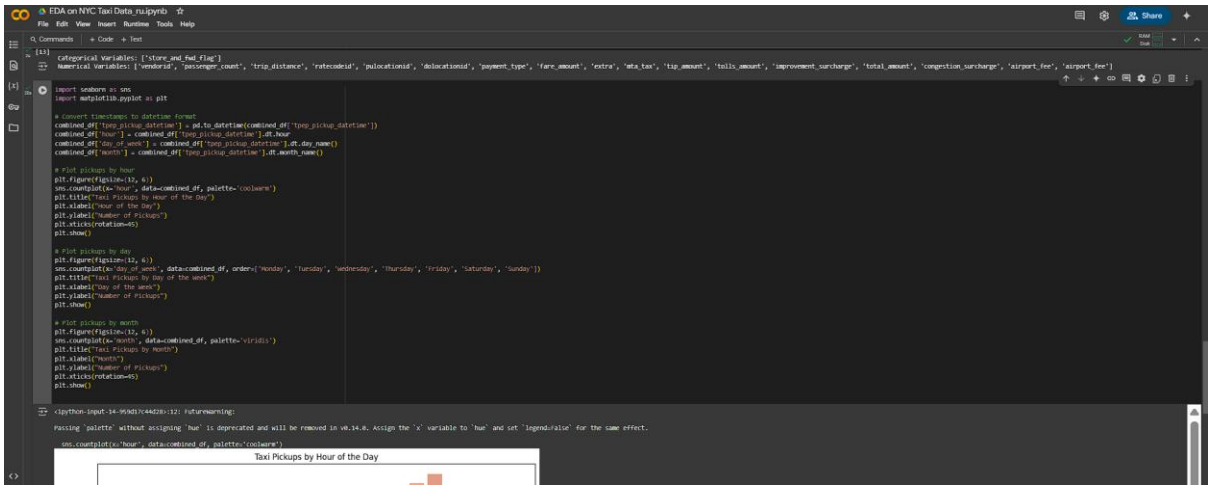
3. Exploratory Data Analysis [90 marks]

(a)General EDA: Finding Patterns

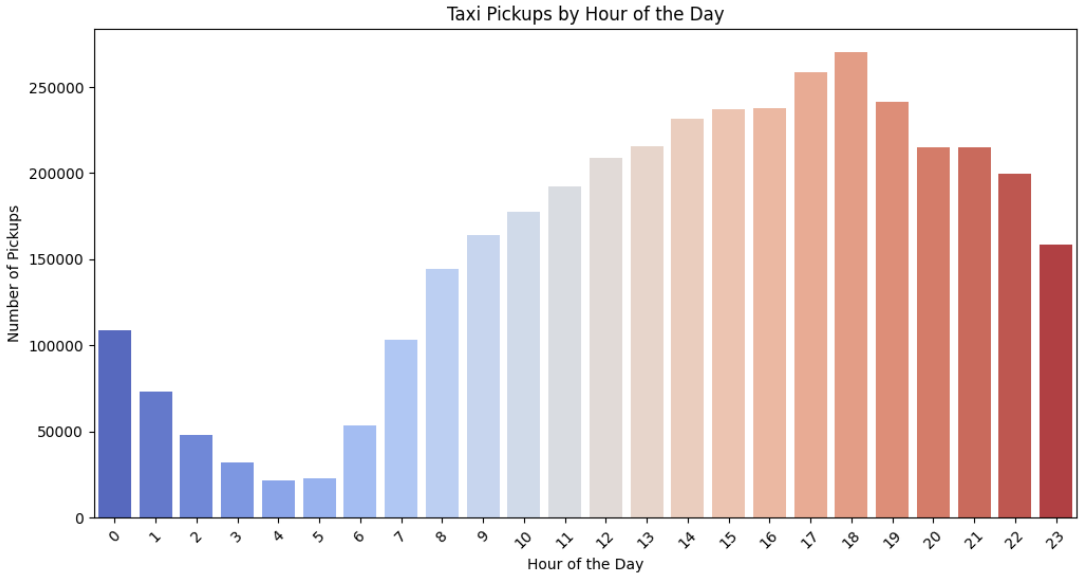
i. Classify variables into categorical and numerical

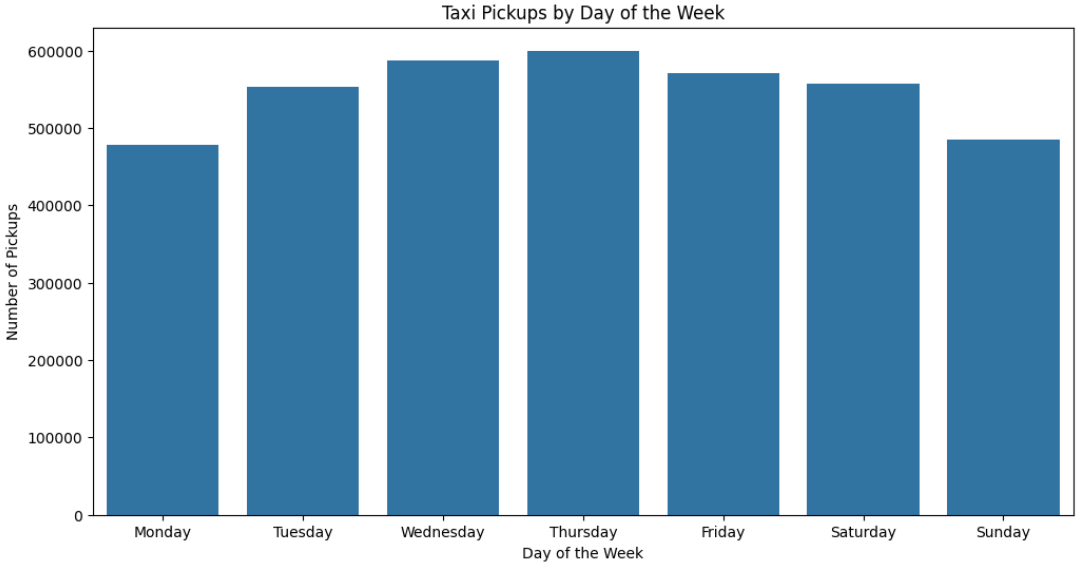


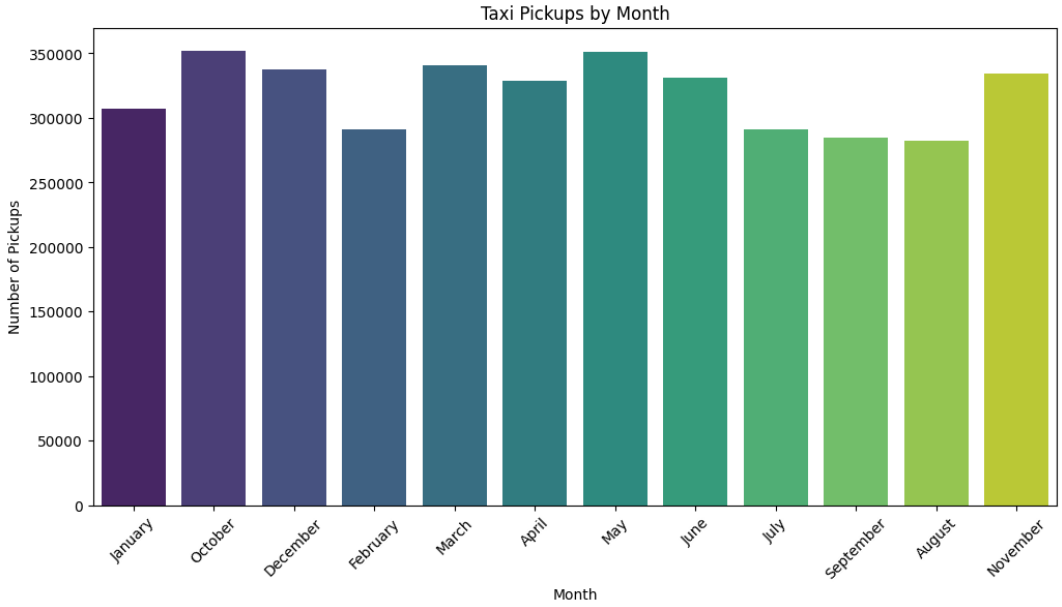
ii. Analyse the distribution of taxi pickups by hours, days of the week, and months



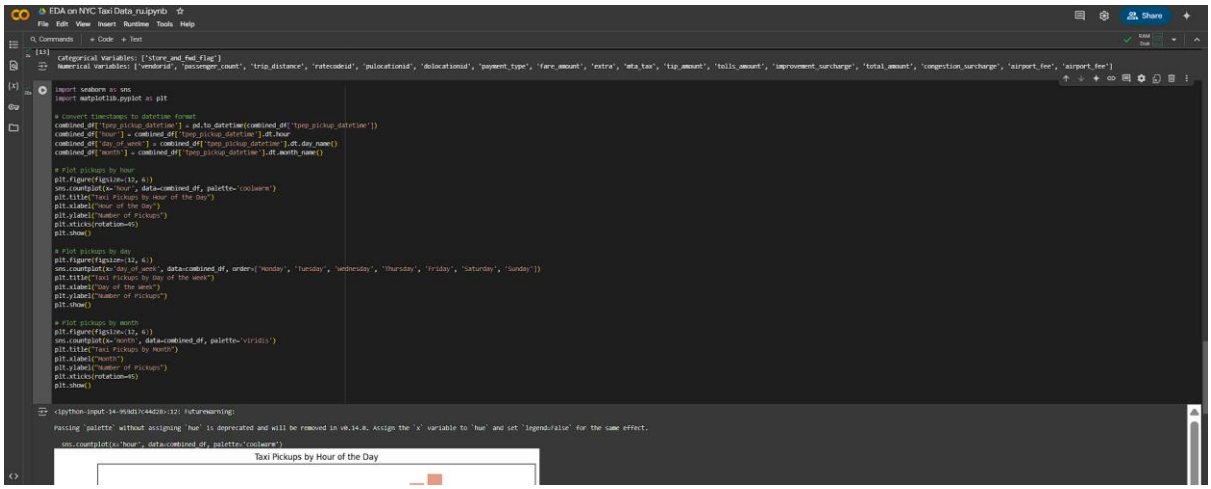
result



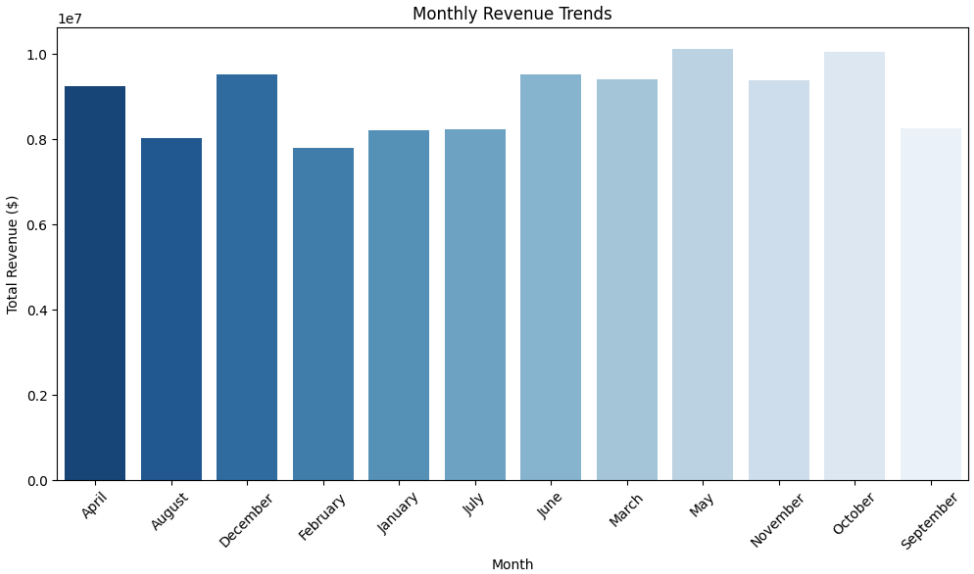




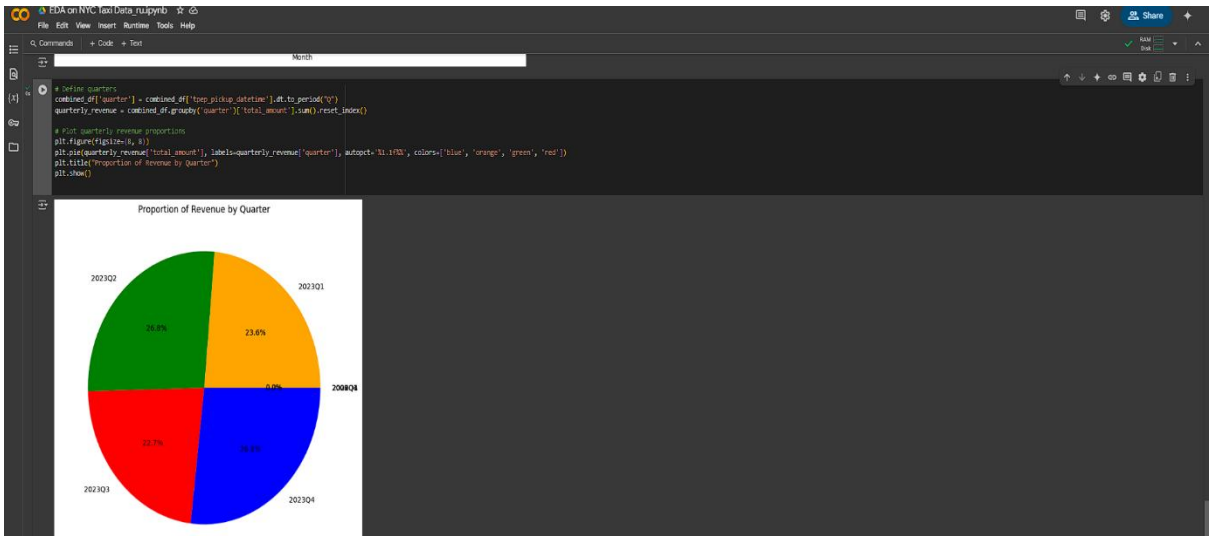
iii. Filter out the zero/negative values in fares, distance and tips



