

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>

```
EDA on NYC Taxi Data_ru.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text

import pandas as pd
import requests
import pyarrow.parquet as pq
from io import BytesIO

# Base URL for NYC TLC data
base_url = "https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-XX.parquet"

# Define the sampling fraction (e.g., 10%)
sample_fraction = 0.1

# List to store sampled DataFrames
sampled_dfs = []

# Loop through all 12 months
for month in range(1, 13):
    month_str = str(month).zfill(2) # Ensure two-digit month format (e.g., 01, 02, ..., 12)
    file_url = base_url.replace("XX", month_str) # Replace XX with actual month number
    print(f"Downloading: {file_url}")

    # Download the Parquet file using requests
    response = requests.get(file_url)

    if response.status_code == 200:
        df = pd.read_parquet(BytesIO(response.content), engine="pyarrow") # Read file into DataFrame

        # Sample 10% of the data
        sampled_df = df.sample(frac=sample_fraction, random_state=42)
        sampled_dfs.append(sampled_df)
    else:
        print(f"✗ Failed to download: {file_url}")

# Combine all sampled DataFrames
combined_df = pd.concat(sampled_dfs, ignore_index=True)

# Save the merged dataset to Google Colab storage
combined_df.to_parquet("/content/nyc_taxi_sampled_2023.parquet")
combined_df.to_csv("/content/nyc_taxi_sampled_2023.csv", index=False)

print("\n✓ Data Sampling Completed! Files saved in Colab:")
print("- /content/nyc_taxi_sampled_2023.parquet")
print("- /content/nyc_taxi_sampled_2023.csv")

Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-01.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-02.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-03.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-04.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-05.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-06.parquet
```

```

if response.status_code == 200:
    df = pd.read_parquet(BytesIO(response.content), engine="pyarrow") # Read file into DataFrame

    # Sample 10% of the data
    sampled_df = df.sample(frac=sample_fraction, random_state=42)
    sampled_dfs.append(sampled_df)
else:
    print(f"✗ Failed to download: {file_url}")

# Combine all sampled DataFrames
combined_df = pd.concat(sampled_dfs, ignore_index=True)

# Save the merged dataset to Google Colab storage
combined_df.to_parquet("/content/nyc_taxi_sampled_2023.parquet")
combined_df.to_csv("/content/nyc_taxi_sampled_2023.csv", index=False)

print("\n✓ Data Sampling Completed! Files saved in colab:")
print("- /content/nyc_taxi_sampled_2023.parquet")
print("- /content/nyc_taxi_sampled_2023.csv")

Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-01.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-02.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-03.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-04.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-05.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-06.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-07.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-08.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-09.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-10.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-11.parquet
Downloading: https://d37ci6vzurychx.cloudfront.net/trip-data/yellow_tripdata_2023-12.parquet

✓ Data Sampling Completed! Files saved in Colab:
- /content/nyc_taxi_sampled_2023.parquet
- /content/nyc_taxi_sampled_2023.csv

```

Now, merged dataset is prepared.

2: Data Cleaning:

a) Fixing Columns

```

# Standardize column names
combined_df.columns = combined_df.columns.str.strip() # remove any extra spaces
            .str.lower() # convert to lowercase
            .str.replace("_", " ") # Replace spaces with underscores
            .str.replace(r"[^\w\d\-.]", "", regex=True) # remove special characters
)

print("Column names standardized:")
print(combined_df.columns)

Column names standardized:
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'ratecodeid', 'store_and_fwd_flag',
       'pickuplocationid', 'dropofflocationid', 'payment_type', 'fare_amount', 'extra',
       'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge',
       'total_amount', 'congestion_surcharge', 'airport_fee', 'airport_fee'],
      dtype='object')

```

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

b) Handling Missing Values:

```

[9] In [9]: columns names standardized:
Index(['vendorid', 'trip_pickup_datetime', 'trip_dropoff_datetime',
       'passenger_count', 'trip_distance', 'ratecodeid', 'store_and_fwd_flag',
       'pickuplocationid', 'dolocationid', 'payment_type', 'fare_amount', 'extra',
       'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge',
       'total_amount', 'congestion_surcharge', 'airport_fee', 'airport_fee'],
      dtype='object')

[10] In [10]: # Check for missing values
missing_values = combined_df.isnull().sum()
missing_percentage = (missing_values / len(combined_df)) * 100

# Display missing values
missing_df = pd.DataFrame({'Missing Values': missing_values, 'Percentage': missing_percentage})
print("Missing Value Analysis:")
print(missing_df[missing_df['Missing Values'] > 0])

Missing Value Analysis:
   Missing Values    Percentage
passenger_count          130737     3.412586
ratecodeid                 130737     3.412586
store_and_fwd_flag          130737     3.412586
congestion_surcharge        130737     3.412586
airport_fee                3531449  92.188029
airport_fee                430012    11.232297

```

c) Handling Outliers

```

[11] In [11]: import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Define numerical columns for outlier detection
num_cols = ["fare_amount", "tip_amount", "passenger_count", "trip_distance", "total_amount"]

# Boxplot to detect outliers visually
plt.figure(figsize=(12, 5))
sns.boxplot(data=combined_df[num_cols])
plt.xticks(rotation=45)
plt.title("Boxplot for outlier detection")
plt.show()

# Using Interquartile Range (IQR) method
Q1 = combined_df[num_cols].quantile(0.25)
Q3 = combined_df[num_cols].quantile(0.75)
IQR = Q3 - Q1

# Define lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outlier counts
outliers = ((combined_df[num_cols] < lower_bound) | (combined_df[num_cols] > upper_bound)).sum()
print("Outlier detection results:")
print(outliers)

```

3. Exploratory Data Analysis [90 marks]

(a) General EDA: Finding Patterns

i. Classify variables into categorical and numerical

```

[11] In [11]: total_amount
Out[11]: 478490
dtype: int64

[12] In [12]: pip install geopandas
Requirement already satisfied: geopandas in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.11/dist-packages (from geopandas) (1.26.4)
Requirement already satisfied: pyproj<2.7.2 in /usr/local/lib/python3.11/dist-packages (from geopandas) (0.10.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from geopandas) (24.2)
Requirement already satisfied: pandas>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (3.2.2)
Requirement already satisfied: pytz>=2023.1 in /usr/local/lib/python3.11/dist-packages (from geopandas) (3.7.1)
Requirement already satisfied: shapely>=2.0.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.4.7)
Requirement already satisfied: six>=1.15.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (1.17.0)
Requirement already satisfied: python>=3.6.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (3.6.9)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>1.4.0->geopandas) (2025.1)
Requirement already satisfied: fiona>=2.0.7 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2025.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from pyproj>0.1.2->geopandas) (2025.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>2.8.2->pandas>1.4.0->geopandas) (1.17.0)

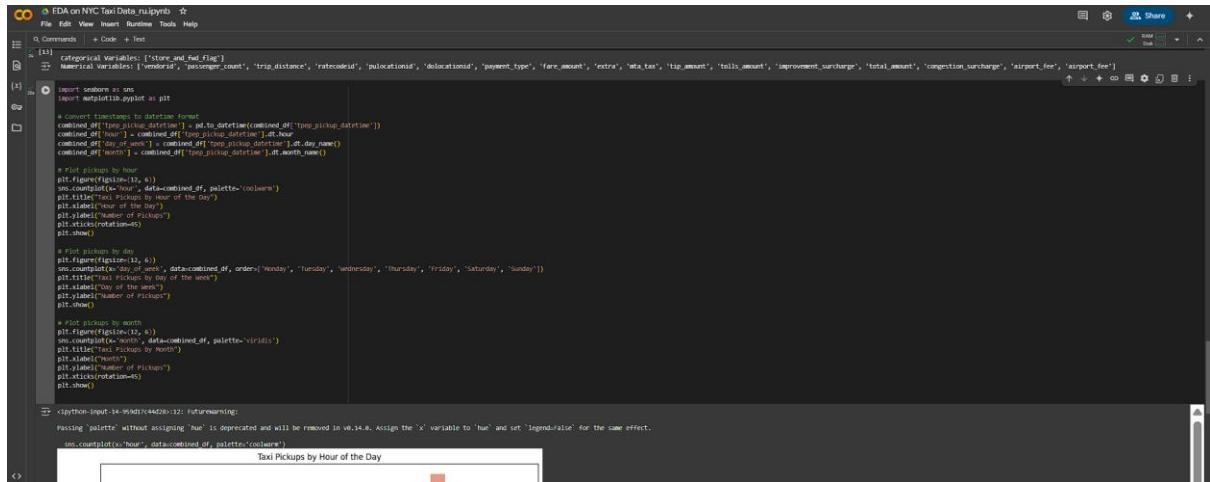
[13] In [13]: # Identify categorical and numerical columns
categorical_cols = combined_df.select_dtypes(include=['object', 'category']).columns.tolist()
numerical_cols = combined_df.select_dtypes(include=['int64', 'float64']).columns.tolist()

print("Categorical Variables:", categorical_cols)
print("Numerical Variables:", numerical_cols)

Categorical Variables: ['store_and_fwd_flag']
Numerical Variables: ['vendorid', 'passenger_count', 'trip_distance', 'ratecodeid', 'pickuplocationid', 'dolocationid', 'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge', 'total_amount', 'congestion_surcharge', 'airport_fee', 'airport_fee']

