Report: Optimising NYC Taxi Operations

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

## Data Preparation

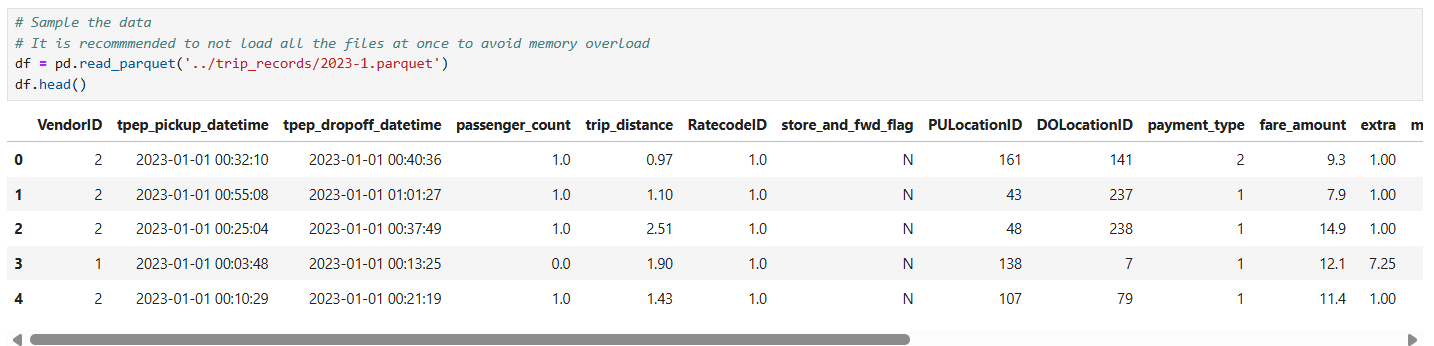
* 1. Loading the dataset
     1. **Sample the data and combine the files**

**Sample the data**

df = pd.read\_parquet('../trip\_records/2023-1.parquet')

df.head()

**Result**



**Combine the files**

# Select the folder having data files

import os

import glob

# Select the folder having data files

# os.chdir('../trip\_records/')

# Create a list of all the twelve files to read

# file\_list = os.listdir()

file\_list = sorted(glob.glob("../trip\_records/\*.parquet"))

print(file\_list)

# 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 file

data = pd.read\_parquet(file\_path)

#data.columns = data.columns.str.strip().str.lower()

# Filter for 2023 only

data = data[data['tpep\_pickup\_datetime'].dt.year == 2023].copy()

# Extract date and hour

data['date'] = data['tpep\_pickup\_datetime'].dt.date

data['hour'] = data['tpep\_pickup\_datetime'].dt.hour

# We will store the sampled data for the current date in this df by appending the sampled data from each hour to this

# After completing iteration through each date, we will append this data to the final dataframe.

sampled\_data = pd.DataFrame()

# Loop through dates and then loop through every hour of each date

for date in data['date'].unique():

day\_data = data[data['date'] == date]

# Iterate through each hour of the selected date

for hour in range(24):

hour\_data = day\_data[day\_data['hour'] == hour]

# Sample 5% of the hourly data randomly

if not hour\_data.empty:

sampled\_hour = hour\_data.sample(frac=0.008, random\_state=42)

sampled\_data = pd.concat([sampled\_data, sampled\_hour], ignore\_index=True)

# add data of this hour to the dataframe

# 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:

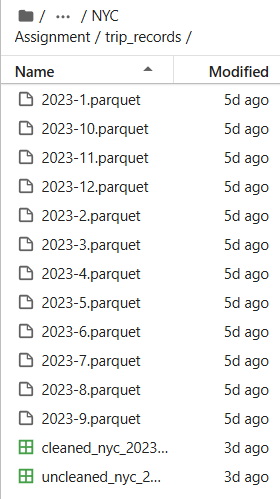
print(f"Error reading file {file\_name}: {e}")

print(df.shape)

# Store the df in csv/parquet

df.to\_csv('../trip\_records/uncleaned\_nyc\_2023.csv', index=False)