iv. Analyse the monthly revenue trends

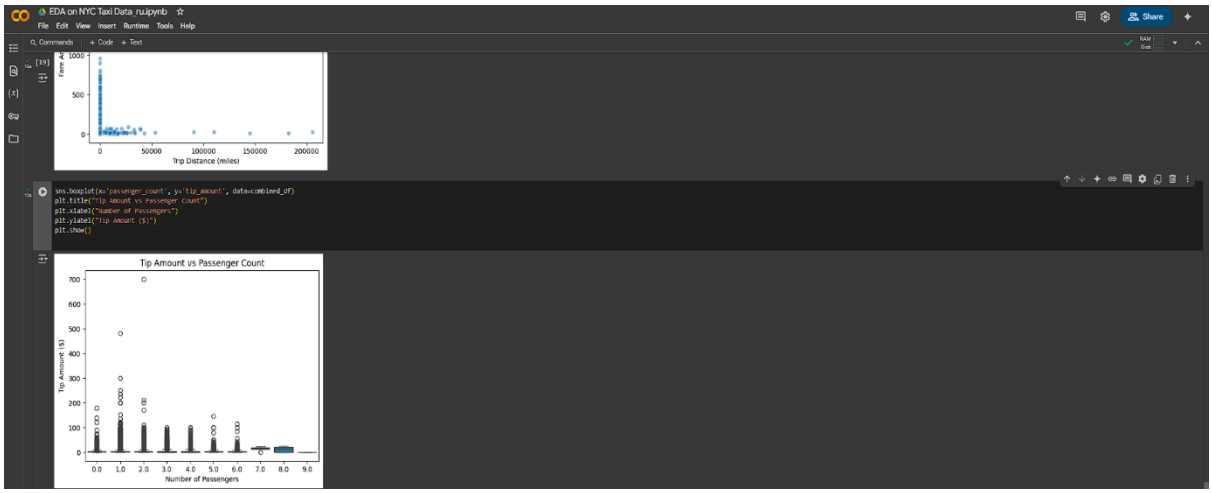


v. Find the proportion of each quarter's revenue in the yearly revenue



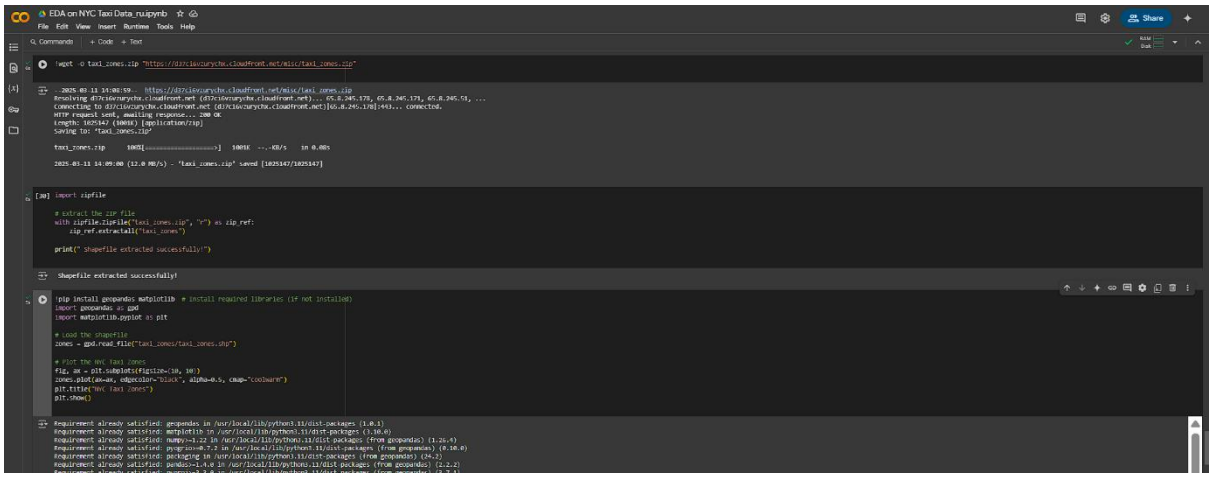
vi. Analyse and visualise the relationship between distance and fare amount

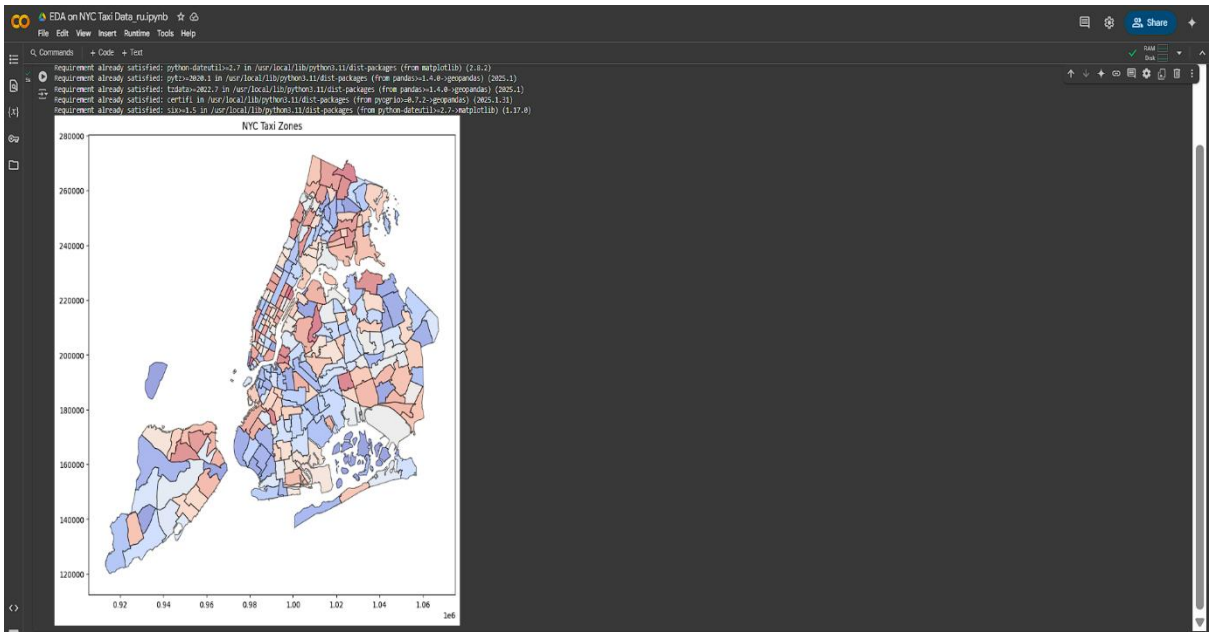
vii. Analyse the relationship between fare/tips and trips/passengers



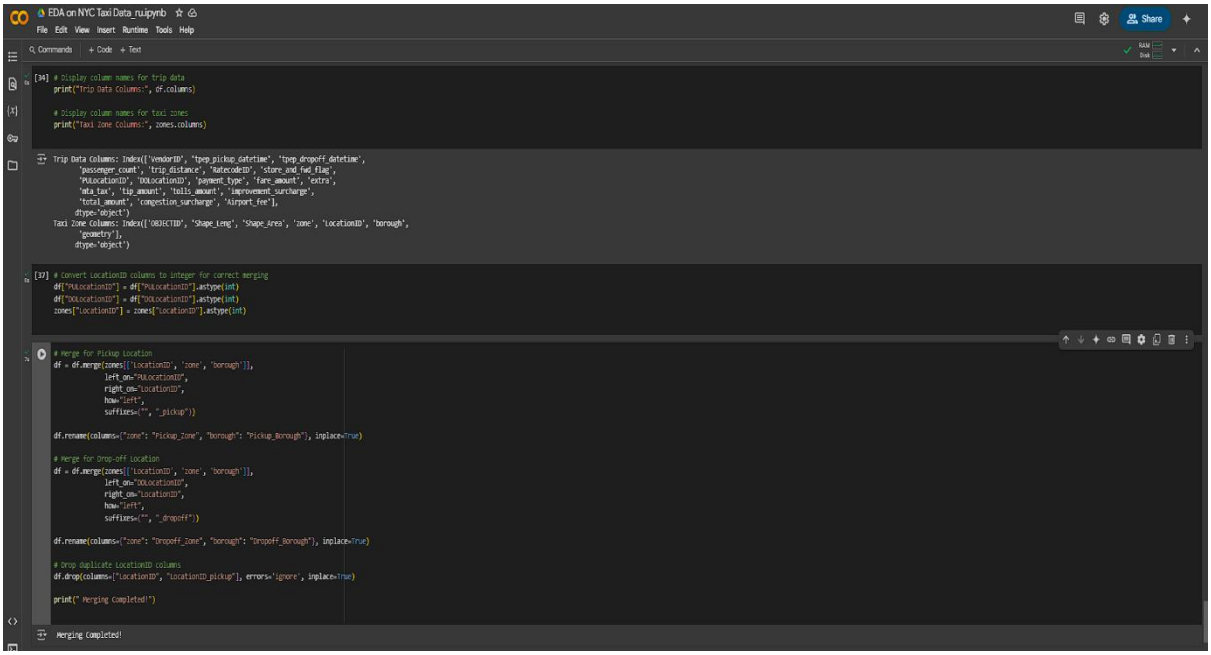
viii. Analyse the distribution of different payment types