```

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



```

File Edit View Insert Runtime Tools Help
Cell Commands + Code + Test
In [1]: Categorical Variables: ['store_and_fwd_flag']
Numerical Variables: ['vendorid', 'passenger_count', 'trip_distance', 'ratecodeid', 'pickuplocationid', 'dolocationid', 'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge', 'total_amount', 'congestion_surcharge', 'airport_fee', 'airport_fee']
Out[1]: 0
In [2]: Import seaborn as sns
Import matplotlib.pyplot as plt
# convert timestamps to datetime format
combined_df['trip_pickup_datetime'] = pd.to_datetime(combined_df['trip_pickup_datetime'])
combined_df['dropoff_datetime'] = pd.to_datetime(combined_df['dropoff_datetime'])
combined_df['day'] = pd.DatetimeIndex(combined_df['trip_pickup_datetime']).day_name()
combined_df['month'] = combined_df['trip_pickup_datetime'].dt.month_name()

# plot pickups by hour
plt.figure(figsize=(10, 6))
sns.countplot(x='hour', data=combined_df, palette='viridis')
plt.title('Taxi Pickups by Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=45)
plt.show()

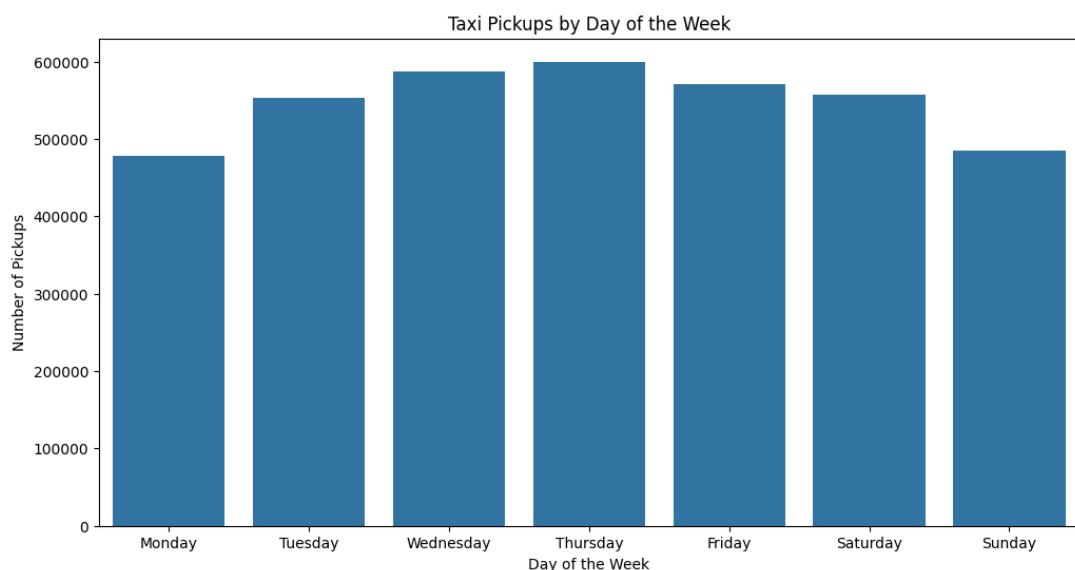
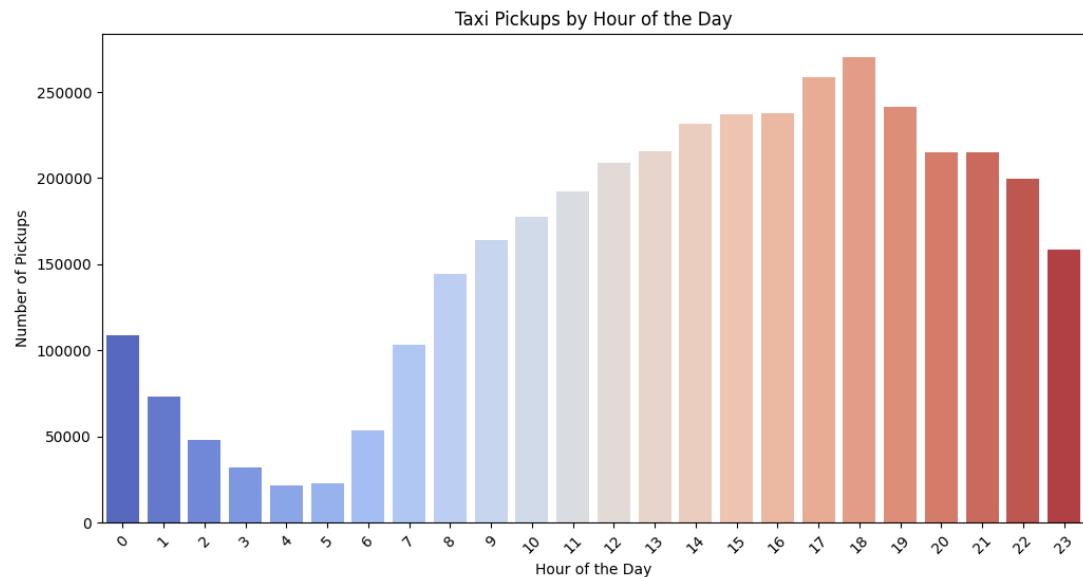
# plot pickups by day
plt.figure(figsize=(10, 6))
sns.countplot(x='day_of_week', data=combined_df, order=[('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday')])
plt.title('Taxi Pickups by Day of the Week')
plt.xlabel('Day of the Week')
plt.ylabel('Number of Pickups')
plt.show()

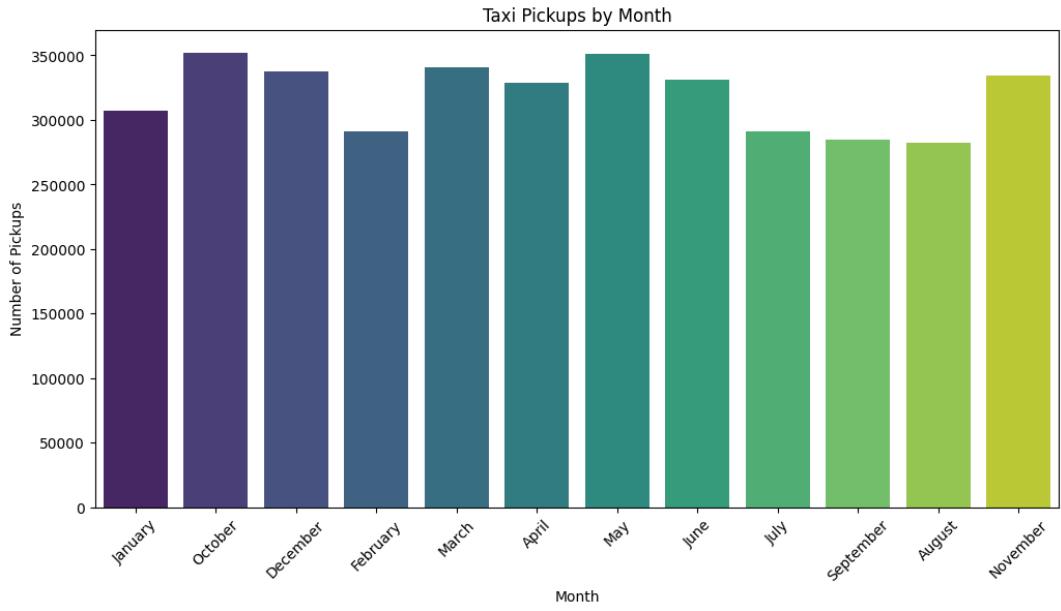
# plot pickups by month
plt.figure(figsize=(10, 6))
sns.countplot(x='month', data=combined_df, palette='viridis')
plt.title('Taxi Pickups by Month')
plt.xlabel('Month')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=45)
plt.show()

</pre>

```

result





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

```

# EDA on NYC Taxi Data.ipynb
File Edit View Insert Runtime Help
File Edit View Insert Runtime Help
In [1]: [REDACTED]
Out[1]: [REDACTED]
Categorical Variables: ['store_and_fwd_flag']
Numerical Variables: ['vendorid', 'passenger_count', 'trip_distance', 'tripcodeid', 'pickup_datetime', 'pickup_longitude', 'pickup_latitude', 'payment_type', 'fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge', 'total_amount', 'congestion_surcharge', 'airport_fee']

In [2]: [REDACTED]
Out[2]: [REDACTED]
import seaborn as sns
import matplotlib.pyplot as plt

# Convert timestamp to datetime format
sns.set(style="whitegrid")
sns.set_color_codes("viridis")
datacombined_df = datacombined_df[(datacombined_df['trip_pickup_datetime'] >= '2014-01-01') & (datacombined_df['trip_pickup_datetime'] <= '2014-12-31')]
combined_df['date'] = combined_df['trip_pickup_datetime'].dt.date
combined_df['day_of_week'] = combined_df['trip_pickup_datetime'].dt.day_name()
combined_df['month'] = combined_df['trip_pickup_datetime'].dt.month_name()

# Plot pickups by hour
plt.figure(figsize=(10, 6))
sns.countplot(x='hour', data=combined_df, palette='viridis')
plt.title('Taxi Pickups By Hour of the Day')
plt.xlabel('Hour of the Day')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=45)
plt.show()

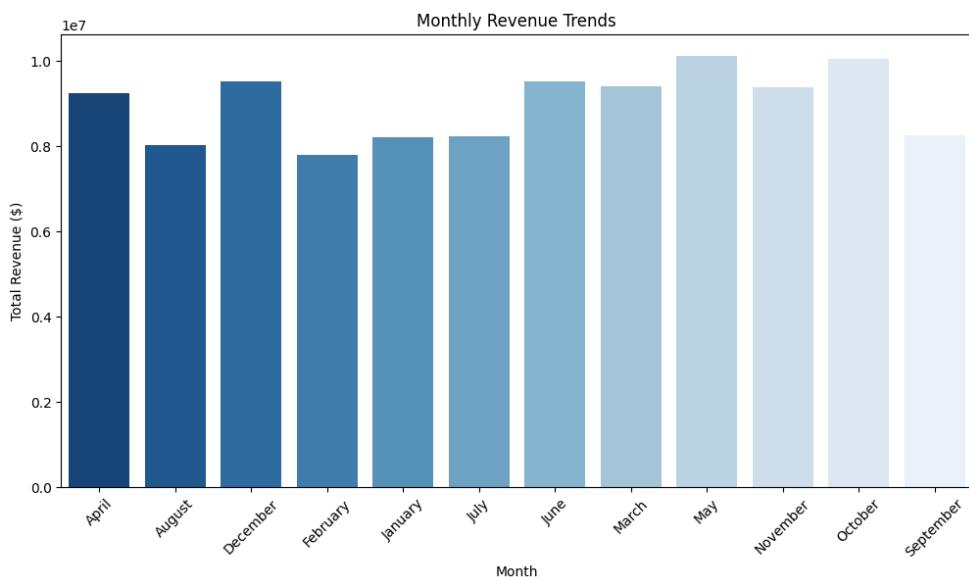
# Plot pickups by day
plt.figure(figsize=(10, 6))
sns.countplot(x='day', data=combined_df, order=['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.title('Taxi Pickups By Day of the week')
plt.xlabel('Day of the week')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=45)
plt.show()

# Plot pickups by month
plt.figure(figsize=(10, 6))
sns.countplot(x='month', data=combined_df, palette='viridis')
plt.title('Taxi Pickups By Month')
plt.xlabel('Month')
plt.ylabel('Number of Pickups')
plt.xticks(rotation=45)
plt.show()

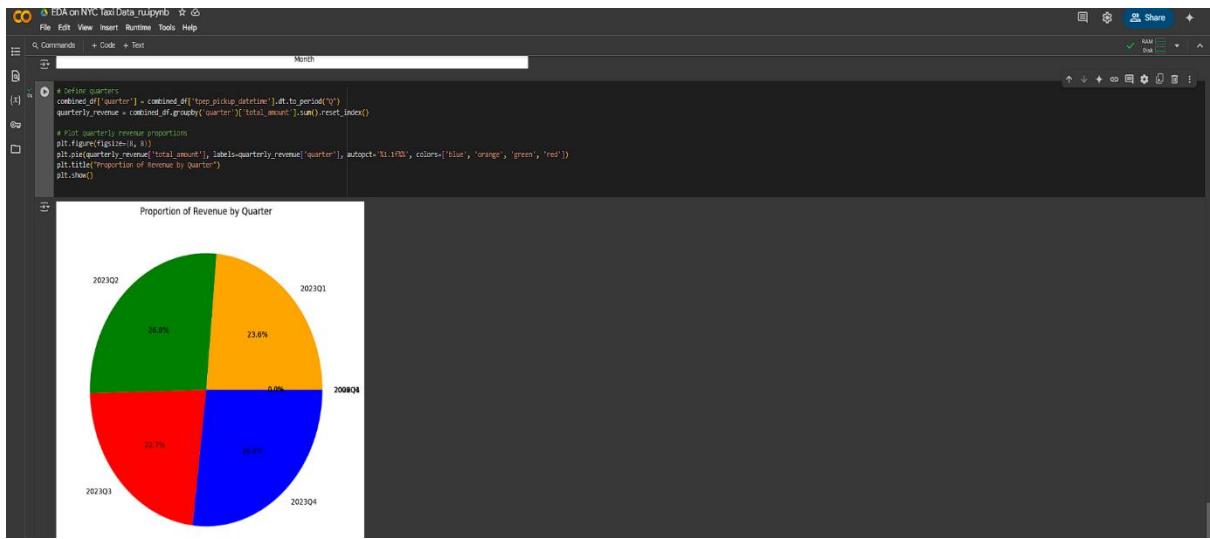
# Python warning
print(warnings.filterwarnings('ignore'))