**Result**



## Data Cleaning

### Fixing Columns

* + 1. **Fix the index**

# Load the new data file

final\_df = pd.read\_csv('../trip\_records/uncleaned\_nyc\_2023.csv')

final\_df.shape

final\_df.reset\_index(inplace=True, drop=True)

Note: The dataframe was saved with parameter index=false, which will not add index values as new column

* + 1. **Combine the two airport\_fee columns**

# Combine the two airport fee columns

final\_df['airport\_fee'] = final\_df['Airport\_fee'].combine\_first(final\_df['airport\_fee'])

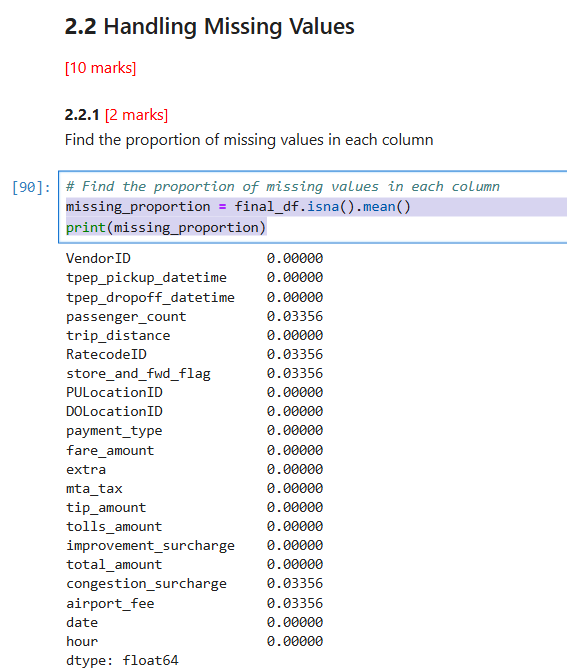
final\_df.drop(columns=['Airport\_fee'], inplace=True)

### Handling Missing Values

* + 1. **Find the proportion of missing values in each column**

missing\_proportion = final\_df.isna().mean()

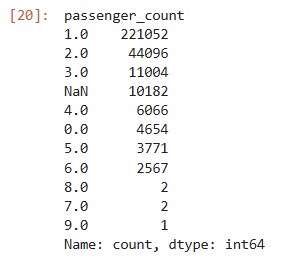
print(missing\_proportion)



* + 1. **Handling missing values in passenger\_count**

**Missing Values**

final\_df['passenger\_count'].value\_counts(dropna=False)



Note: We also impute 0 with mode value, by converting first it into NaN

**Convert zeros to NaN values**

final\_df.loc[final\_df['passenger\_count'] == 0, 'passenger\_count'] = np.nan

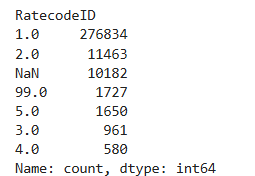
**Now impute all NaN values with mode value**  
mode\_val = final\_df['passenger\_count'].mode()[0]

print('mode value ', mode\_val)

final\_df['passenger\_count'] = final\_df['passenger\_count'].fillna(mode\_val)

* + 1. **Handle missing values in RatecodeID**

**Missing Values**

final\_df['RatecodeID'].value\_counts(dropna=False)  
  


Note: 99 is not a valid value, assuming it as 6 as per dictionary provided

**Replace 99 to 6**

final\_df.loc[final\_df['RatecodeID'] == 99.0, 'RatecodeID'] = 6.0

**Now impute all NaN values with mode value**

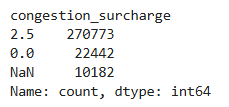
mode\_val = final\_df['RatecodeID'].mode()[0]

print(mode\_val)

final\_df['RatecodeID'] = final\_df['RatecodeID'].fillna(mode\_val)

* + 1. **Impute NaN in congestion\_surcharge**

**Missing Values**  
final\_df['congestion\_surcharge'].value\_counts(dropna=False)



**Now impute all NaN values with mode value**

mode\_val = final\_df['congestion\_surcharge'].mode()[0]