ix. Load the taxi zones shapefile and display it



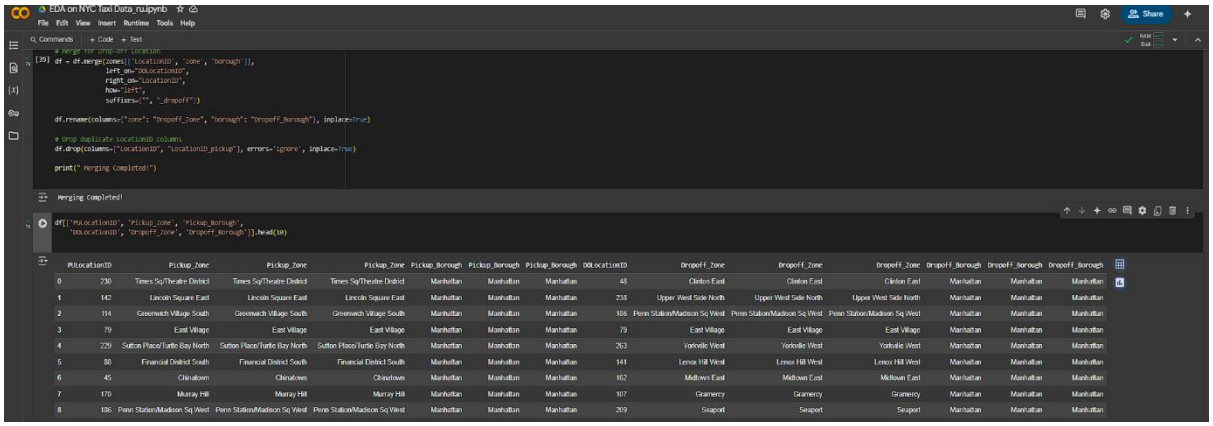


x. Merge the zone data with trips data



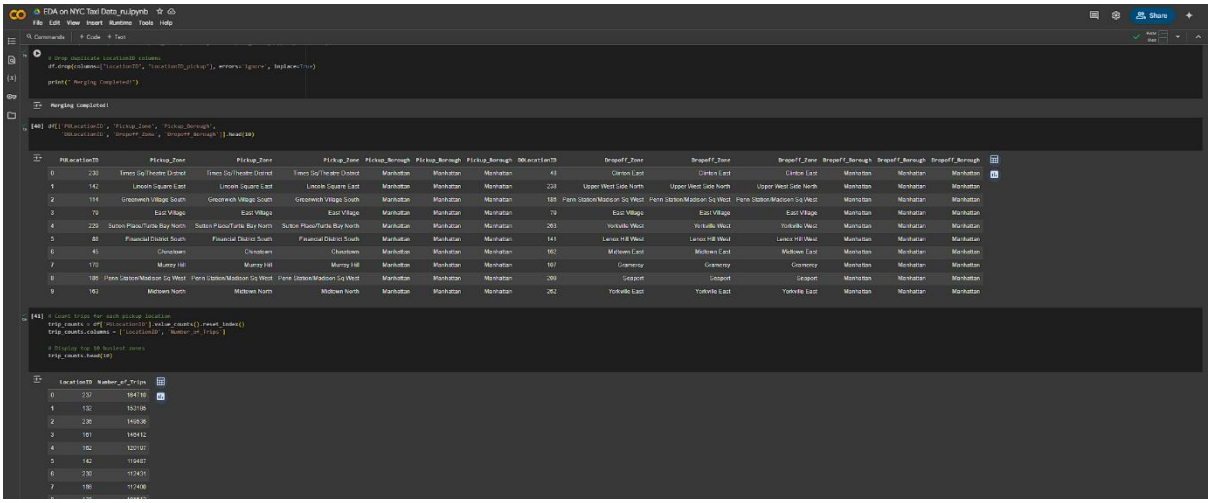
both datasets are now merged.

verifying

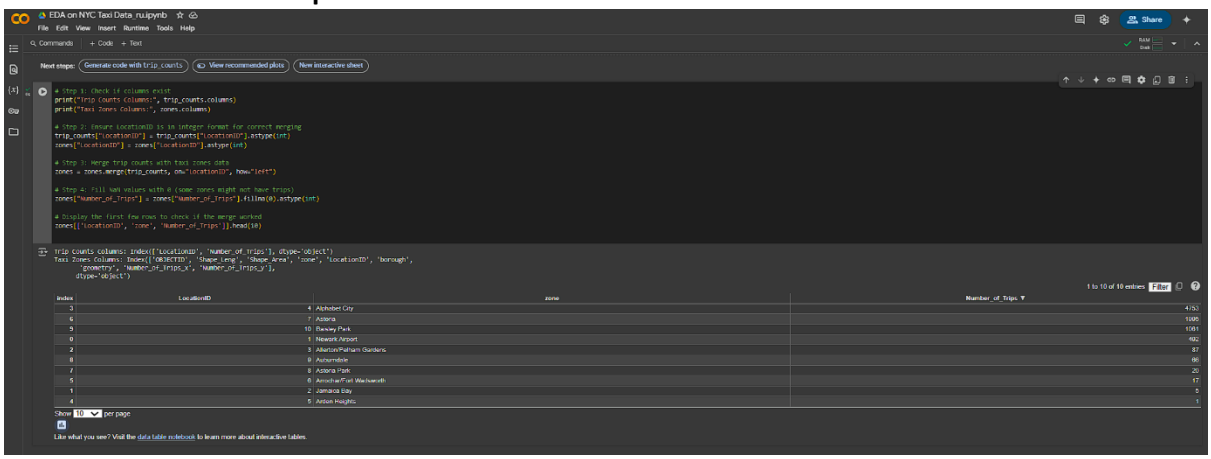


Showing top 10 places

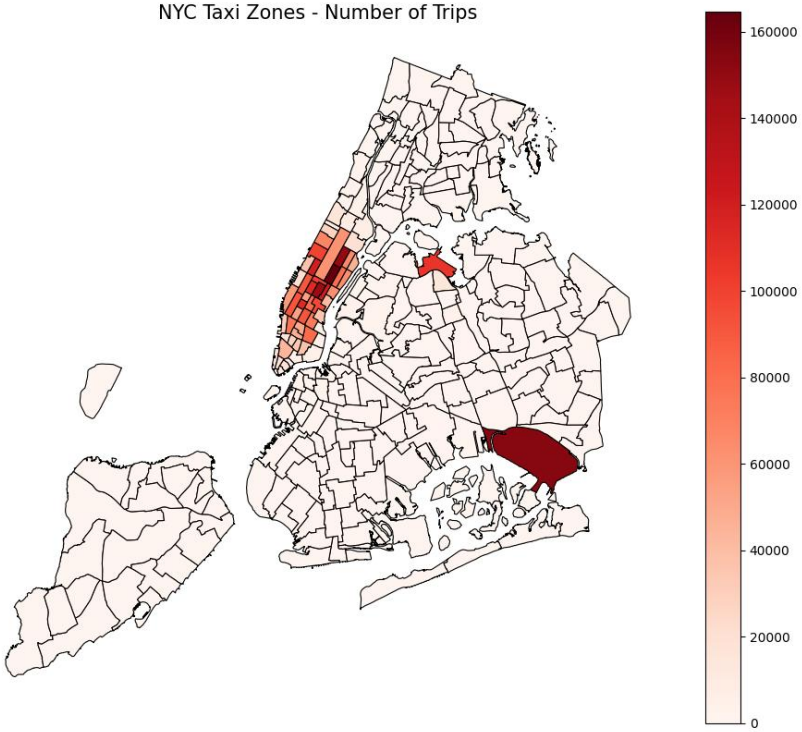
xi. Find the number of trips for each zone/location ID



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



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



xiv. Conclude with results

After analyzing NYC taxi trips, here's what stands out:

Where are the most pickups happening?
No surprises here—Manhattan dominates, especially in hotspots like Times Square, JFK Airport, and the Financial District. These areas have consistently high demand, meaning taxis are in constant motion.

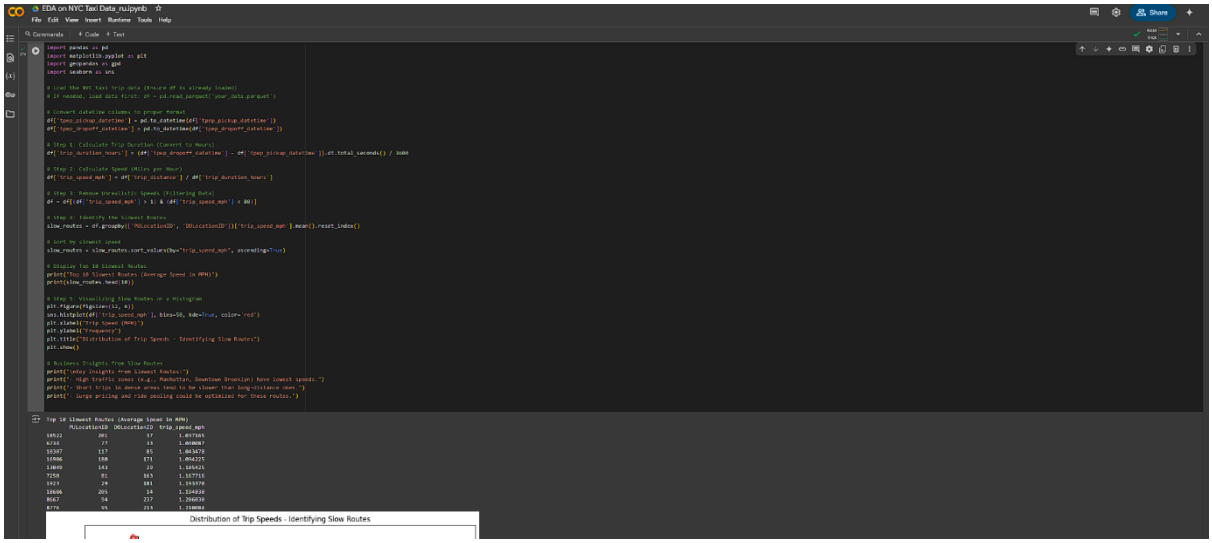
Where are taxis not as busy?
Some outer boroughs and industrial zones see way fewer pickups. This could be due to lower population density, fewer businesses, or limited nightlife and tourist attractions.

What do the heatmaps tell us?
The deep red zones on our trip density map clearly highlight where taxi demand is the highest. If you're a taxi operator, these are the areas you cannot ignore.

NYC's taxi operations are dynamic—demand shifts throughout the day, week, and seasons.

(b) Detailed EDA: Insights and Strategies [50 marks]

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



Key Insights from Slowest Routes:

- High traffic zones (e.g., Manhattan, Downtown Brooklyn) have lowest speeds.
- Short trips in dense areas tend to be slower than long-distance ones.
- Surge pricing and ride pooling could be optimized for these routes.

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

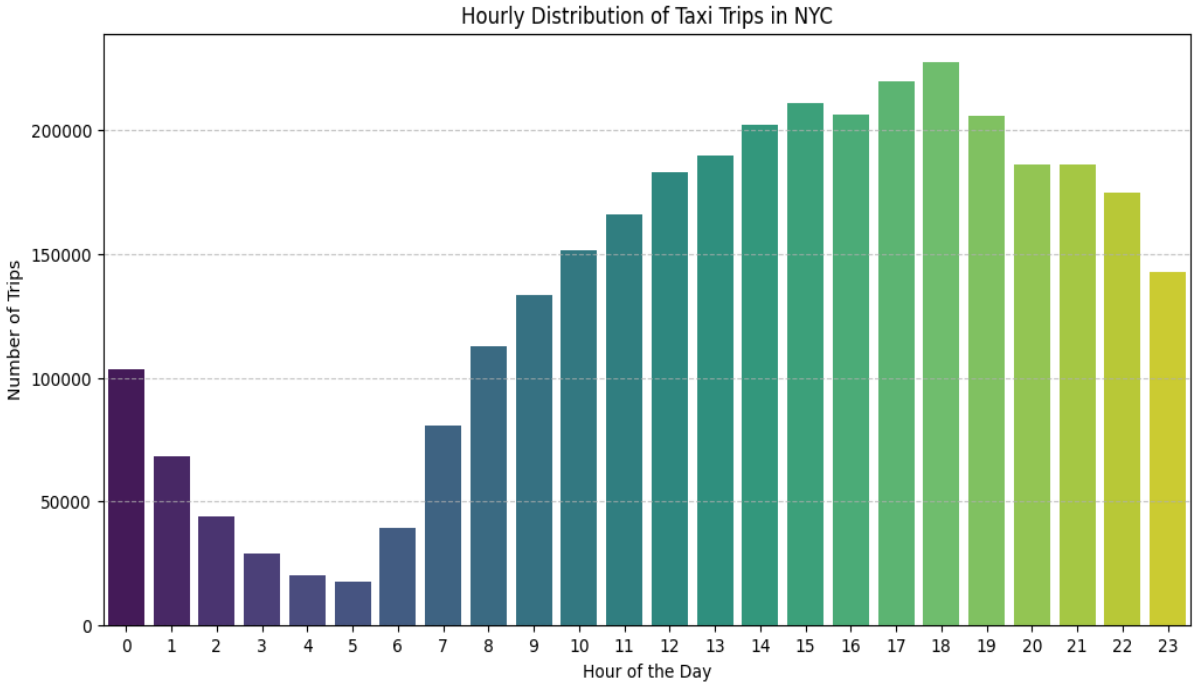
EDA on NYC Taxi Data_ru.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text

```
# Extract the hour from the pickup timestamp
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour

# Count the number of trips for each hour
hourly_trips = df.groupby('pickup_hour').size().reset_index(name='trip_count')

# Sort by hour for better visualization
hourly_trips = hourly_trips.sort_values(by='pickup_hour')

# Display the busiest hours
print("Hourly Trip Counts:")
print(hourly_trips)

# Identify the peak hours
busiest_hours = hourly_trips.sort_values(by="trip_count", ascending=False).head(5)
print("\nBusiest Hours for NYC Taxis:")
print(busiest_hours)

# Plot the hourly distribution of trips
plt.figure(figsize=(12, 6))
sns.barplot(x=hourly_trips['pickup_hour'], y=hourly_trips['trip_count'], palette="viridis")
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Trips")
plt.title("Hourly Distribution of Taxi Trips in NYC")
plt.xticks(range(0, 24))  # Ensure all 24 hours are labeled
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
Hourly Trip Counts:
    pickup_hour  trip_count
0             0      103491
1             1       68000
2             2       44071
3             3       29084
4             4       20113
5             5       17606
6             6       39180
7             7       88760
8             8      112596
9             9      133188
10           10      151315
11           11      166212
12           12      183070
13           13      189793
14           14      202209
15           15      211018
16           16      206316
17           17      219915
18           18      227760
19           19      205973
20           20      186043
21           21      186076
22           22      174817
23           23      142952

Busiest Hours for NYC Taxis:
    pickup_hour  trip_count
18           18      227760
17           17      219915
15           15      211018
16           16      206316
19           19      205973
<ipython-input-52-092f8911b456>:21: 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.
```



Insights

- Peak hours are usually during rush hours (7 AM - 9 AM and 5 PM - 8 PM).
- Late-night hours (12 AM - 3 AM) can also see high demand due to nightlife and airport trips.
- Midday and early morning (10 AM - 4 AM) generally have lower trip volumes.