```

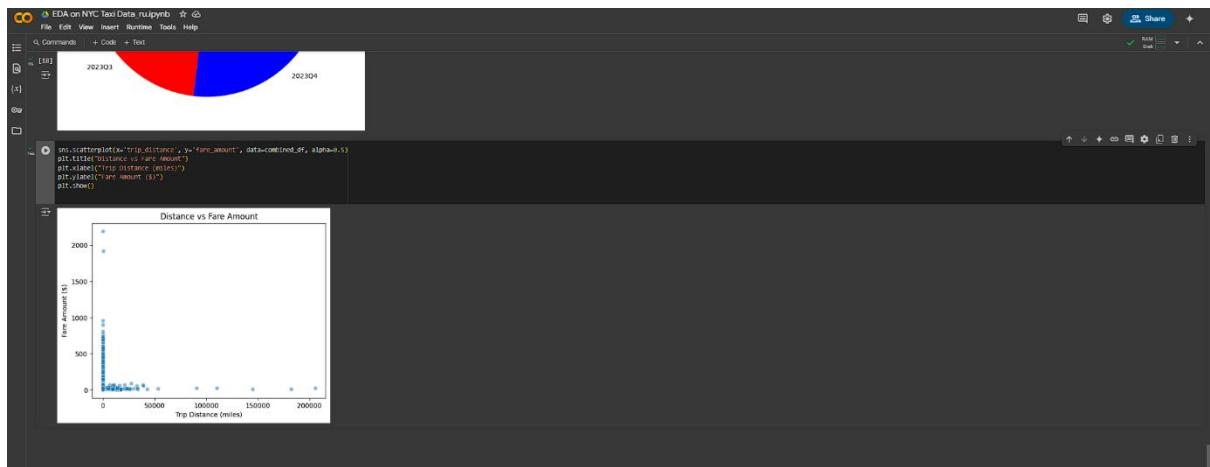
iv. Analyse the monthly revenue trends



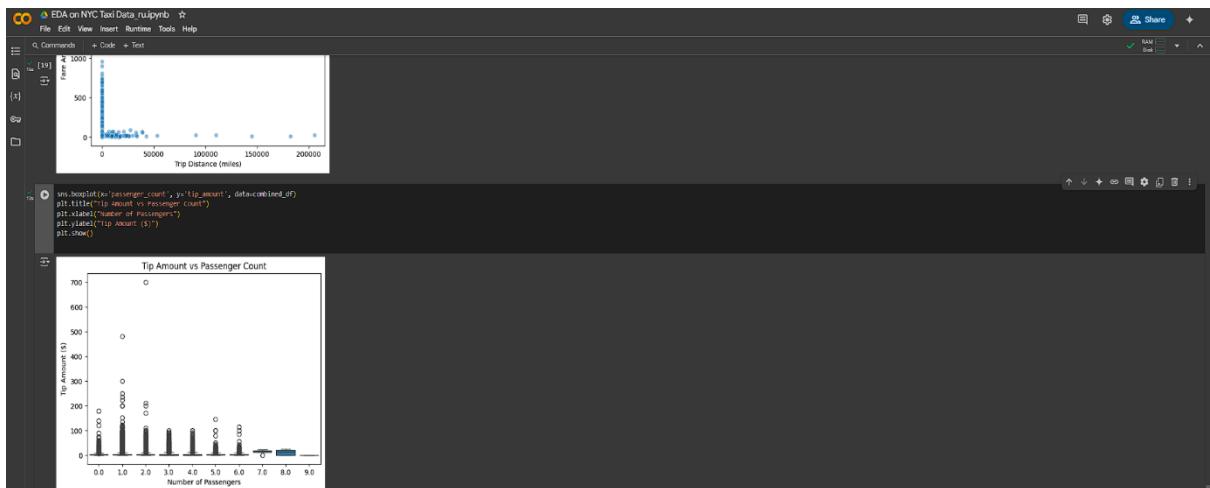
v. Find the proportion of each quarter's revenue in the yearly revenue



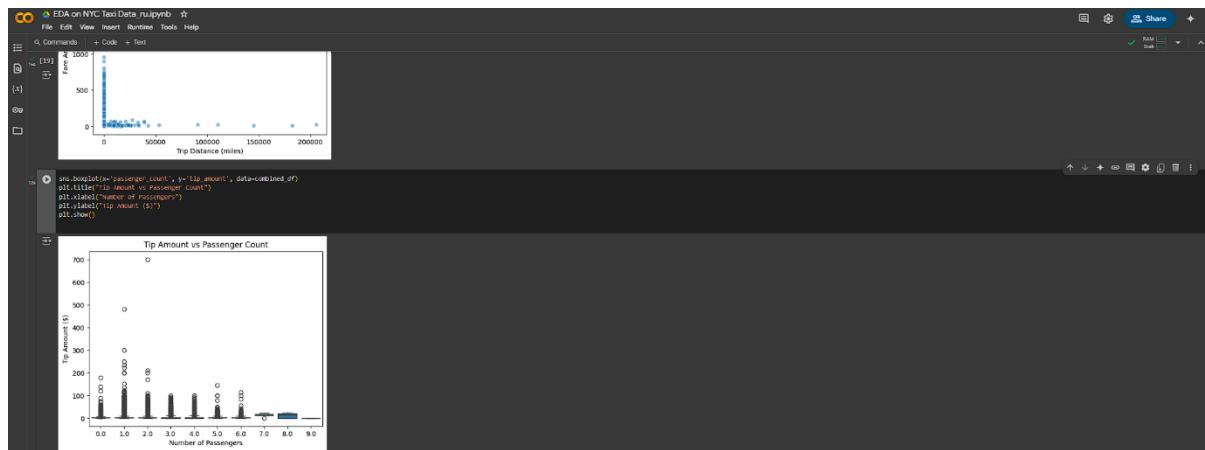
vi. Analyse and visualise the relationship between distance and fare amount



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



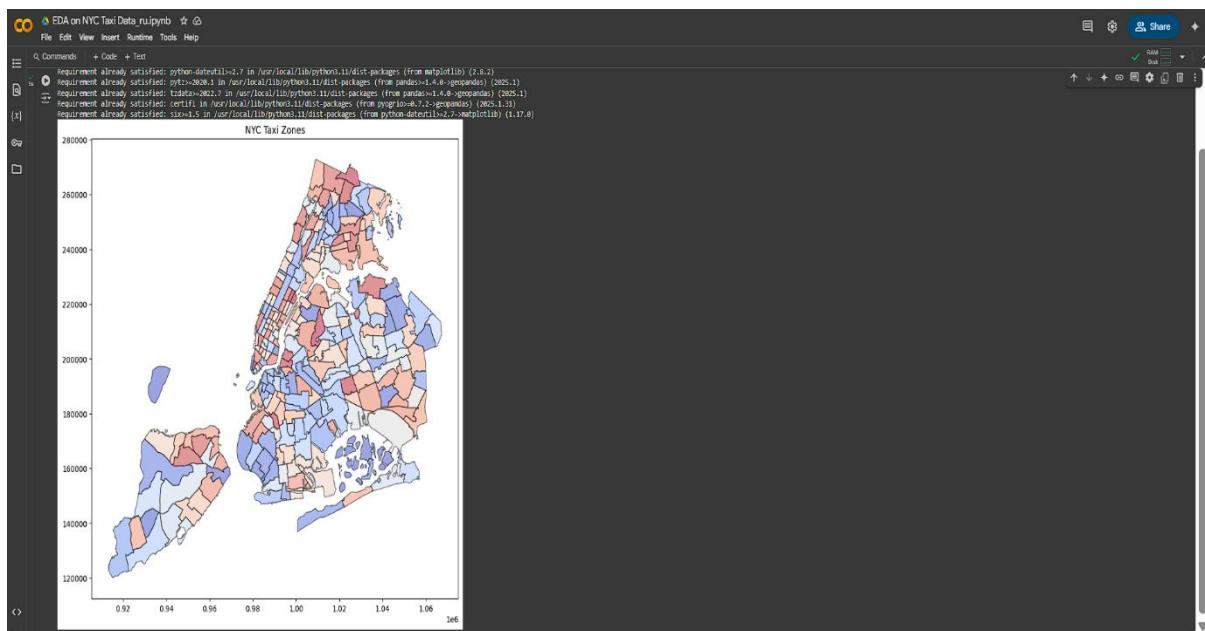
viii. Analyse the distribution of different payment types