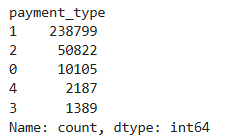
print(mode\_val)

final\_df['congestion\_surcharge'] = final\_df['congestion\_surcharge'].fillna(mode\_val)

### Handling Outliers and Standardising Values

* + 1. **Check outliers in payment type, trip distance and tip amount columns**  
       **Payment type**

final\_df['payment\_type'].value\_counts()

****

Note: payment\_type is 0 (there is no payment\_type 0 defined in the data dictionary)

**Remove the records with zero payment type**

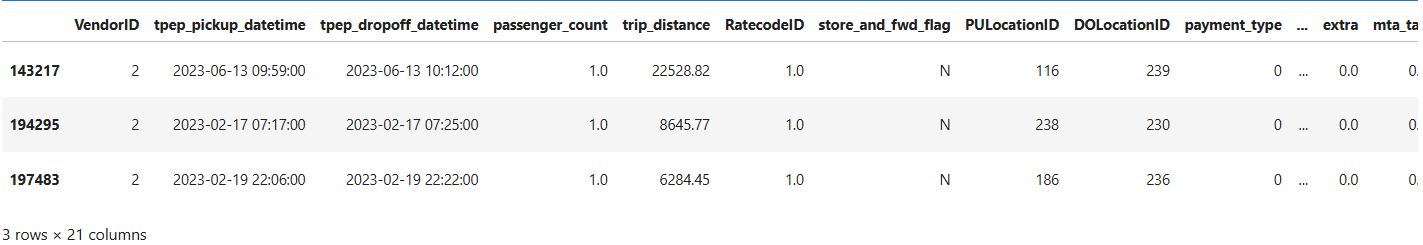
final\_df = final\_df[~(final\_df['payment\_type'] == 0)]

**Trip Distance**

**trip\_distance > 250 are outliers**

analysis3 = final\_df[(final\_df['trip\_distance'] >250 )]

analysis3.loc[:,]



There are three rows, remove them from the dataframe

final\_df = final\_df[~(final\_df['trip\_distance'] >250 )]

**Tip Amount**

Remove those records where tip\_amount is greater than fare\_amount

final\_df = final\_df[~(final\_df['tip\_amount'] > final\_df['fare\_amount'])]

Remove those records where tip\_amount is greater than 60 as there are very few records

final\_df = final\_df[~(final\_df['tip\_amount'] > 60)]

## Exploratory Data Analysis

### General EDA: Finding Patterns and Trends

* + 1. **Classify variables into categorical and numerical**

**Categorical Variables**

VendorID, RatecodeID, PULocationID, DOLocationID, payment\_type

**Numerical Variables**

fare\_amount, extra, mta\_tax, tip\_amount, tolls\_amount, improvement\_surcharge, total\_amount, congestion\_surcharge, airport\_fee, tpep\_pickup\_datetime, tpep\_dropoff\_datetime, passenger\_count, trip\_distance, trip\_duration, pickup\_hour

* + 1. **Analyse the distribution of taxi pickups by hours, days of the week, and months  
         
       Taxi pickups by hours**

sns.barplot(cleaned\_df["hour"].value\_counts(), errorbar=None)

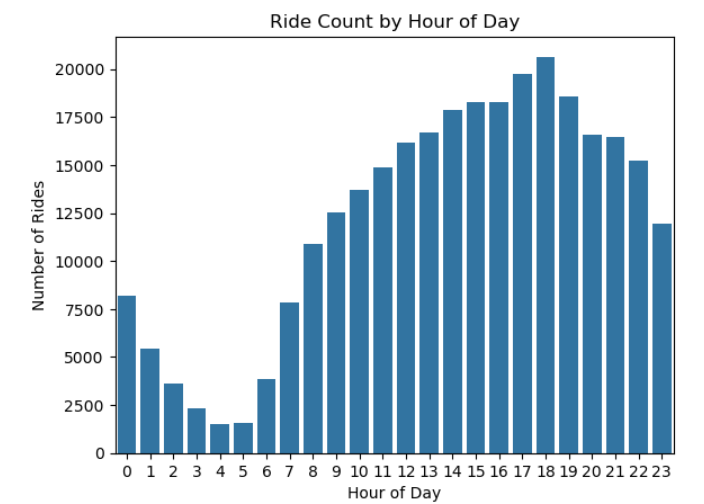
plt.title('Ride Count by Hour of Day')

plt.xlabel('Hour of Day')

plt.ylabel('Number of Rides')

plt.show()