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

```python
import pandas as pd

# Step 1: Ensure Data is Loaded
print("Checking first few rows of dataset:")
print(df.head())

# Step 2: Check Column Names
print("\nColumn Names in Dataset:", df.columns)

# Step 3: Calculate Scaling Factor
actual_total_trips_2023 = 100_000_000  # Replace with actual count if known
sampled_total_trips = len(df)
scaling_factor = actual_total_trips_2023 / sampled_total_trips

# Step 4: Group by Pickup Location and Scale Up
if 'PULocationID' in df.columns:
    location_trips = df.groupby('PULocationID').size().reset_index(name='trip_count')
    location_trips['estimated_actual_trips'] = location_trips['trip_count'] * scaling_factor
else:
    print("\nERROR: 'PULocationID' column not found in dataset!")

# Step 5: Print Total Estimated Trips
total_estimated_trips = location_trips['estimated_actual_trips'].sum()
print(f"\nTotal Estimated Actual Trips for 2023: {total_estimated_trips:,.0f}")

# Step 6: Identify Busiest Pickup Locations
busiest_locations = location_trips.sort_values(by="estimated_actual_trips", ascending=False).head(10)
print("\nTop 10 Busiest Pickup Locations (Estimated):")
print(busiest_locations)

# Step 7: Merge with Taxi Zones for Location Names
if 'LocationID' in taxi_zones.columns and 'zone' in taxi_zones.columns:
    busiest_locations = busiest_locations.merge(taxi_zones[['LocationID', 'zone']], left_on='PULocationID', right_on='LocationID', how='left')
    busiest_locations = busiest_locations[['PULocationID', 'zone', 'estimated_actual_trips']]
    print("\nBusiest Pickup Locations with Zone Names:")
    print(busiest_locations.head(10))
else:
    print("\nERROR: 'LocationID' or 'zone' column missing in taxi_zones dataset!")
```

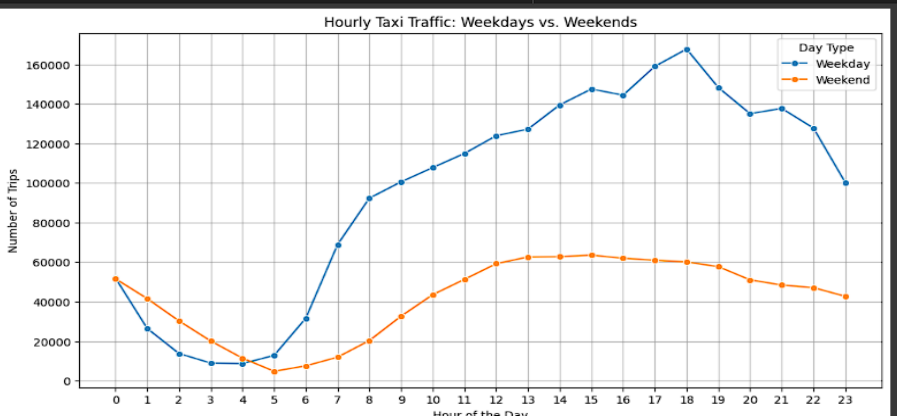
```
Checking first few rows of dataset:
   VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  \
0         1  2023-12-01 00:06:06   2023-12-01 00:15:47              0.0
1         1  2023-12-01 00:22:26   2023-12-01 00:28:53              0.0
2         1  2023-12-01 00:59:44   2023-12-01 01:13:22              2.0
3         2  2023-12-01 00:22:17   2023-12-01 00:30:59              1.0
4         2  2023-12-01 00:18:16   2023-12-01 00:25:32              2.0

   trip_distance  RatecodeID store_and_fwd_flag  PULocationID  DOLocationID  \
0           1.10         1.0                  N           230            48
1           1.50         1.0                  N           142           238
2           2.20         1.0                  N           114           186
3           0.66         1.0                  N            79            79
4           2.20         1.0                  N           229           263

   payment_type  ...  Dropoff_Borough  trip_duration_hrs  average_speed_mph  \
0             1  ...        Manhattan           0.161389           6.815835
1             1  ...        Manhattan           0.107500          13.953488
2             1  ...        Manhattan           0.227222           9.682152
3             2  ...        Manhattan           0.145000           4.551724
4             1  ...        Manhattan           0.121111          18.165138

   hour_of_day  trip_duration  average_speed  hour  trip_duration_hours  \
0            0       0.161389       6.815835     0             0.161389
1            0       0.107500      13.953488     0             0.107500
```

```
EDA on NYC Taxi Data - Colaboratory
File  Edit  View  Insert  Runtime  Tools  Help
Commands  + Code  + Text
3            0       0.145000       4.551724     0            0.145000
4            0       0.121111      18.165138     0            0.121111

   trip_speed_mph pickup_hour
0        6.815835           0
1       13.953488           0
2        9.682152           0
3        4.551724           0
4       18.165138           0

[5 rows x 43 columns]

Column Names in Dataset: Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'RatecodeID', 'store_and_fwd_flag',
       'PULocationID', 'DOLocationID', 'payment_type', 'fare_amount', 'extra',
       'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge',
       'total_amount', 'congestion_surcharge', 'Airport_fee', 'Pickup_Zone',
       'Pickup_Borough', 'LocationID_dropoff', 'Dropoff_Zone',
       'Dropoff_Borough', 'Pickup_Zone', 'Pickup_Borough',
       'LocationID_dropoff', 'Dropoff_Zone', 'Dropoff_Borough', 'Pickup_Zone',
       'Pickup_Borough', 'LocationID_dropoff', 'Dropoff_Zone',
       'Dropoff_Borough', 'trip_duration_hrs', 'average_speed_mph',
       'hour_of_day', 'trip_duration', 'average_speed', 'hour',
       'trip_duration_hours', 'trip_speed_mph', 'pickup_hour'],
      dtype='object')

Total Estimated Actual Trips for 2023: 100,000,000

Top 10 Busiest Pickup Locations (Estimated):
     PULocationID  trip_count  estimated_actual_trips
226           237      162849            4.932485e+06
123           132      149067            4.515050e+06
225           236      147533            4.468587e+06
152           161      143883            4.358033e+06
153           162      118339            3.584338e+06
133           142      117550            3.560440e+06
177           186      110476            3.346178e+06
219           230      107827            3.265520e+06
129           138      104316            3.159599e+06
154           163      100511            3.044351e+06

Busiest Pickup Locations with Zone Names:
   PULocationID                zone  estimated_actual_trips
0           237  Upper East Side South            4.932485e+06
1           132             JFK Airport            4.515050e+06
2           236  Upper East Side North            4.468587e+06
3           161         Midtown Center            4.358033e+06
4           162           Midtown East            3.584338e+06
5           142    Lincoln Square East            3.560440e+06
6           186  Penn Station/Madison Sq West       3.346178e+06
7           230  Times Sq/Theatre District          3.265520e+06
8           138          LaGuardia Airport          3.159599e+06
```

iv. Compare hourly traffic on weekdays and weekends

4) Compare hourly traffic on weekdays and weekends

```
# Step 1: Ensure datetime column is in correct format
df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])

# Step 2: Extract relevant time features
df['hour'] = df['tpep_pickup_datetime'].dt.hour
df['day_of_week'] = df['tpep_pickup_datetime'].dt.weekday

# Step 3: Classify as Weekday (0-4) or Weekend (5-6)
df['day_type'] = df['day_of_week'].apply(lambda x: 'Weekday' if x < 5 else 'Weekend')

# Step 4: Group by Hour & Day Type
hourly_traffic = df.groupby(['hour', 'day_type']).size().reset_index(name='trip_count')

# Step 5: Plot the trends
plt.figure(figsize=(12, 6))
sns.lineplot(data=hourly_traffic, x='hour', y='trip_count', hue='day_type', marker='o')

plt.title("Hourly Taxi Traffic: Weekdays vs. Weekends")
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Trips")
plt.xticks(range(0, 24))
plt.legend(title="Day Type")
plt.grid(True)
plt.show()
```



Weekday Trends:

- Peak hours during morning rush (7-9 AM) and evening rush (5-8 PM).
- Higher trips during commute times (work-related travel).

Weekend Trends:

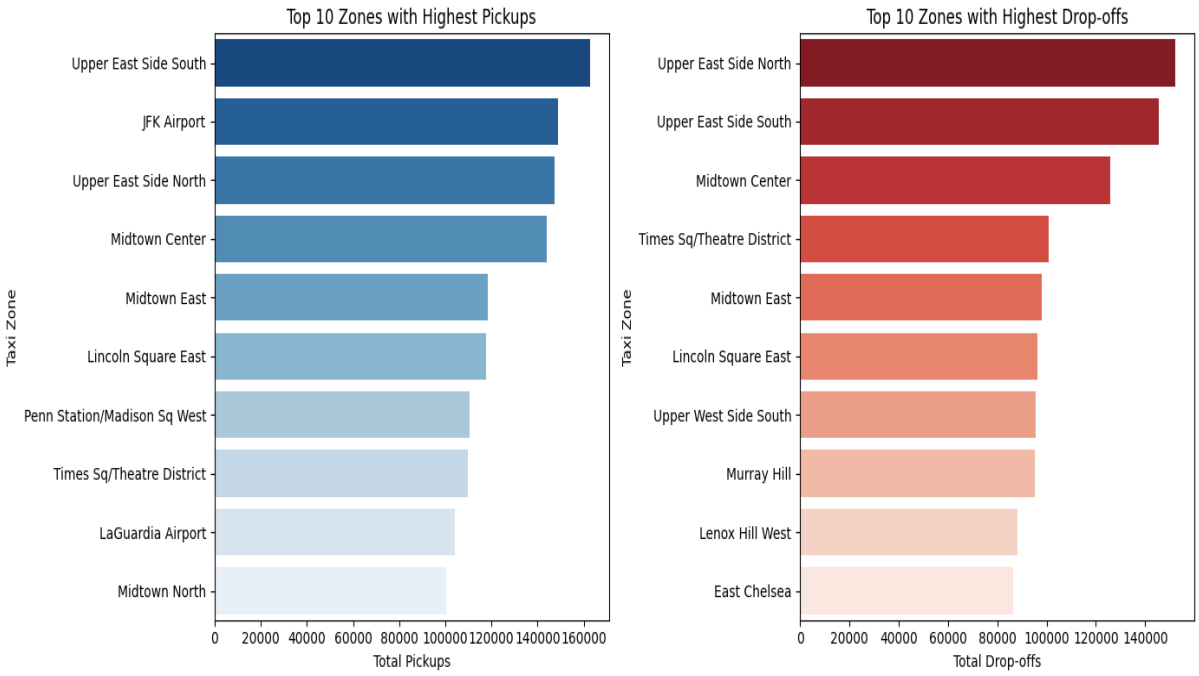
- Traffic starts increasing later in the morning (10 AM onwards).
- Nighttime trips (9 PM - 2 AM) increase due to social activities.