ix. Load the taxi zones shapefile and display it

```
EDa on NYC Taxi Data.ipynb
File Edit View Insert Runtime Tools Help
Q Commands + Code + Text
Q wget < taxi_zones.zip "https://d37ci2e4zrcrfv.cloudfront.net/nyc/taxi_zones.zip"
A [1]: 2025-03-11 14:00:59 -> https://d37ci2e4zrcrfv.cloudfront.net/nyc/taxi_zones.zip
  0%|          | 0/100 [00:00<00:00, ?B/s]
  20%|██████████| 20/100 [00:00<00:00, 10B/s]
  40%|███████████| 40/100 [00:00<00:00, 10B/s]
  60%|███████████| 60/100 [00:00<00:00, 10B/s]
  80%|███████████| 80/100 [00:00<00:00, 10B/s]
  100%|███████████| 100/100 [00:00<00:00, 10B/s]
Saving to: 'taxi_zones.zip' [100%] 100K --:-- /s in 0.00s
2025-03-11 14:00:59 (12.0 MB/s) - 'taxi_zones.zip' saved [100%]
[2]: import zipfile
# extract the zip file
with zipfile.ZipFile('taxi_zones.zip', 'r') as zip_ref:
    zip_ref.extractall("taxi_zones")
print("Shapefile extracted successfully!")

Shapefile extracted successfully!
[3]: pip install geopandas matplotlib # install required libraries (if not installed)
Requirement already satisfied: geopandas in /usr/local/lib/python3.11/dist-packages (from matplotlib) (0.9.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from matplotlib) (3.16.0)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.11/dist-packages (from geopandas) (1.24.0)
Requirement already satisfied: pandas>=1.4.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (1.4.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.4.2)
Requirement already satisfied: python-dateutil>=2.7.0 in /usr/local/lib/python3.11/dist-packages (from geopandas) (2.2.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7>matplotlib) (1.17.0)
```



x. Merge the zone data with trips data

```

File Edit View Insert Runtime Tools Help
Q Commands + Code + Text
[3] # Display column names for trip data
print("Trip Data columns:", trip_data.columns)

[4] # Display column names for taxi zones
print("Taxi Zone columns:", zones.columns)

[5] Trip Data Columns: Index(['VendorID', 'trip_start_datetime', 'trip_dropoff_datetime',
   'passenger_count', 'trip_distance', 'RatecodeID', 'store_and_fwd_flag',
   'payment_type', 'fare_amount', 'extra', 'mtaxi', 'tip_amount', 'tolls_amount', 'improvement_surcharge',
   'total_amount', 'congestion_surcharge', 'Airport_fee'],
   dtype='object')
Taxi Zone Columns: Index(['locationID', 'Shape_Leng', 'Shape_Area', 'zone', 'locationBorough', 'borough',
   'geography', 'object'])

[6] # Convert locationID columns to integer for correct merging
df['LocationID'] = df['PULocationID'].astype(int)
df['locationID'] = df['DOLocationID'].astype(int)
zones['locationID'] = zones['locationID'].astype(int)

[7] # Merge for pickup location
df = df.merge(zones[['locationID', 'zone', 'borough']],
             left_on='PULocationID',
             right_on='locationID',
             how='left',
             suffixes('_x', '_pickup'))

df.rename(columns={'zone': 'Pickup_Zone', 'borough': 'Pickup_Borough'}, inplace=True)

# Merge for drop-off location
df = df.merge(zones[['locationID', 'zone', 'borough']],
             left_on='DOLocationID',
             right_on='locationID',
             how='left',
             suffixes('_x', '_dropoff'))

df.rename(columns={'zone': 'Dropoff_Zone', 'borough': 'Dropoff_Borough'}, inplace=True)

# Drop duplicate locationID columns
df.drop(columns=['locationID'], errors='ignore', inplace=True)

print("Merging completed!")

Merging completed!

```

both datasets are now merged.

verifying

```

File Edit View Insert Runtime Tools Help
Q Commands + Code + Text
[3] df = df.merge(zones[['locationID', 'zone', 'borough']],
                 left_on='PULocationID',
                 right_on='locationID',
                 how='left',
                 suffixes('_x', '_dropoff'))

df.rename(columns={'zone': 'Dropoff_Zone', 'borough': 'Dropoff_Borough'}, inplace=True)

# Drop duplicate locationID columns
df.drop(columns=['locationID'], errors='ignore', inplace=True)

print("Merging completed!")

df[['LocationID', 'Pickup_Zone', 'Pickup_Borough', 'Dropoff_Zone', 'Dropoff_Borough']].head(10)

```

	LocationID	Pickup_Zone	Pickup_Borough	Dropoff_Zone	Dropoff_Borough
0	230	Times Sq/Hudson Dkred	Times Sq/Hudson Dkred	Clinton East	Clinton East
1	142	Lincoln Square East	Lincoln Square East	Clinton East	Clinton East
2	114	Greenwich Village South	Greenwich Village South	Clinton East	Clinton East
3	79	East Village	East Village	Clinton East	Clinton East
4	229	Sutton Place/Turtle Bay North	Sutton Place/Turtle Bay North	Clinton East	Clinton East
5	88	Financial District South	Financial District South	Clinton East	Clinton East
6	45	Chinatown	Chinatown	Clinton East	Clinton East
7	170	Murray Hill	Murray Hill	Clinton East	Clinton East
8	198	Pine Station/Madison Sq West	Pine Station/Madison Sq West	Clinton East	Clinton East

Showing top 10 places

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

```

File Edit View Insert Runtime Tools Help
Q Commands + Code + Text
[1] # Drop duplicate locationID columns
df.drop_duplicates('locationID', errors='ignore', inplace=True)

print("Merging completed!")

df[['LocationID', 'Pickup_Zone', 'Pickup_Borough', 'Dropoff_Zone', 'Dropoff_Borough']].head(10)

```

	LocationID	Pickup_Zone	Pickup_Borough	Dropoff_Zone	Dropoff_Borough
0	230	Times Sq/Hudson Dkred	Times Sq/Hudson Dkred	Clinton East	Clinton East
1	142	Lincoln Square East	Lincoln Square East	Clinton East	Clinton East
2	114	Greenwich Village South	Greenwich Village South	Clinton East	Clinton East
3	79	East Village	East Village	Clinton East	Clinton East
4	229	Sutton Place/Turtle Bay North	Sutton Place/Turtle Bay North	Clinton East	Clinton East
5	88	Financial District South	Financial District South	Clinton East	Clinton East
6	45	Chinatown	Chinatown	Clinton East	Clinton East
7	170	Murray Hill	Murray Hill	Clinton East	Clinton East
8	198	Pine Station/Madison Sq West	Pine Station/Madison Sq West	Clinton East	Clinton East

```

[4] # Count trips for each location
trip_counts = df['LocationID'].value_counts().reset_index()
trip_counts.columns = ['LocationID', 'Number_of_Trips']

# Sorting by 50 highest ones
trip_counts.sort_values('Number_of_Trips', ascending=False).head(50)

```

LocationID	Number_of_Trips
0	232
1	132
2	236
3	101
4	152
5	112487
6	230
7	188
8	130

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

The screenshot shows a Jupyter Notebook interface with the following code and output:

```

# Step 1: Check if columns exist
print(trip_counts.columns)
print(zones.columns)

# Step 2: Ensure locationID is in integer format for correct merging
trip_counts['locationID'] = trip_counts['locationID'].astype(int)
zones['locationID'] = zones['locationID'].astype(int)

# Step 3: Merge trip counts with taxi zones data
zones = zones.merge(trip_counts, on='locationID', how='left')

# Step 4: Fill na values with 0 (some zones might not have trips)
zones['number_of_trips'] = zones['number_of_trips'].fillna(0).astype(int)

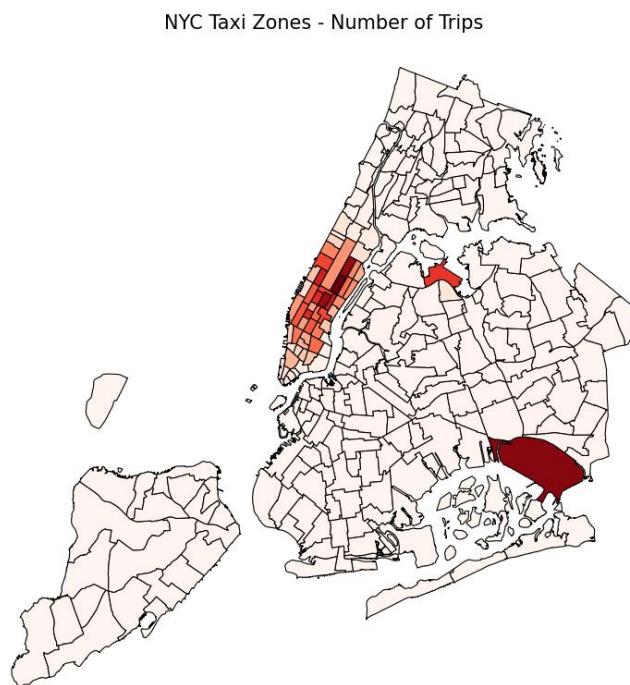
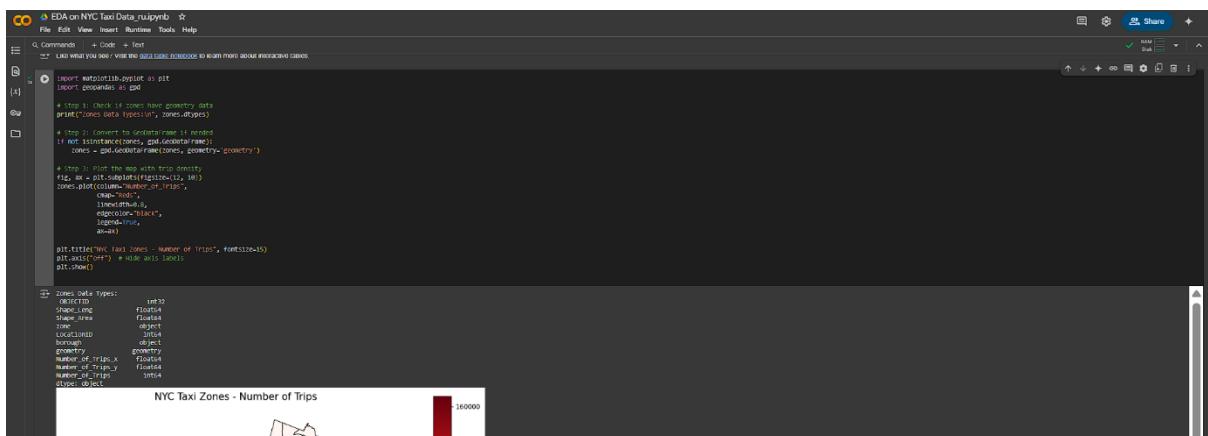
# Display the first few rows to check if the merge worked
zones[['locationID', 'zone', 'number_of_trips']].head(10)