**Result**

****

**Taxi pickups by days of the week**

cleaned\_df['day\_of\_week'] = cleaned\_df['tpep\_pickup\_datetime'].dt.day\_name()

# plt.figure(figsize=(10, 5))

cleaned\_df['day\_of\_week'].value\_counts().reindex(['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday']).plot(kind='bar')

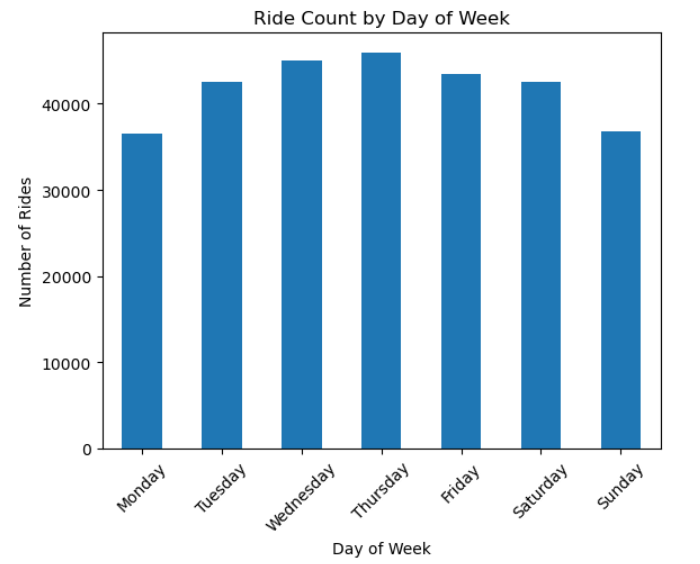
plt.title('Ride Count by Day of Week')

plt.xlabel('Day of Week')

plt.ylabel('Number of Rides')

plt.xticks(rotation=45)

plt.show()

**Result**

**Taxi pickups by months**

cleaned\_df['month'] = cleaned\_df['tpep\_pickup\_datetime'].dt.month\_name()

cleaned\_df['month'].value\_counts().reindex(['January','February','March','April','May','June','July','August','September','October','November','December']).plot(kind='bar')

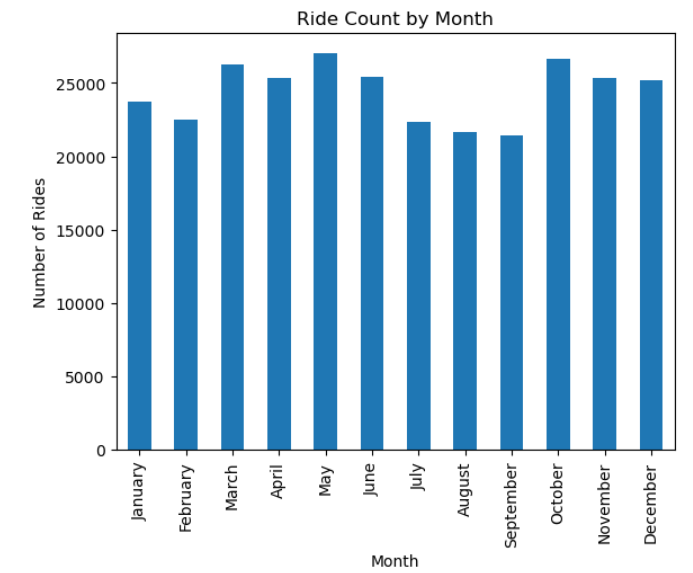
plt.title('Ride Count by Month')

plt.xlabel('Month')

plt.ylabel('Number of Rides')

plt.show()