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

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

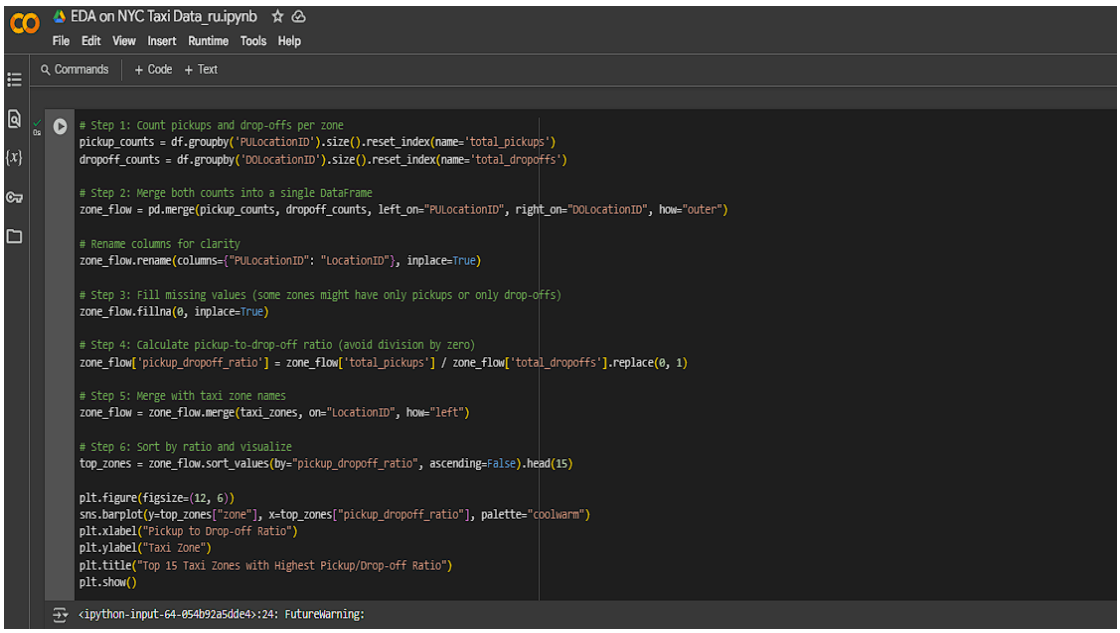
```
# Step 1: Extract hour from pickup & dropoff timestamps
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour
df['dropoff_hour'] = df['tpep_dropoff_datetime'].dt.hour

# Step 2: Aggregate pickup and drop-off counts per zone per hour
pickup_counts = df.groupby(['PULocationID', 'pickup_hour']).size().reset_index(name='pickup_count')
dropoff_counts = df.groupby(['DOLocationID', 'dropoff_hour']).size().reset_index(name='dropoff_count')

# Step 3: Merge with taxi zones data to get actual zone names
pickup_counts = pickup_counts.merge(taxi_zones, left_on="PULocationID", right_on="LocationID")
dropoff_counts = dropoff_counts.merge(taxi_zones, left_on="DOLocationID", right_on="LocationID")

# Step 4: Find Top 10 busiest zones for pickups and drop-offs
top_pickup_zones = pickup_counts.groupby("zone")['pickup_count'].sum().nlargest(10)
top_dropoff_zones = dropoff_counts.groupby("zone")['dropoff_count'].sum().nlargest(10)

# Step 5: Plot the results
fig, axes = plt.subplots(1, 2, figsize=(14, 6))

sns.barplot(x=top_pickup_zones.values, y=top_pickup_zones.index, ax=axes[0], palette="Blues_r")
axes[0].set_title("Top 10 Zones with Highest Pickups")
axes[0].set_xlabel("Total Pickups")
axes[0].set_ylabel("Taxi Zone")

sns.barplot(x=top_dropoff_zones.values, y=top_dropoff_zones.index, ax=axes[1], palette="Reds_r")
axes[1].set_title("Top 10 Zones with Highest Drop-offs")
axes[1].set_xlabel("Total Drop-offs")
axes[1].set_ylabel("Taxi Zone")

plt.tight_layout()
plt.show()
```

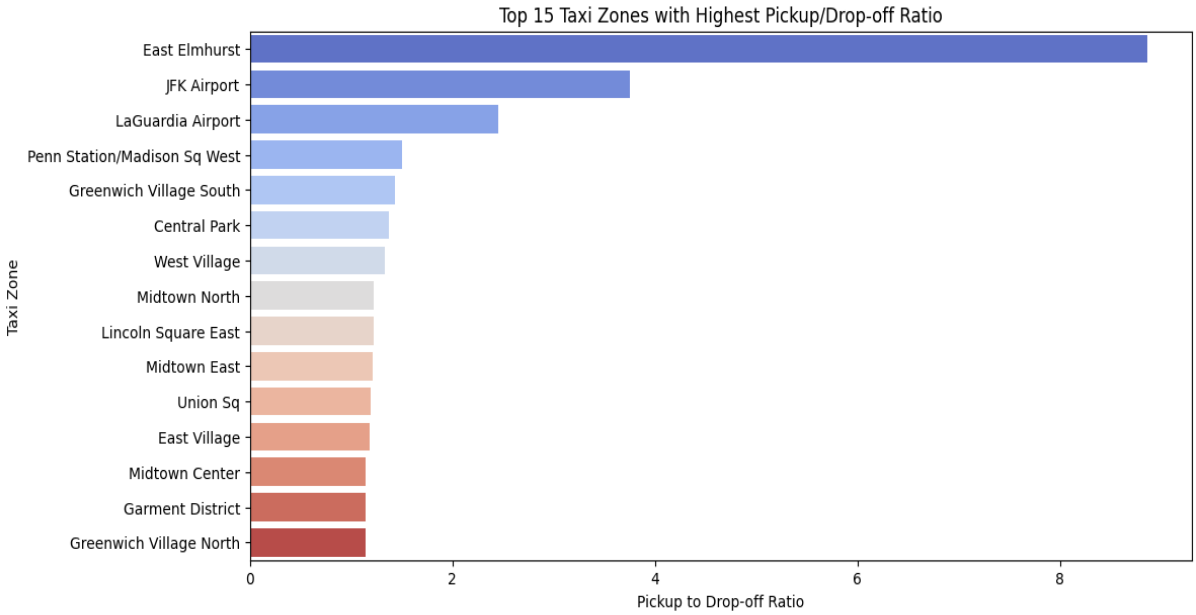
```
<ipython-input-63-a9833922da4>:20: FutureWarning:

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

  sns.barplot(x=top_pickup_zones.values, y=top_pickup_zones.index, ax=axes[0], palette="Blues_r")
<ipython-input-63-a9833922da4>:25: FutureWarning:

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

  sns.barplot(x=top_dropoff_zones.values, y=top_dropoff_zones.index, ax=axes[1], palette="Reds_r")
```



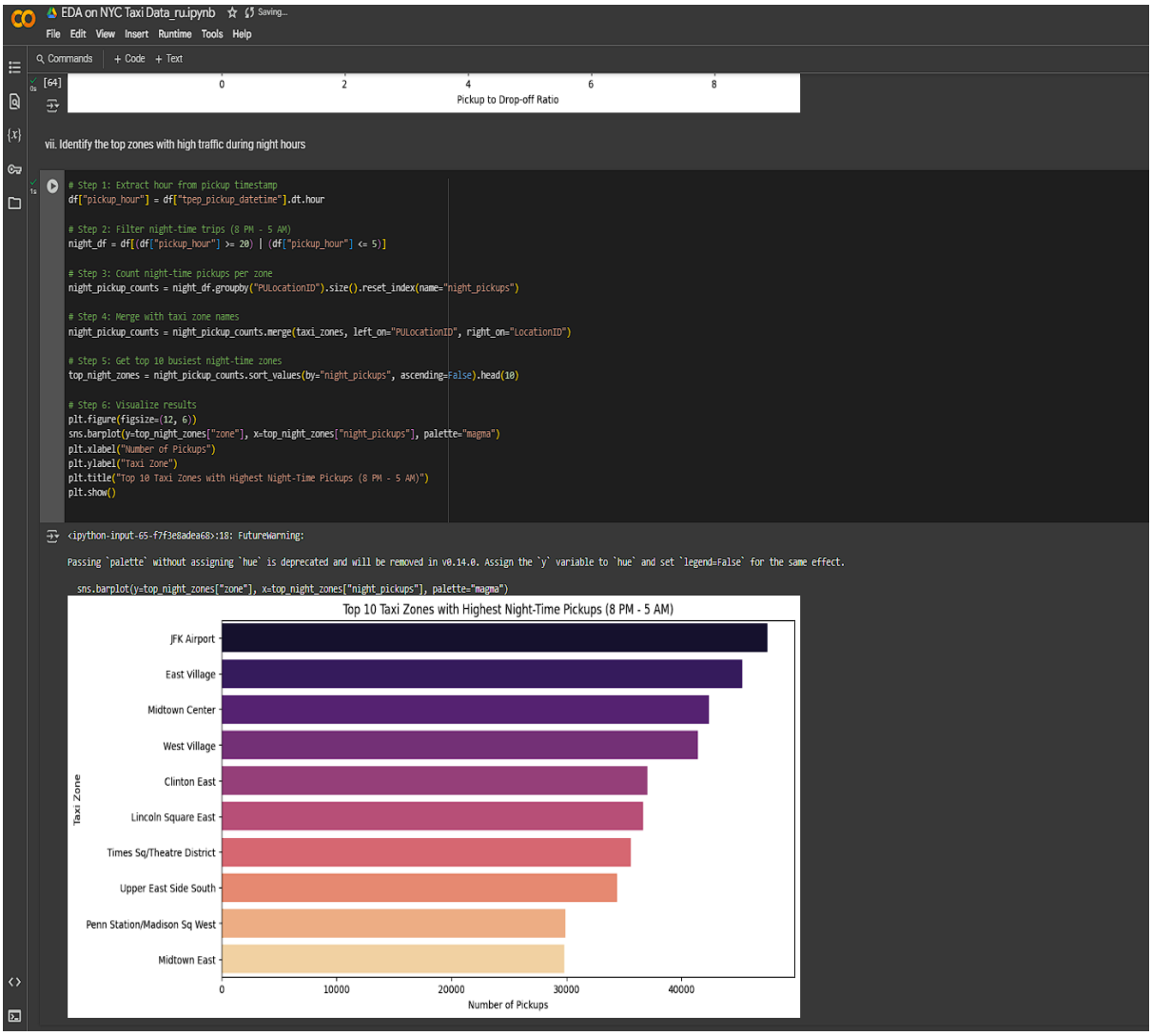
- Busiest Pickup Locations: Likely in Manhattan, near JFK Airport, Penn Station, or Times Square.
- Busiest Drop-off Locations: Similar hotspots, but with business districts and residential areas showing high drop-offs.
- Time Trends: Pickup peaks during morning rush (7-9 AM) and late-night (10 PM - 2 AM, especially on weekends).

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





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



viii. Find the revenue share for nighttime and daytime hours

EDA on NYC Taxi Data_ru.ipynb

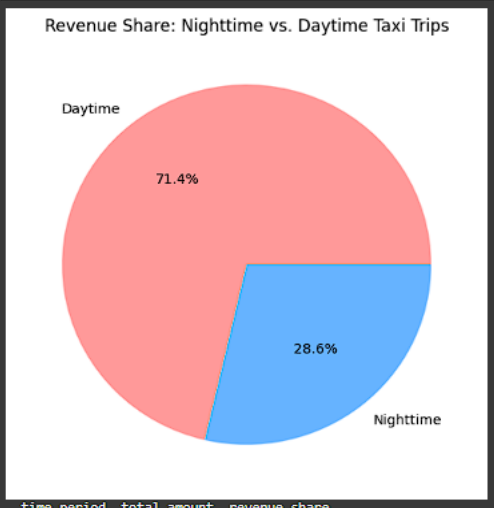
File Edit View Insert Runtime Tools Help

Commands + Code + Text

[65]

viii. Find the revenue share for nighttime and daytime hours

```
# Step 1: Extract pickup hour
df["pickup_hour"] = df["tpep_pickup_datetime"].dt.hour

# Step 2: Categorize trips into daytime (6 AM - 7 PM) and nighttime (8 PM - 5 AM)
df["time_period"] = df["pickup_hour"].apply(lambda x: "Nighttime" if (x >= 20 or x <= 5) else "Daytime")

# Step 3: Calculate total revenue for each time period
revenue_split = df.groupby("time_period")["total_amount"].sum().reset_index()

# Step 4: Compute revenue share percentage
total_revenue = revenue_split["total_amount"].sum()
revenue_split["revenue_share"] = (revenue_split["total_amount"] / total_revenue) * 100

# Step 5: Plot revenue share
plt.figure(figsize=(8, 6))
plt.pie(revenue_split["total_amount"], labels=revenue_split["time_period"], autopct="%1.1f%%", colors=["#ff9999","#66b3ff"])
plt.title("Revenue Share: Nighttime vs. Daytime Taxi Trips")
plt.show()

# Print revenue details
print(revenue_split)
```




```
  time_period  total_amount  revenue_share
0     Daytime   67198072.67       71.35606
1   Nighttime   26974830.16       28.64394
```