```

The output table shows the merged data with the 'Number of Trips' column added:

Index	LocationID	zone	Number of Trips
0	4	Alphabet City	4153
1	5	Astoria	1095
2	6	Bayway Park	1091
3	7	Heathrow Airport	692
4	8	Midtown Manhattan	23
5	9	Armenia	96
6	10	Astoria Park	20
7	11	Queens Center Mall	17
8	12	Central Islip	5
9	13	Asian Heights	1

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

```
② EDA on NYC Taxi Data - JupyterLab
File Edit View Insert Runtime Tools Help
Cell Commands Code Test
③ Cell Kernel Help
Import module(s) as plt
Import pandas as pd
Import numpy as np
Import square as sq
Load the NYC taxi data file (ensure it is already loaded)
# If needed, load auto first (it is already present in your data package)
# Convert datetime columns to proper format
df['trip_start_datetime'] = pd.to_datetime(df['trip_start_datetime'])
df['trip_end_datetime'] = pd.to_datetime(df['trip_end_datetime'])

# Step 1: Calculate Trip Duration (Convert to Hours)
df['trip_duration_hours'] = (df['trip_duration'] / df['trip_pickup_datetime']) / dt.total_seconds() / 3600

# Step 2: Calculate Speed (in MPH)
df['trip_speed_mph'] = df['trip_distance'] / df['trip_duration_hours']

# Step 3: Remove Invalid Speeds (Filtering Outliers)
df = df[(df['trip_speed_mph'] > 0) & (df['trip_speed_mph'] < 60)]

slow_routes = df.groupby(['PULocationID', 'DOLocationID'])['trip_speed_mph'].mean().reset_index()

# Sort by lowest speed
slow_routes = slow_routes.sort_values(by='trip_speed_mph', ascending=True)

# Display Top 10 Slowest Routes
print("Top 10 Slowest Routes (Average Speed in MPH):")
print(slow_routes[['PULocationID', 'DOLocationID', 'trip_speed_mph']].head(10))

# Step 4: Visualising Slow Routes in a histogram
plt.figure(figsize=(12, 4))
plt.hist(df['trip_speed_mph'], bins=50, norm=True, color='red')
plt.xlabel('Trip Speed (MPH)')
plt.ylabel('Frequency')
plt.title('Distribution of Trip Speeds - Identifying Slow Routes')
plt.show()

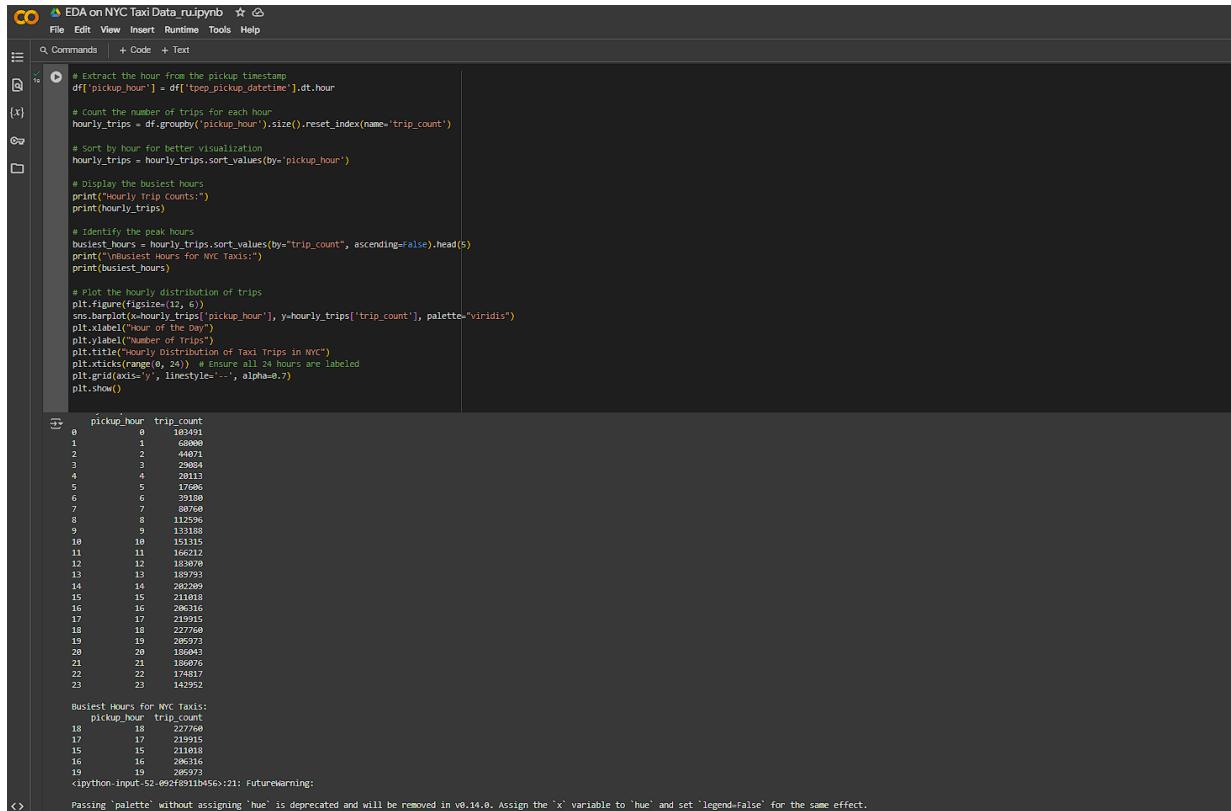
# Business Insights from Slow Routes
print("Business Insights from Slow Routes:")
print("Slowest routes are mostly in dense urban centers (e.g., Manhattan, Downtown Brooklyn) have lowest speeds.")
print("Short trips in dense areas tend to be slower than long-distance ones.")
print("Traffic pricing and ride pooling could be optimized for these 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



```
# Extract the hour from the pickup timestamp
df['pickup_hour'] = df['trip_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("Busiest 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()
```

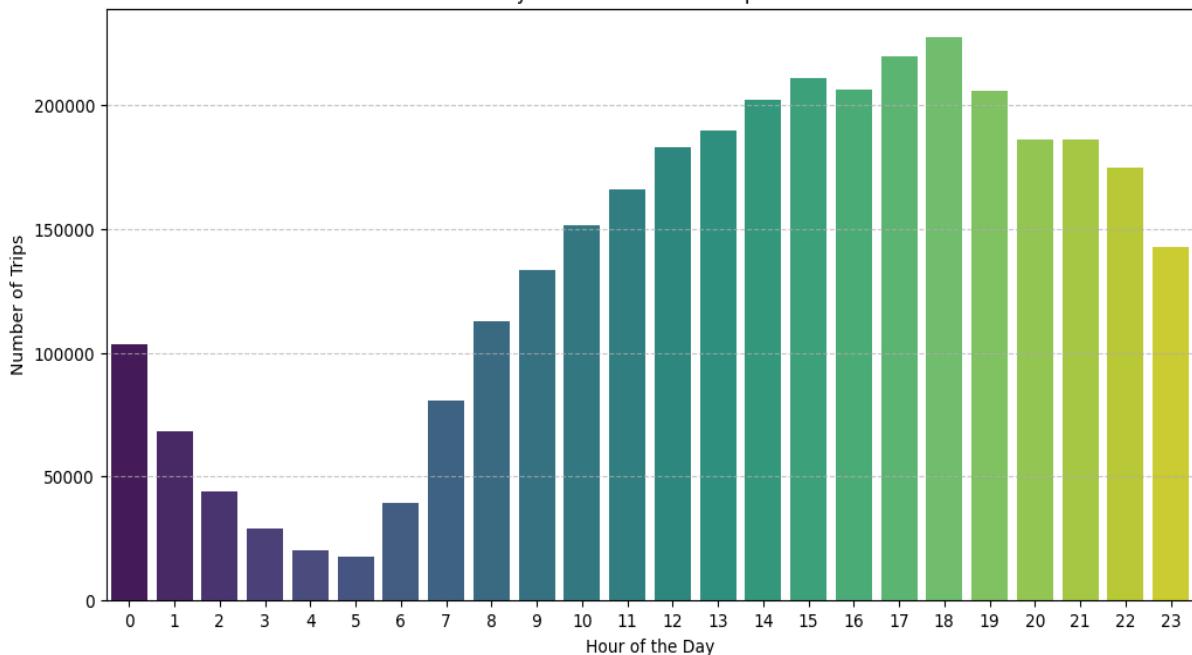
pickup_hour	trip_count
0	103491
1	68000
2	44671
3	29298
4	20113
5	17686
6	39188
7	39978
8	11256
9	133188
10	151315
11	166212
12	13779
13	189793
14	202209
15	211818
16	206116
17	21153
18	227768
19	205973
20	186643
21	186676
22	175017
23	142952

Busiest Hours for NYC Taxis:

pickup_hour	trip_count
18	227768
17	211915
15	211818
16	206116
19	205973

<ipython-input-52-e9248911b056>: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.

Hourly Distribution of Taxi Trips in NYC

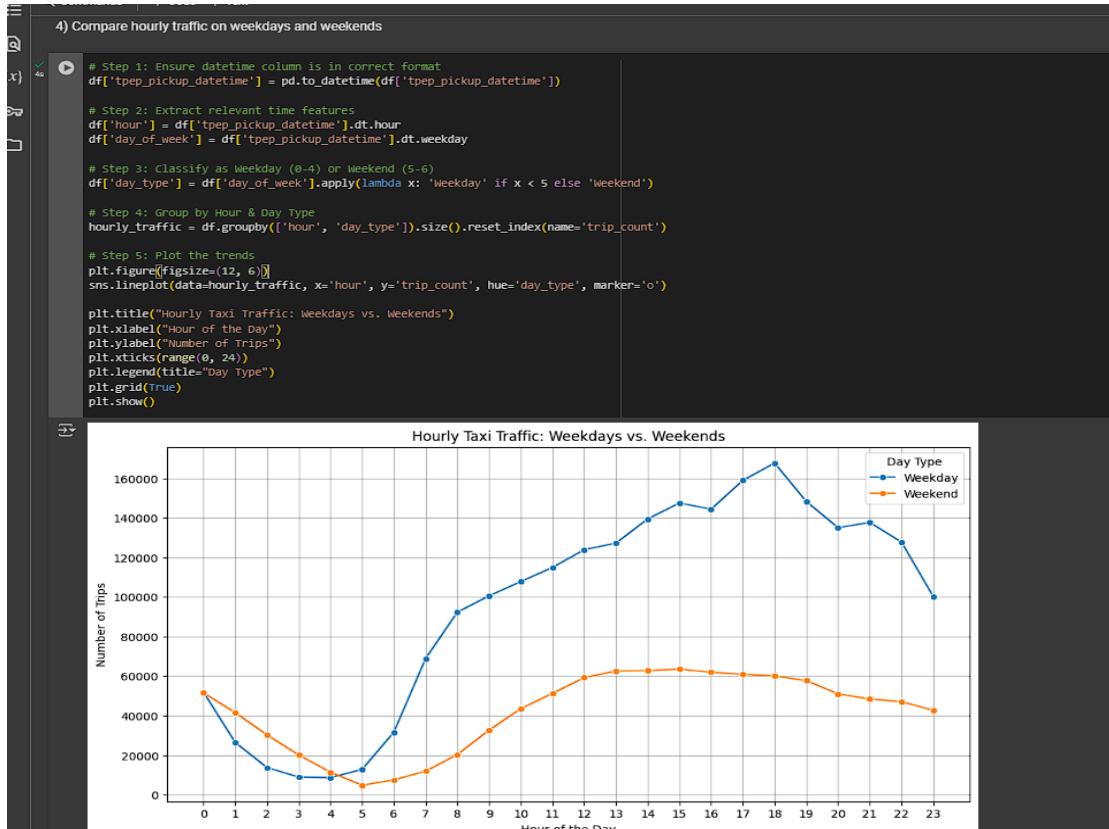


Insights

- 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

iv. Compare hourly traffic on weekdays and weekends



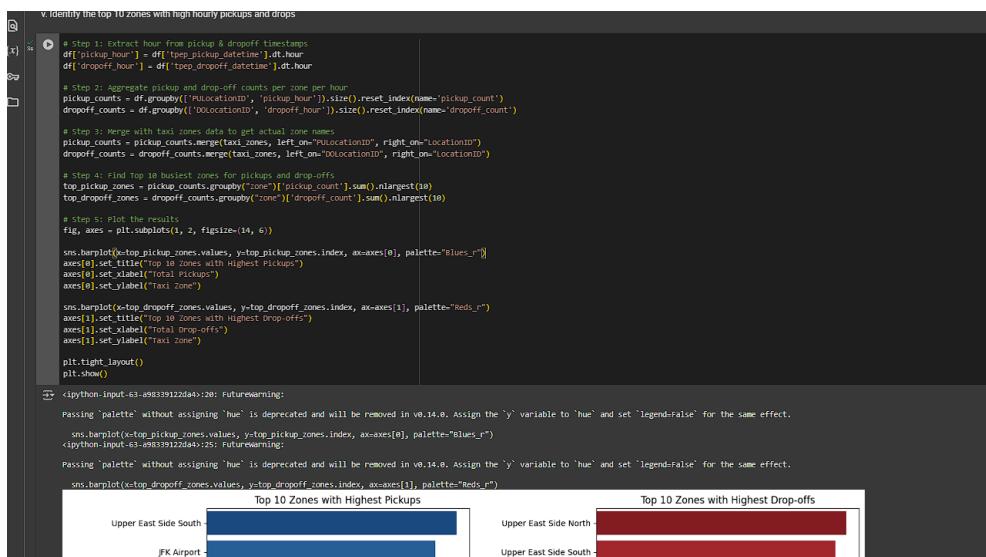
Weekday Trends:

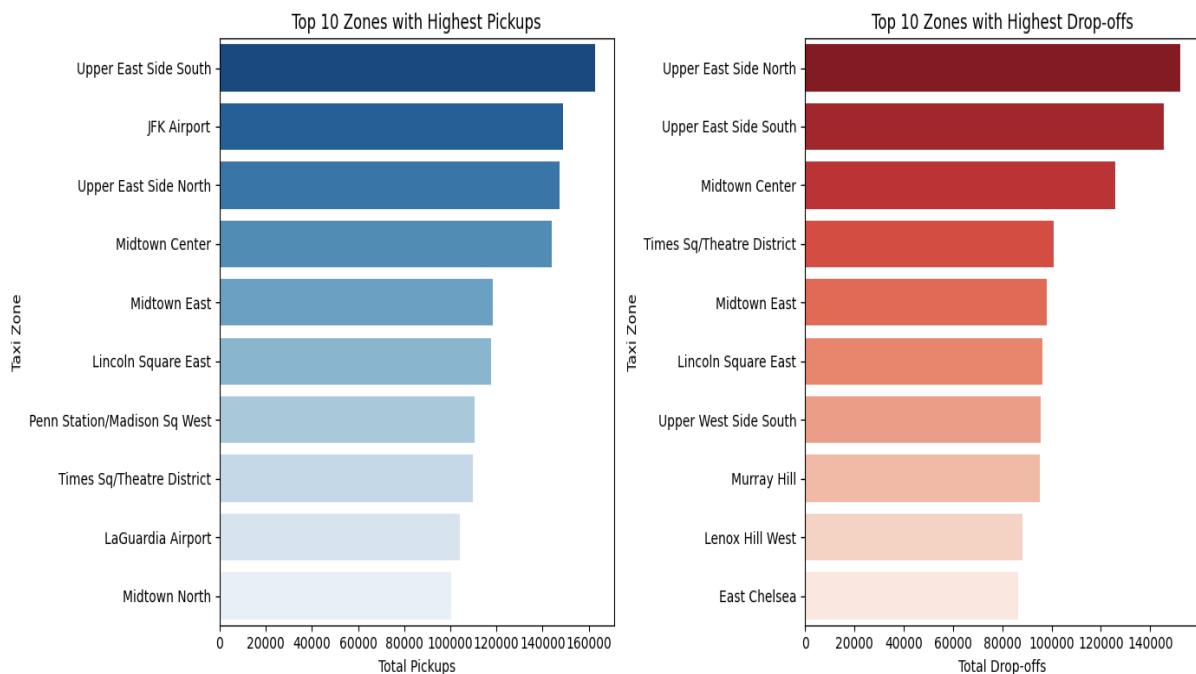
- Peak hours during morning rush (7-9 AM) and evening rush (5-8 PM).
- Higher trips during commute times (work-related travel).

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





- 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

```

# Step 1: Count pickups and drop-offs per zone
pickup_counts = df.groupby('PULocationID').size().reset_index(name='total_pickups')
dropoff_counts = df.groupby('DOLocationID').size().reset_index(name='total_dropoffs')

# Step 2: Merge both counts into a single DataFrame
zone_flow = pd.merge(pickup_counts, dropoff_counts, left_on="PULocationID", right_on="DOLocationID", how="outer")

# Rename columns for clarity
zone_flow.rename(columns={"PULocationID": "LocationID"}, inplace=True)

# Step 3: Fill missing values (some zones might have only pickups or only drop-offs)
zone_flow.fillna(0, inplace=True)

# Step 4: Calculate pickup-to-drop-off ratio (avoid division by zero)
zone_flow['pickup_dropoff_ratio'] = zone_flow['total_pickups'] / zone_flow['total_dropoffs'].replace(0, 1)

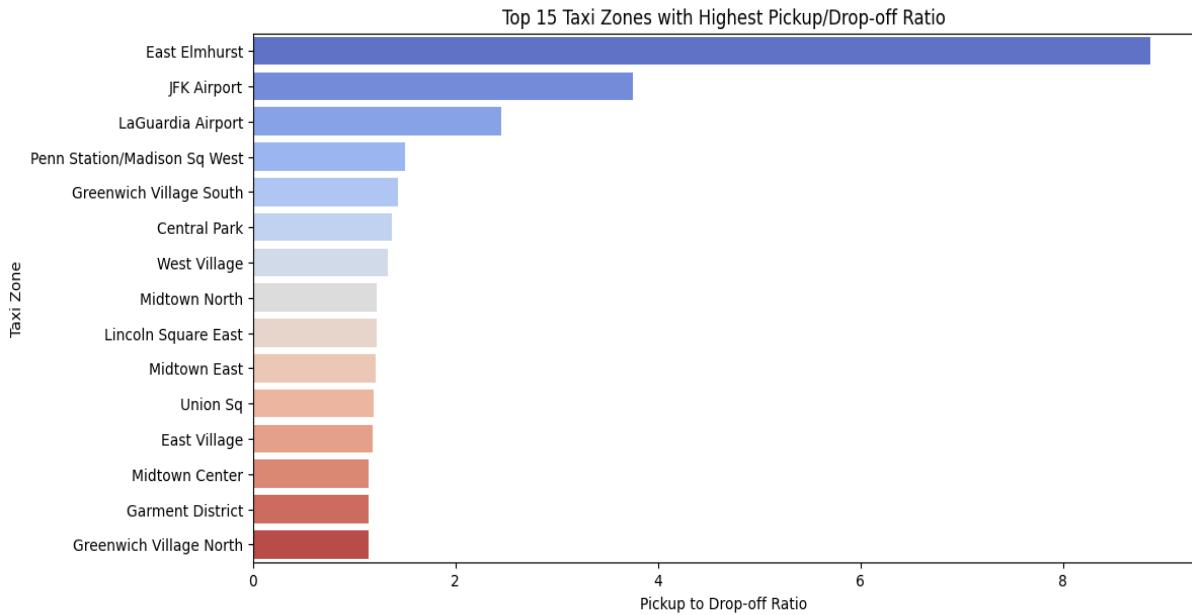
# Step 5: Merge with taxi zone names
zone_flow = zone_flow.merge(taxi_zones, on="LocationID", how="left")