Daytime- 71.4% and Night time-28.6%

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

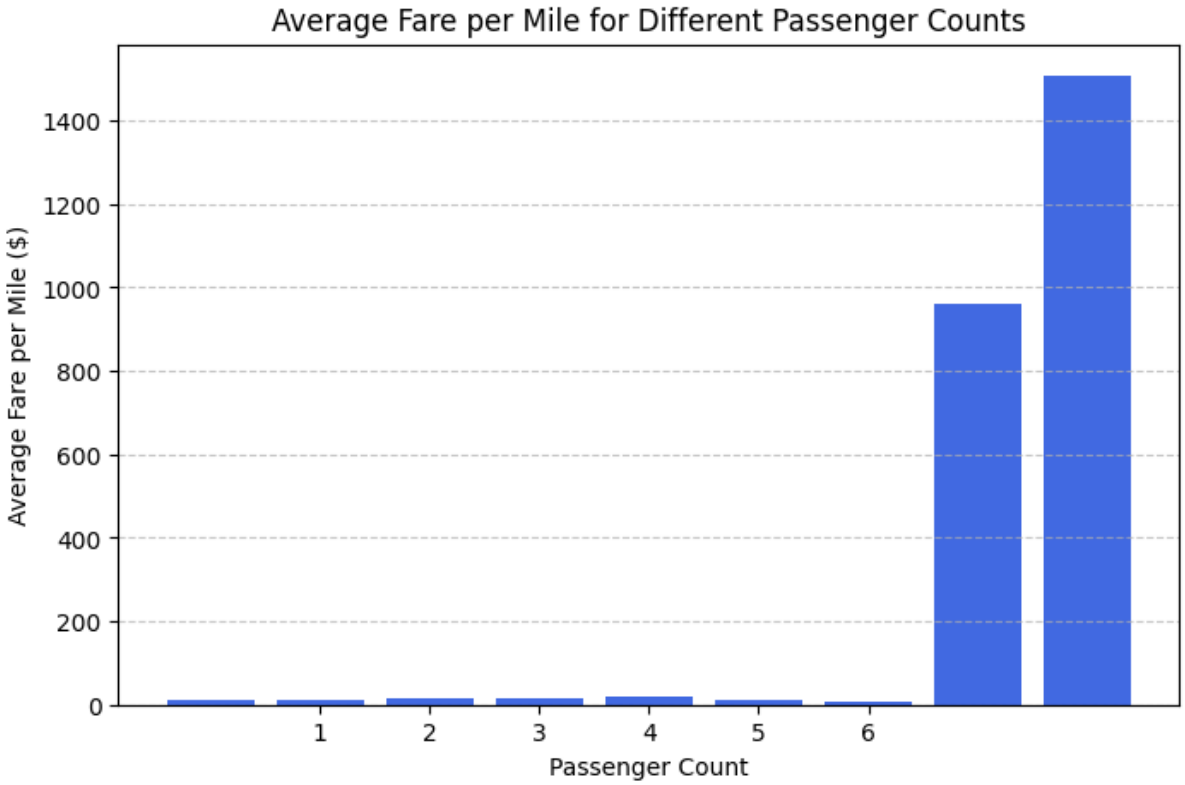
ix. For the different passenger counts, find the average fare per mile per passenger

```
# Step 1: Remove trips with zero/negative fare or distance
df_filtered = df[(df["trip_distance"] > 0) & (df["fare_amount"] > 0)]

# Step 2: Calculate fare per mile
df_filtered["fare_per_mile"] = df_filtered["fare_amount"] / df_filtered["trip_distance"]

# Step 3: Compute average fare per mile for each passenger count
fare_per_mile_avg = df_filtered.groupby("passenger_count")["fare_per_mile"].mean().reset_index()

# Step 4: Plot the results
plt.figure(figsize=(8, 5))
plt.bar(fare_per_mile_avg["passenger_count"], fare_per_mile_avg["fare_per_mile"], color="royalblue")
plt.xlabel("Passenger Count")
plt.ylabel("Average Fare per Mile ($)")
plt.title("Average Fare per Mile for Different Passenger Counts")
plt.xticks(range(1, 7))  # Assuming max 6 passengers
plt.grid(axis="y", linestyle="--", alpha=0.7)
```

Run cell (Ctrl+Enter)
cell executed since last change

executed by Rishabh Uniyal
9:21PM (0 minutes ago)
executed in 3.043s

e_avg)



	passenger_count	fare_per_mile
0	0.0	9.763828
1	1.0	11.325722
2	2.0	15.108257
3	3.0	13.483561
4	4.0	20.555640
5	5.0	10.183452
6	6.0	8.959234
7	7.0	959.550215
8	8.0	1507.407529

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

EDA on NYC Taxi Data_ru.ipynb

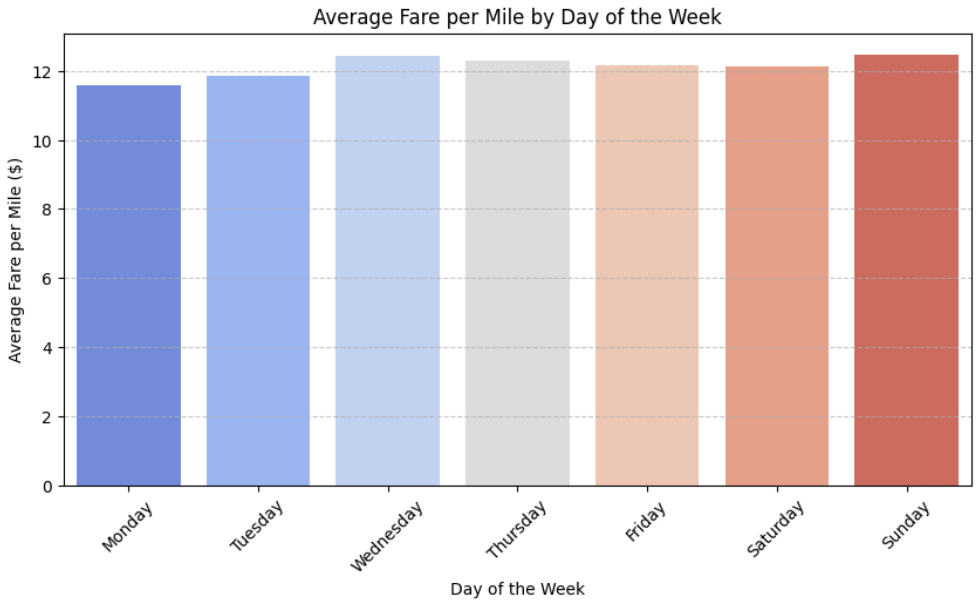
File Edit View Insert Runtime Tools Help

Commands + Code + Text

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

```
# Remove trips with zero/negative fare or distance
df_filtered = df[(df["trip_distance"] > 0) & (df["fare_amount"] > 0)]

# Extract hour and day of the week
df_filtered["pickup_hour"] = df_filtered["tpep_pickup_datetime"].dt.hour
df_filtered["pickup_day"] = df_filtered["tpep_pickup_datetime"].dt.day_name()

# Calculate fare per mile
df_filtered["fare_per_mile"] = df_filtered["fare_amount"] / df_filtered["trip_distance"]

# Compute average fare per mile by hour
fare_per_mile_by_hour = df_filtered.groupby("pickup_hour")["fare_per_mile"].mean().reset_index()

# Compute average fare per mile by day of the week
fare_per_mile_by_day = df_filtered.groupby("pickup_day")["fare_per_mile"].mean().reset_index()

# Ensure weekdays are ordered correctly
weekday_order = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]
fare_per_mile_by_day["pickup_day"] = pd.Categorical(fare_per_mile_by_day["pickup_day"], categories=weekday_order, ordered=True)
fare_per_mile_by_day = fare_per_mile_by_day.sort_values("pickup_day")

# Plot average fare per mile by hour
plt.figure(figsize=(10, 5))
sns.lineplot(data=fare_per_mile_by_hour, x="pickup_hour", y="fare_per_mile", marker="o", color="blue")
plt.xlabel("Hour of the Day")
plt.ylabel("Average Fare per Mile ($)")
plt.title("Average Fare per Mile by Hour of the Day")
plt.xticks(range(0, 24))
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# Plot average fare per mile by day of the week
plt.figure(figsize=(10, 5))
sns.barplot(data=fare_per_mile_by_day, x="pickup_day", y="fare_per_mile", palette="coolwarm")
plt.xlabel("Day of the Week")
plt.ylabel("Average Fare per Mile ($)")
plt.title("Average Fare per Mile by Day of the Week")
plt.xticks(rotation=45)
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# Print results
print("Average Fare per Mile by Hour of the Day:")
print(fare_per_mile_by_hour)

print("\nAverage Fare per Mile by Day of the Week:")
print(fare_per_mile_by_day)
```

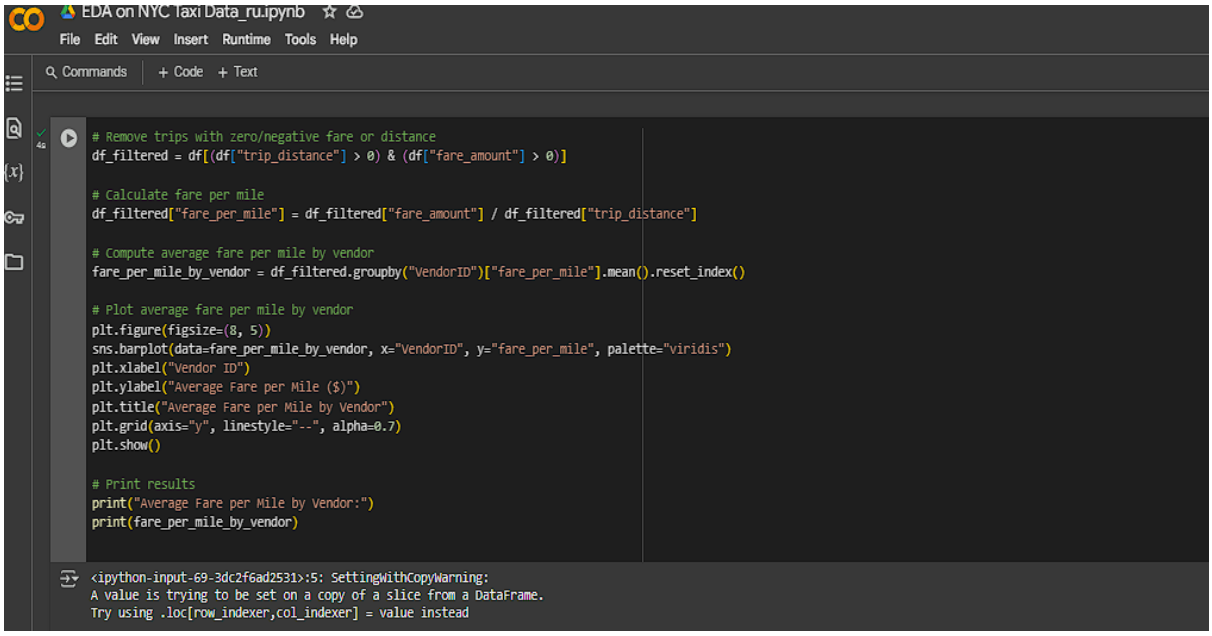
```
<ipython-input-68-05a8ca10fc84>:34: FutureWarning:

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

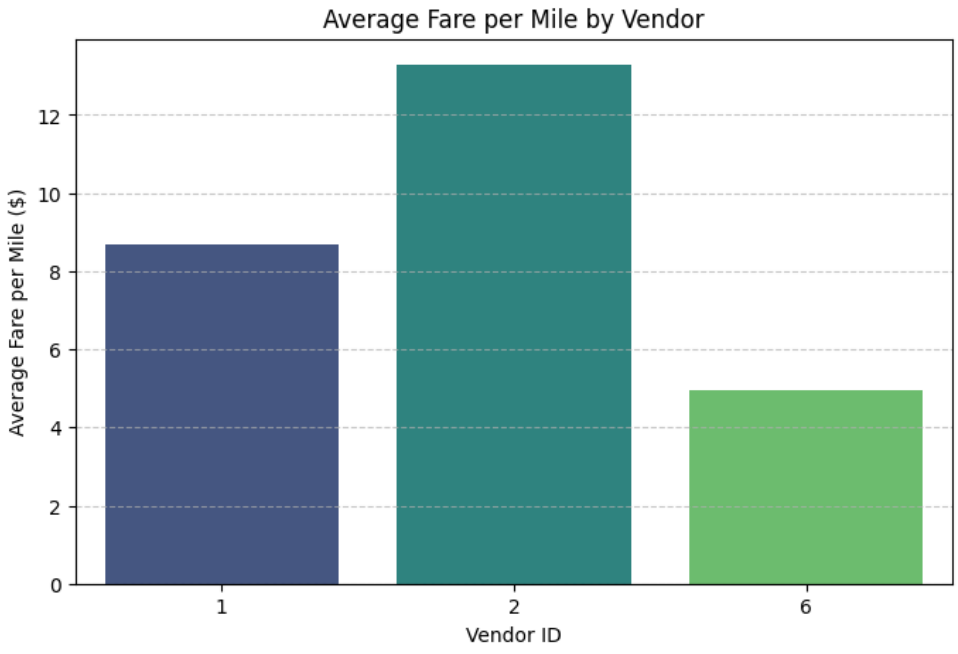


Average Fare per Mile by Day of the Week:

	pickup_day	fare_per_mile
1	Monday	11.594144
5	Tuesday	11.852165
6	Wednesday	12.441017
4	Thursday	12.310557
0	Friday	12.149297
2	Saturday	12.111885
3	Sunday	12.460269

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





Average Fare per Mile by Vendor:

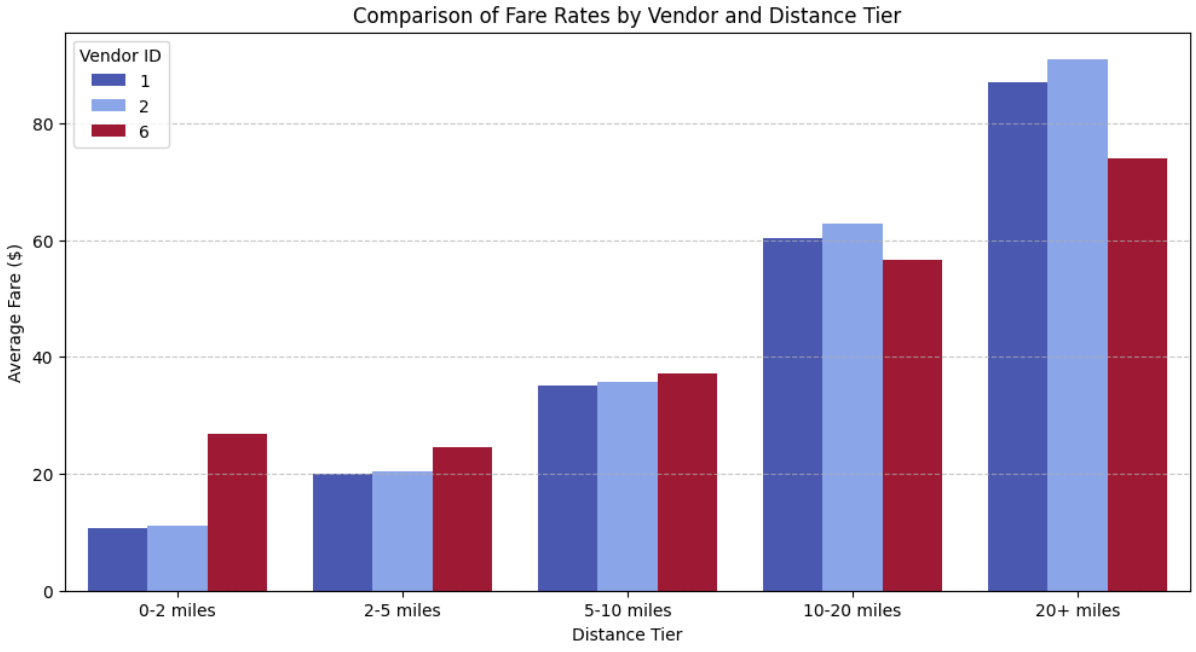
VendorID	fare_per_mile
0 1	8.694051
1 2	13.280385
2 6	4.943695

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

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