# Step 6: Sort by ratio and visualize
top_zones = zone_flow.sort_values(by="pickup_dropoff_ratio", ascending=False).head(15)

plt.figure(figsize=(12, 6))
sns.barplot(y=top_zones["zone"], x=top_zones["pickup_dropoff_ratio"], palette="coolwarm")
plt.xlabel("Pickup to Drop-off Ratio")
plt.ylabel("Taxi Zone")
plt.title("Top 15 Taxi Zones with Highest Pickup/Drop-off Ratio")
plt.show()

```

<ipython-input-64-054b92a5dd4>:24: FutureWarning:



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

EDA on NYC Taxi Data.ipynb

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

# Step 2: Filter night-time trips (8 PM - 5 AM)
night_df = df[(df["pickup_hour"] >= 20) | (df["pickup_hour"] <= 5)]

# Step 3: Count night-time pickups per zone
night_pickup_counts = night_df.groupby("PULocationID").size().reset_index(name="night_pickups")

# Step 4: Merge with taxi zone names
night_pickup_counts = night_pickup_counts.merge(taxi_zones, left_on="PULocationID", right_on="LocationID")

# Step 5: Get top 10 busiest night-time zones
top_night_zones = night_pickup_counts.sort_values(by="night_pickups", ascending=False).head(10)

# Step 6: Visualize results
plt.figure(figsize=(12, 6))
sns.barplot(y=top_night_zones["zone"], x=top_night_zones["night_pickups"], palette="magma")
plt.xlabel("Number of Pickups")
plt.ylabel("Taxi Zone")
plt.title("Top 10 Taxi Zones with Highest Night-Time Pickups (8 PM - 5 AM)")
plt.show()
```

<ipython-input-65-f7f3e0adea6>:18: 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.

Top 10 Taxi Zones with Highest Night-Time Pickups (8 PM - 5 AM)

Taxi Zone	Number of Pickups
JFK Airport	45000
East Village	42000
Midtown Center	40000
West Village	40000
Clinton East	38000
Lincoln Square East	38000
Times Sq/Theatre District	36000
Upper East Side South	36000
Penn Station/Madison Sq West	30000
Midtown East	30000

viii. Find the revenue share for nighttime and daytime hours

EDA on NYC Taxi Data_ru.ipynb

```
# 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
Daytime	67198972.67	71.3506
Nighttime	26974930.16	28.64394

Daytime- 71.4% and Night time-28.6%

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

{x} 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)]

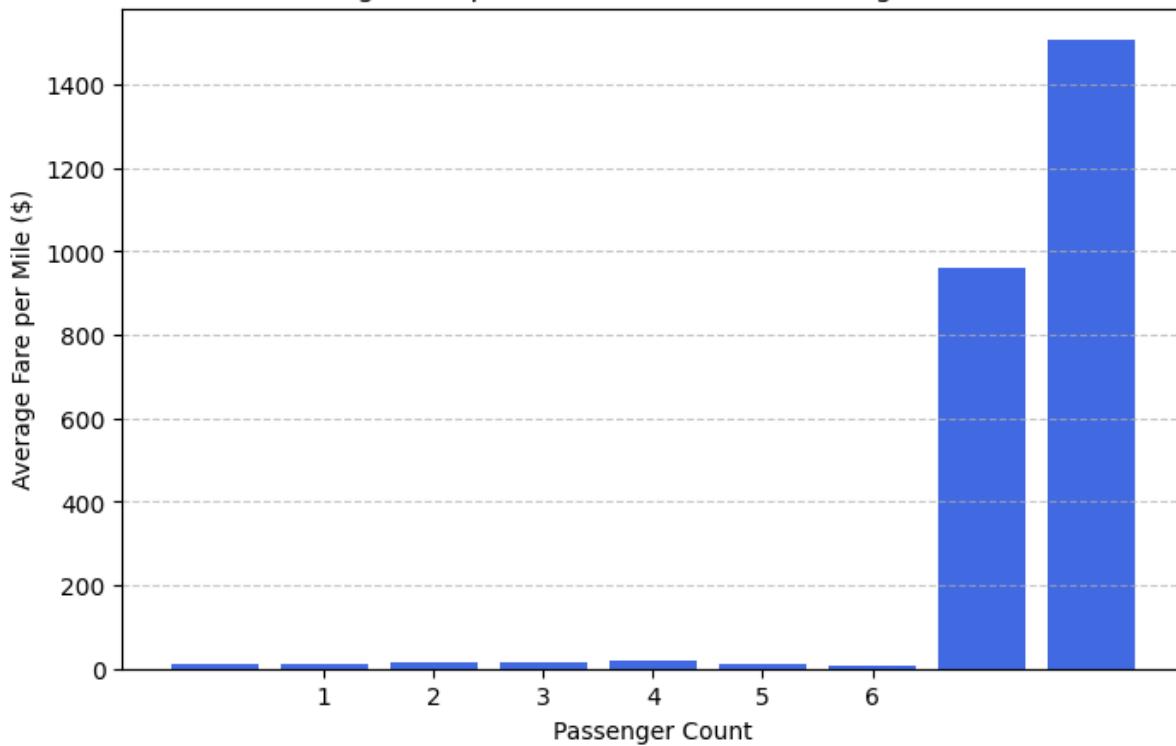
# Step 2: Calculate fare per mile
df_filtered["fare_per_mile"] = df_filtered["fare_amount"] / df_filtered["trip_distance"]

# 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
```

Average Fare per Mile for Different Passenger Counts



passenger_count	fare_per_mile
0	9.763828
1	11.325722
2	15.108257
3	13.483561
4	20.555640
5	10.183452
6	8.959234
7	959.550215
8	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

```
# 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")

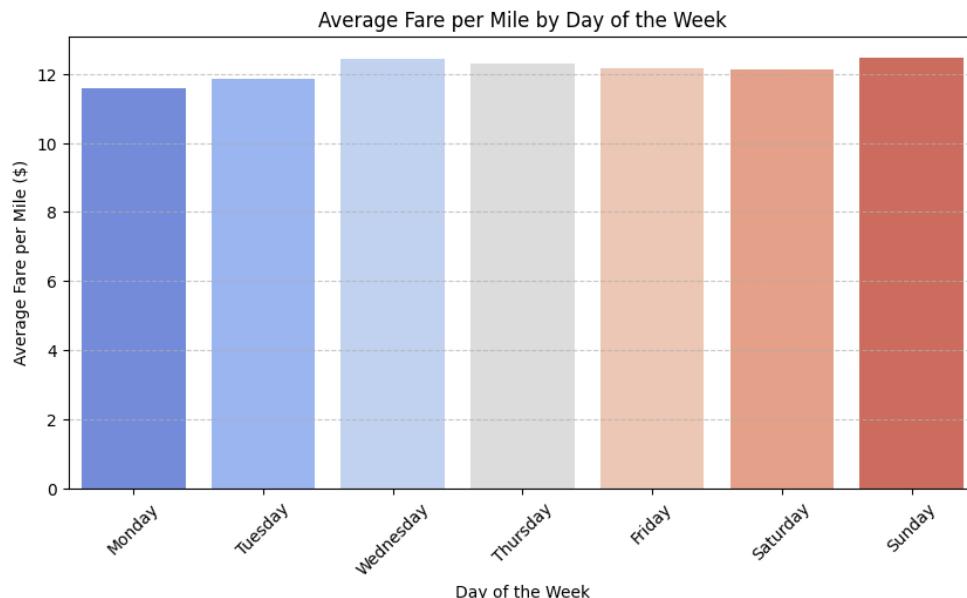
# Plot average fare per mile by hour
plt.figure(figsize=(10, 5))
sns.lineplot(data=fare_per_mile_by_hour, x="pickup_hour", y="fare_per_mile", marker="o", color="blue")
plt.xlabel("Hour of the Day")
plt.ylabel("Average Fare per Mile ($)")
plt.title("Average Fare per Mile by Hour of the Day")
plt.xticks(range(0,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

EDA on NYC Taxi Data.ipynb

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

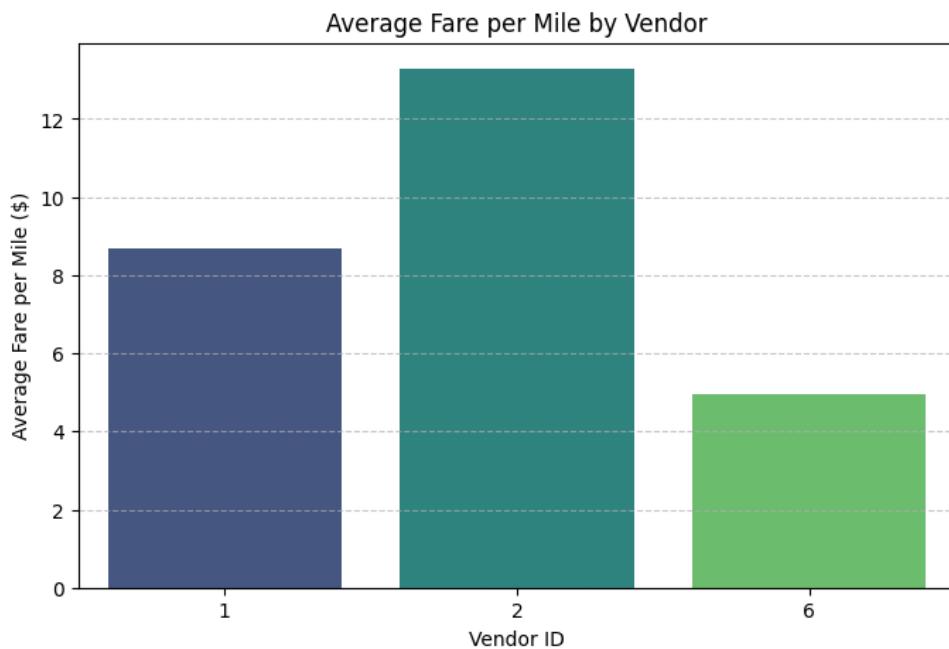
# Calculate fare per mile
df_filtered["fare_per_mile"] = df_filtered["fare_amount"] / df_filtered["trip_distance"]

# Compute average fare per mile by vendor
fare_per_mile_by_vendor = df_filtered.groupby("VendorID")["fare_per_mile"].mean().reset_index()

# Plot average fare per mile by vendor
plt.figure(figsize=(8, 5))
sns.barplot(data=fare_per_mile_by_vendor, x="VendorID", y="fare_per_mile", palette="viridis")
plt.xlabel("Vendor ID")
plt.ylabel("Average Fare per Mile ($)")
plt.title("Average Fare per Mile by Vendor")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# Print results
print("Average Fare per Mile by Vendor:")
print(fare_per_mile_by_vendor)
```

<ipython-input-69-3dc2f6ad2531>:5: 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 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

```
# 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)

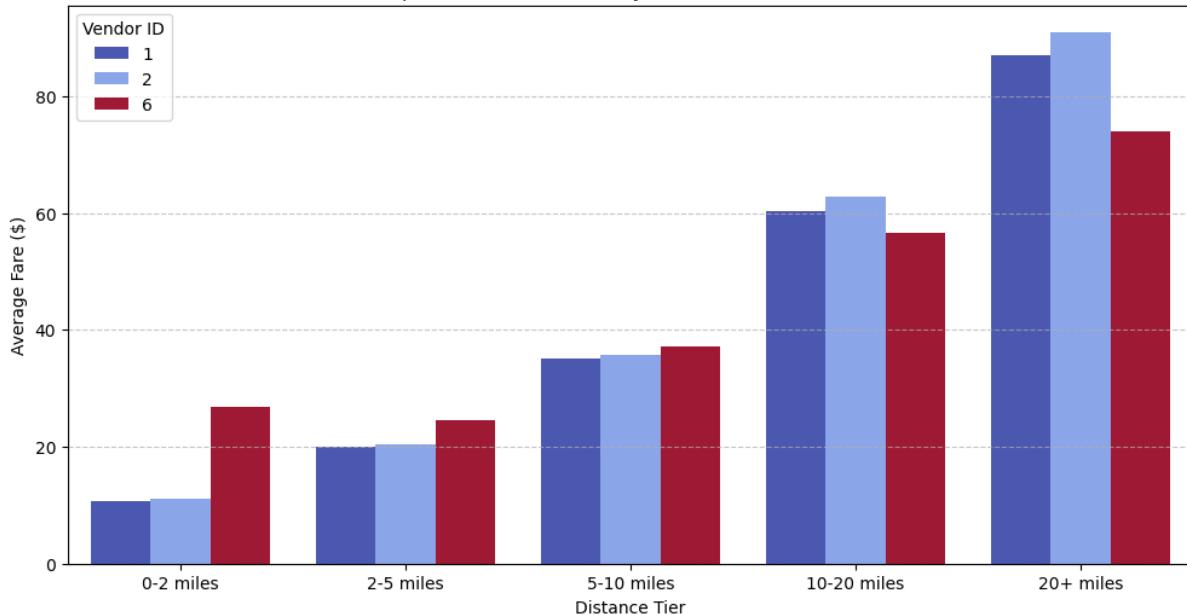
# Calculate average fare per mile for each vendor and distance tier
fare_per_mile_by_vendor_tier = df_filtered.groupby(["VendorID", "distance_tier"])["fare_amount"].mean().reset_index()

# 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-e35dfb83beif>: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

Comparison of Fare Rates by Vendor and Distance Tier



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

```

xiii. Analyse the tip percentages

19a # Remove trips with zero/negative fare or tip amounts
df_filtered = df[(df["fare_amount"] > 0) & (df["tip_amount"] >= 0)]

# Calculate tip percentage
df_filtered["tip_percentage"] = (df_filtered["tip_amount"] / df_filtered["fare_amount"]) * 100

# Summary statistics
tip_stats = df_filtered["tip_percentage"].describe()
print("Tip Percentage Statistics:")
print(tip_stats)

# Visualize tip percentage distribution
plt.figure(figsize=(10, 5))
sns.histplot(df_filtered["tip_percentage"], bins=50, kde=True, color="blue")
plt.xlabel("Tip Percentage")
plt.ylabel("Count of Trips")
plt.title("Distribution of Tip Percentages")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

# Analyze tip percentage by payment type
tip_by_payment = df_filtered.groupby("payment_type")["tip_percentage"].mean().reset_index()

plt.figure(figsize=(8, 5))
sns.barplot(data=tip_by_payment, x="payment_type", y="tip_percentage", palette="coolwarm")
plt.xlabel("Payment Type")
plt.ylabel("Average Tip Percentage")
plt.title("Average Tip Percentage by Payment Type")
plt.grid(axis="y", linestyle="--", alpha=0.7)
plt.show()

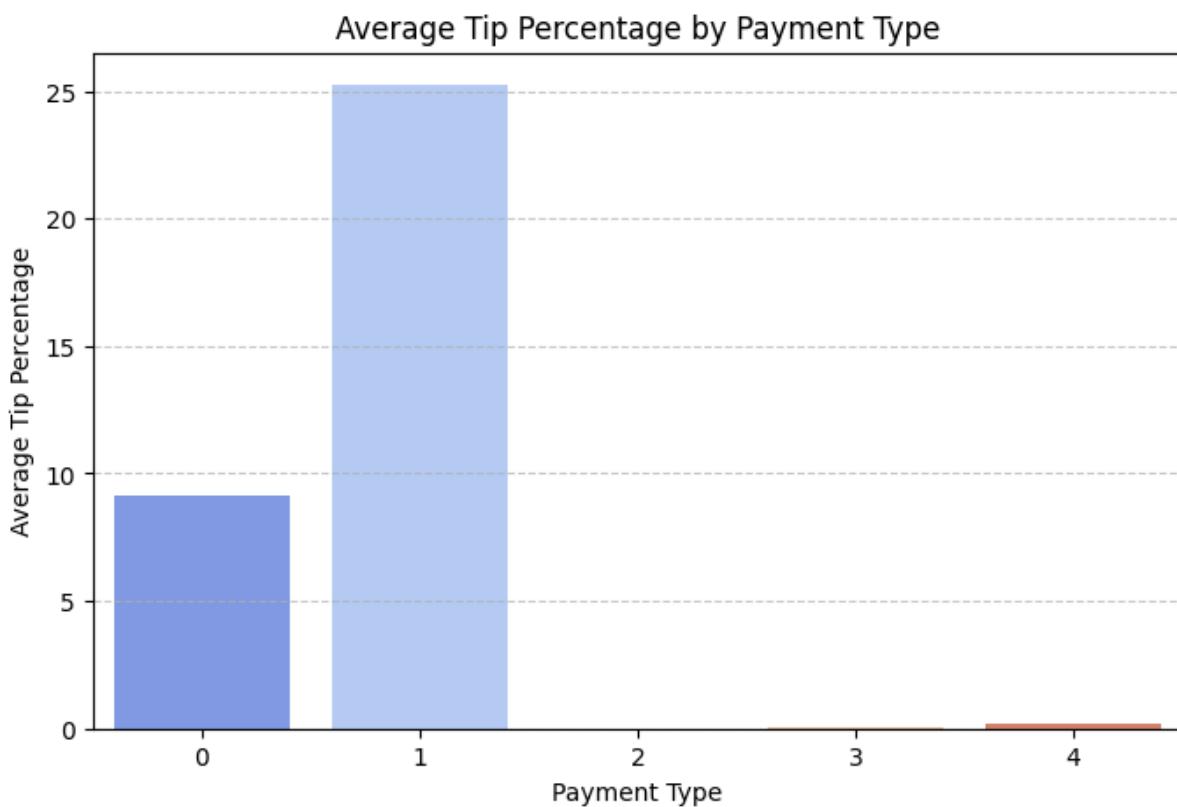
# Print results
print("Average Tip Percentage by Payment Type:")
print(tip_by_payment)

```

→ <ipython-input-71-91e287424efb>:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.

Average Tip Percentage by Payment Type:

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



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

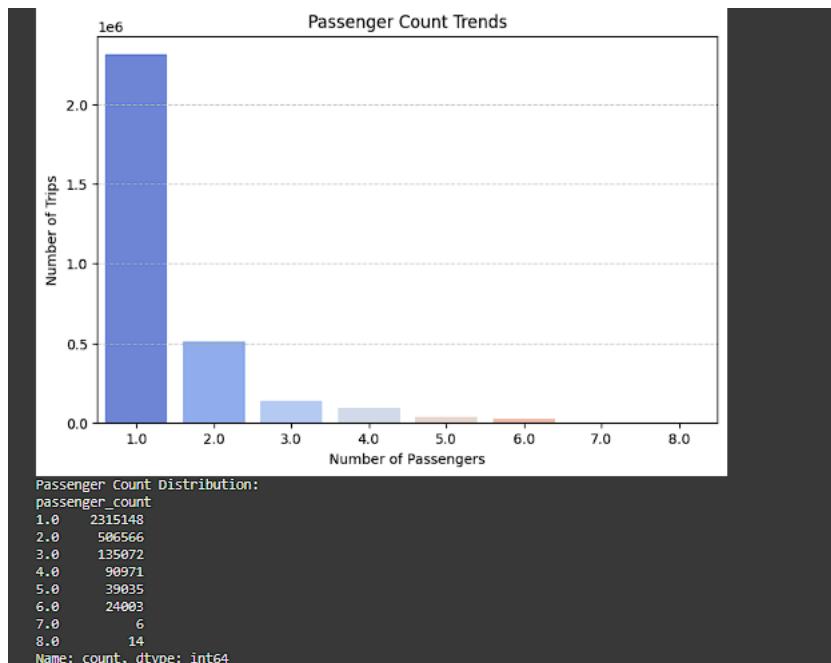
xiv. Analyse the trends in passenger count

```
# 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)
```



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),
                y="zone", x="passenger_count", palette="viridis")
    Top 20 Zones with Highest Average Passenger Count

```

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

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

### 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))

### 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)

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