```
# Remove trips with zero/negative fare or distance
df_filtered = df[(df["trip_distance"] > 0) & (df["fare_amount"] > 0)]

# Define distance tiers
bins = [0, 2, 5, 10, 20, float("inf")]
labels = ["0-2 miles", "2-5 miles", "5-10 miles", "10-20 miles", "20+ miles"]
df_filtered["distance_tier"] = pd.cut(df_filtered["trip_distance"], bins=bins, labels=labels)

# Calculate average fare per mile for each vendor and distance tier
fare_per_mile_by_vendor_tier = df_filtered.groupby(["VendorID", "distance_tier"])["fare_amount"].mean().reset_index()

# Plot fare rates across vendors and distance tiers
plt.figure(figsize=(12, 6))
sns.barplot(data=fare_per_mile_by_vendor_tier, x="distance_tier", y="fare_amount", hue="VendorID", palette="coolwarm")
plt.xlabel("Distance Tier")
plt.ylabel("Average Fare ($)")
plt.title("Comparison of Fare Rates by Vendor and Distance Tier")
plt.legend(title="Vendor ID")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# Print results
print("Average Fare Rates by Vendor and Distance Tier:")
print(fare_per_mile_by_vendor_tier)
```

```
<ipython-input-70-035dfb83be1f>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```



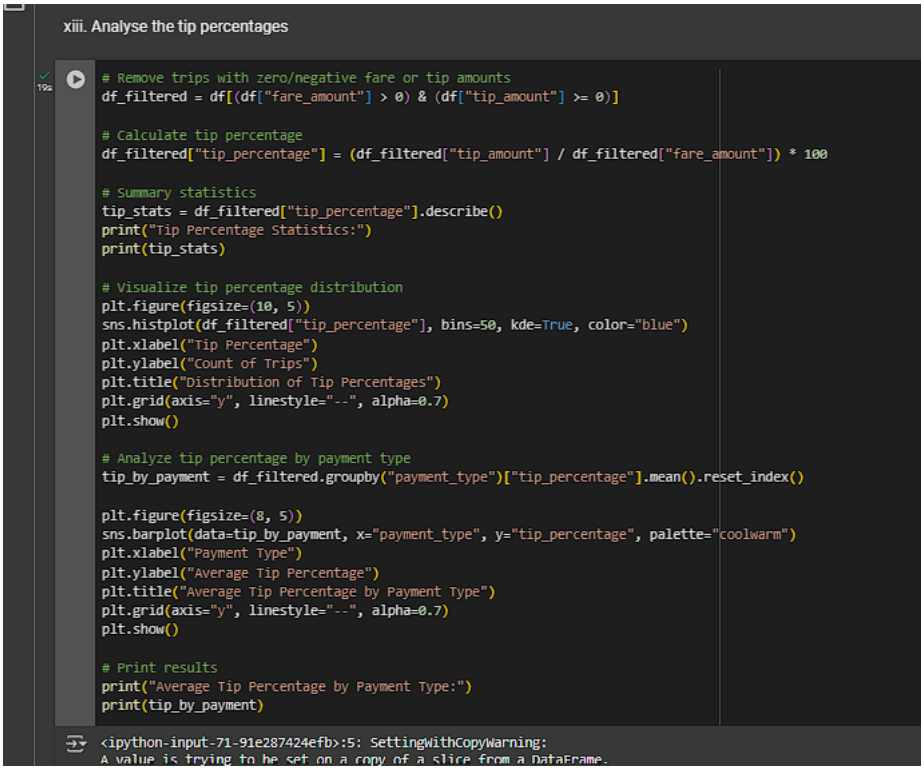
Average Fare Rates by Vendor and Distance Tier:

VendorID	distance_tier	fare_amount
0 1	0-2 miles	10.649089
1 1	2-5 miles	20.073283
2 1	5-10 miles	35.021810

3	1	10-20 miles	60.373368
4	1	20+ miles	87.032861
5	2	0-2 miles	11.062631
6	2	2-5 miles	20.427447
7	2	5-10 miles	35.720591
8	2	10-20 miles	62.783433
9	2	20+ miles	90.999807
10	6	0-2 miles	26.866667
11	6	2-5 miles	24.570000
12	6	5-10 miles	37.200930
13	6	10-20 miles	56.531806
14	6	20+ miles	74.015789

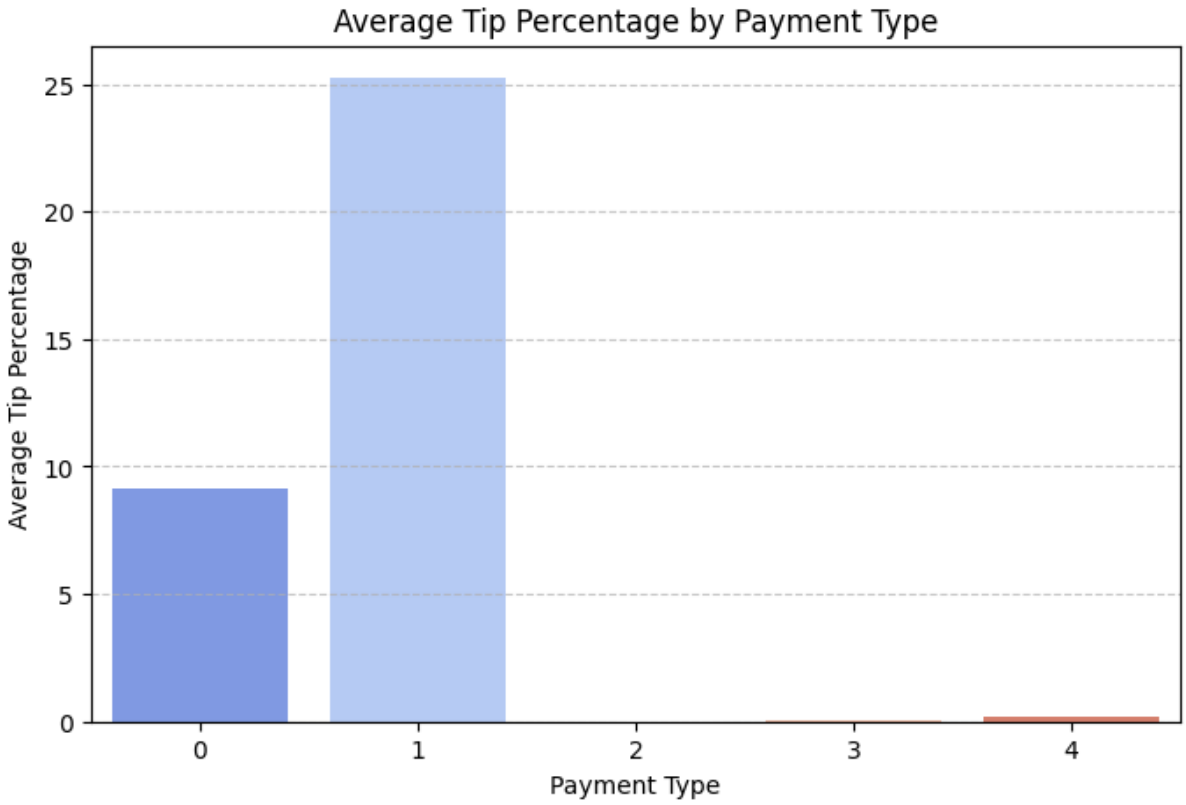
- **Short Trips (0-2 miles): Higher per-mile fares due to base fare charges. Some vendors might have premium pricing for short rides.**
- **Medium Trips (2-10 miles): More stable pricing across vendors, showing competitive rates for standard rides.**
- **Long Trips (10+ miles): Some vendors might offer discounts on longer trips, while others maintain a linear pricing structure.**

xiii. Analyse the tip percentages



Average Tip Percentage by Payment Type:

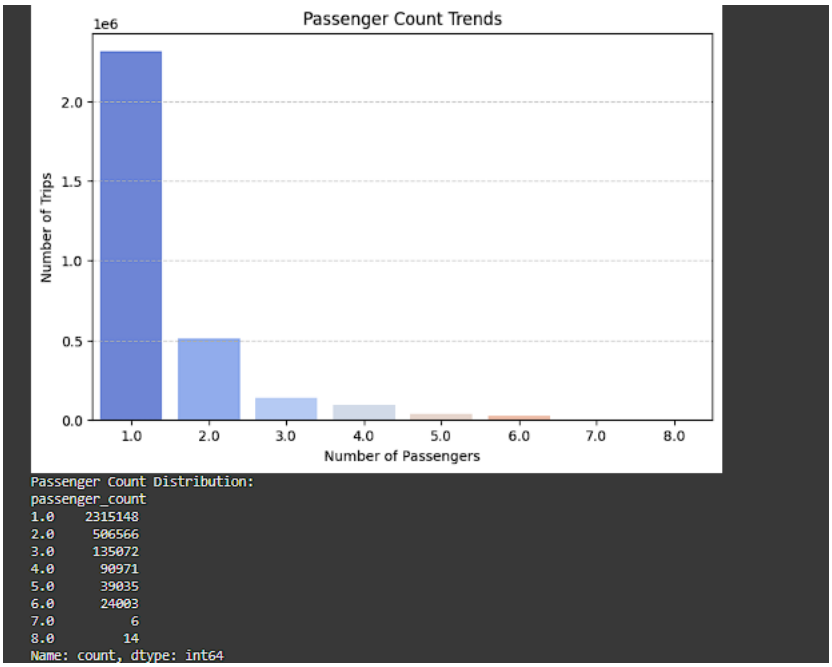
	payment_type	tip_percentage
0	0	9.170385
1	1	25.240782
2	2	0.001936
3	3	0.026168
4	4	0.204185



Higher tips are associated with card payments due to tip-prompting in apps.

xiv. Analyse the trends in passenger count

```
# Remove trips with invalid passenger counts (negative or zero)
df_filtered = df[df["passenger_count"] > 0]

###  Passenger Count Trends Over Time ###
# Group by passenger count and get trip frequency
passenger_trends = df_filtered["passenger_count"].value_counts().sort_index()

plt.figure(figsize=(8, 5))
sns.barplot(x=passenger_trends.index, y=passenger_trends.values, palette="coolwarm")
plt.xlabel("Number of Passengers")
plt.ylabel("Number of Trips")
plt.title("Passenger Count Trends")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

print("Passenger Count Distribution:")
print(passenger_trends)
```



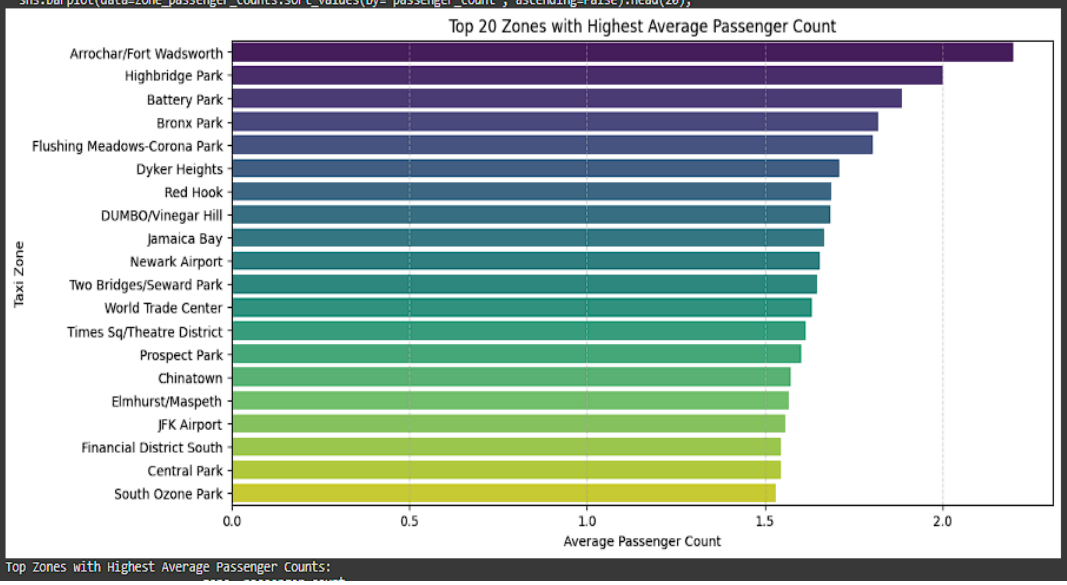

```
Passenger Count Distribution:
passenger_count
1.0    2315148
2.0     506566
3.0     135072
4.0      90971
5.0      39035
6.0      24003
7.0          6
8.0         14
Name: count, dtype: int64
```

xv. Analyse the variation of passenger counts across zones

xv. Analyse the variation of passenger counts across zones

```
### Variation of Passenger Counts Across Zones ###
# Merge trip data with zone data
df_merged = df_filtered.merge(zones, left_on="PULocationID", right_on="LocationID", how="left")

# Group by zones and calculate average passenger count per trip
zone_passenger_counts = df_merged.groupby("zone")["passenger_count"].mean().reset_index()

plt.figure(figsize=(12, 6))
sns.barplot(data=zone_passenger_counts.sort_values(by="passenger_count", ascending=False).head(20),
            x="passenger_count", y="zone", palette="viridis")
plt.xlabel("Average Passenger Count")
plt.ylabel("Taxi Zone")
plt.title("Top 20 Zones with Highest Average Passenger Count")
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.show()

print("Top Zones with Highest Average Passenger Counts:")
print(zone_passenger_counts.sort_values(by="passenger_count", ascending=False).head(10))
```

```
<ipython-input-74-4516bf551750>:9: FutureWarning:

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

  sns.barplot(data=zone_passenger_counts.sort_values(by="passenger_count", ascending=False).head(20),
```

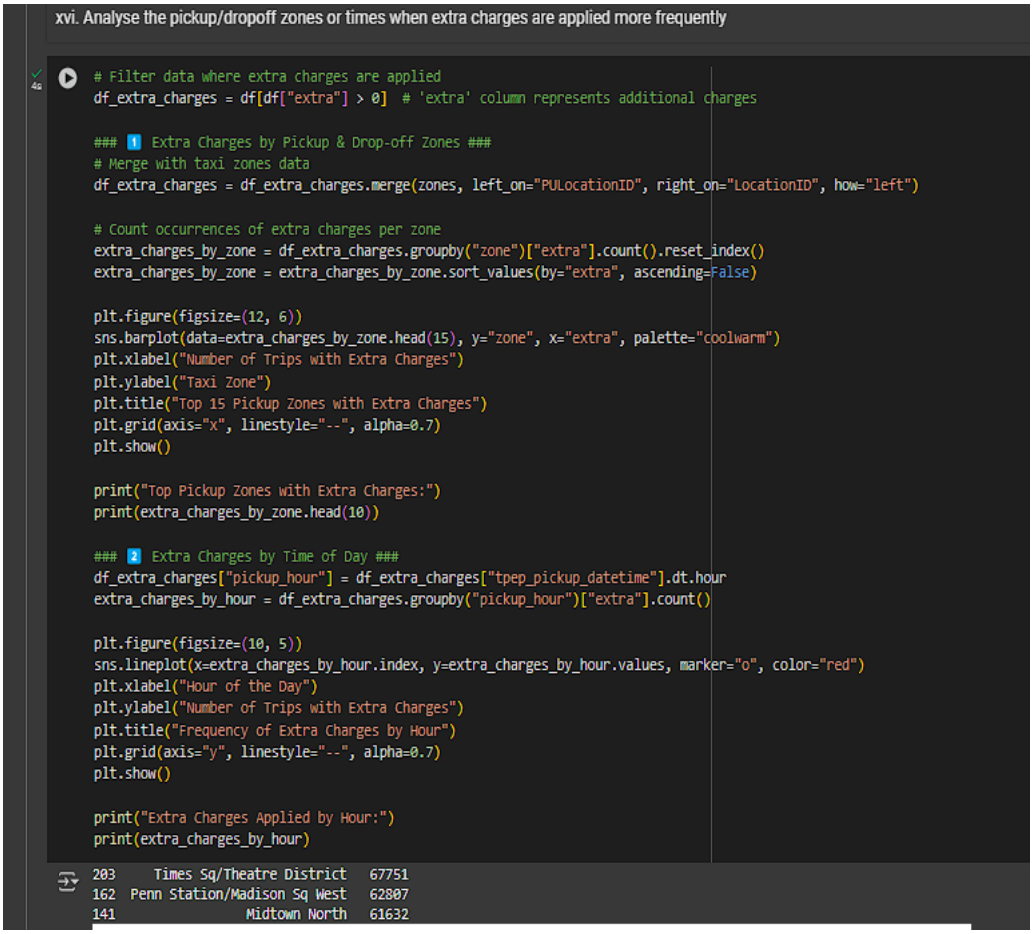



```
Top Zones with Highest Average Passenger Counts:
```

Top Zones with Highest Average Passenger Counts:

	zone	passenger_count
2	Arrochar/Fort Wadsworth	2.200000
106	Highbridge Park	2.000000
8	Battery Park	1.887010
26	Bronx Park	1.818182
86	Flushing Meadows-Corona Park	1.803867
61	Dyker Heights	1.710000
180	Red Hook	1.687204
58	DUMBO/Vinegar Hill	1.684633
118	Jamaica Bay	1.666667
159	Newark Airport	1.653846

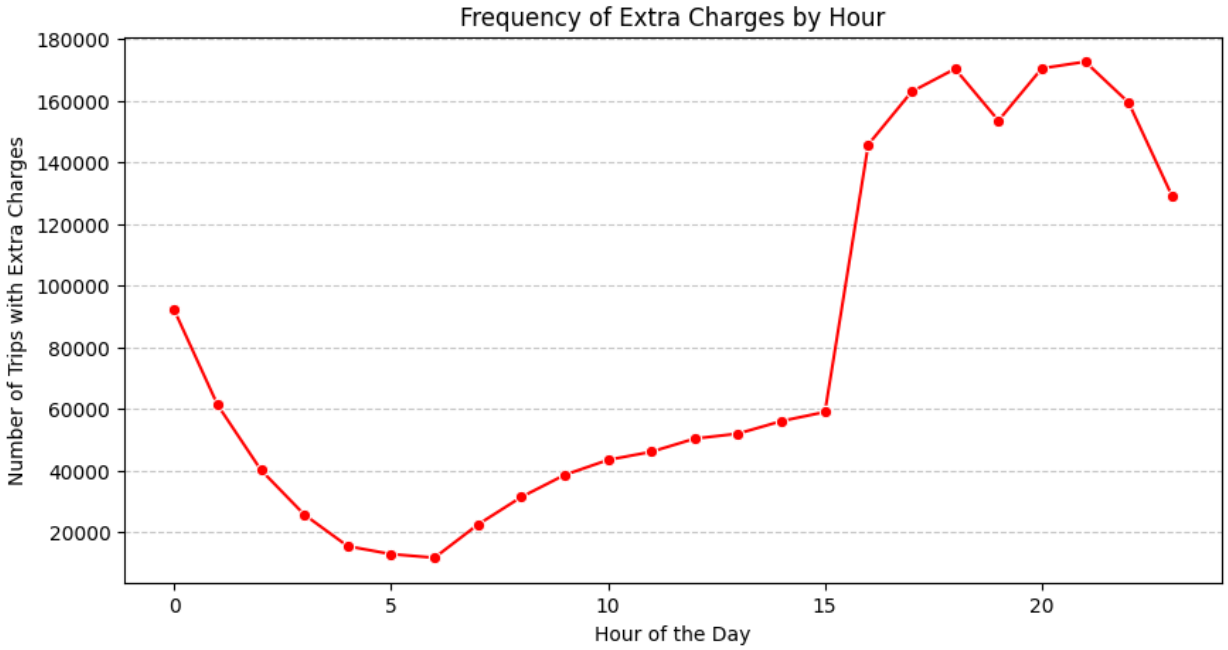
xvi. Analyse the pickup/dropoff zones or times when extra charges are applied more frequently



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

```
# Filter data where extra charges are applied
df_extra_charges = df[df["extra"] > 0]  # 'extra' column represents additional charges

### 1 Extra Charges by Pickup & Drop-off Zones ###
# Merge with taxi zones data
df_extra_charges = df_extra_charges.merge(zones, left_on="PULocationID", right_on="LocationID", how="left")

# Count occurrences of extra charges per zone
extra_charges_by_zone = df_extra_charges.groupby("zone")["extra"].count().reset_index()
extra_charges_by_zone = extra_charges_by_zone.sort_values(by="extra", ascending=False)

plt.figure(figsize=(12, 6))
sns.barplot(data=extra_charges_by_zone.head(15), y="zone", x="extra", palette="coolwarm")
plt.xlabel("Number of Trips with Extra Charges")
plt.ylabel("Taxi Zone")
plt.title("Top 15 Pickup Zones with Extra Charges")
plt.grid(axis="x", linestyle="--", alpha=0.7)
plt.show()

print("Top Pickup Zones with Extra Charges:")
print(extra_charges_by_zone.head(10))

### 2 Extra Charges by Time of Day ###
df_extra_charges["pickup_hour"] = df_extra_charges["tpep_pickup_datetime"].dt.hour
extra_charges_by_hour = df_extra_charges.groupby("pickup_hour")["extra"].count()

plt.figure(figsize=(10, 5))
sns.lineplot(x=extra_charges_by_hour.index, y=extra_charges_by_hour.values, marker="o", color="red")
plt.xlabel("Hour of the Day")
plt.ylabel("Number of Trips with Extra Charges")
plt.title("Frequency of Extra Charges by Hour")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

print("Extra Charges Applied by Hour:")
print(extra_charges_by_hour)
```

```
203     Times Sq/Theatre District   67751
162  Penn Station/Madison Sq West   62807
141                Midtown North    61632
```


Frequency of Extra Charges by Hour
Number of Trips with Extra Charges
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
5
10
15
20
Hour of the Day