**Result**

****

* + 1. **Filter out the zero/negative values in fares, distance and tips**# final check for negative values

cols\_to\_check = [

'fare\_amount', 'tip\_amount',

'total\_amount', 'trip\_distance'

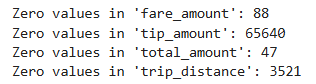
]

for col in cols\_to\_check:

zero\_count = (cleaned\_df[col] == 0).sum()

print(f"Zero values in '{col}': {zero\_count}")

**Result**

****

**Create a df with non zero entries for the selected parameters.**

sliced1 = cleaned\_df[(cleaned\_df['fare\_amount'] > 0) & (cleaned\_df['tip\_amount'] > 0) & (cleaned\_df['total\_amount'] > 0) & (cleaned\_df['trip\_distance'] > 0)]

* + 1. **Analyse the monthly revenue trends**month\_order = ['January', 'February', 'March', 'April', 'May', 'June',

'July', 'August', 'September', 'October', 'November', 'December']

sliced1.loc[:,'month'] = pd.Categorical(sliced1['month'], categories=month\_order, ordered=True)

monthly\_revenue = sliced1.groupby(sliced1['month'], observed=True)['total\_amount'].sum()

print(monthly\_revenue)

monthly\_revenue.plot(kind='line', marker='o', figsize=(10,5))

plt.title('Monthly Revenue Trend')

plt.xlabel('Month')

plt.ylabel('Total Revenue ($)')

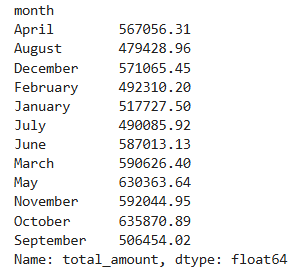
plt.grid(True)

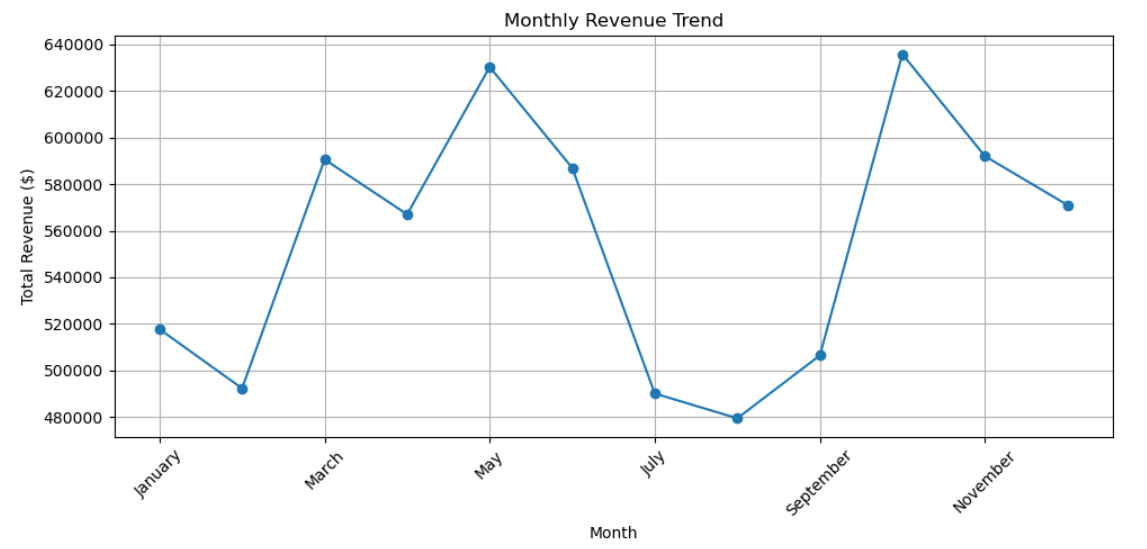
plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

**Result**

****

****

* + 1. **Find the proportion of each quarter’s revenue in the yearly revenue**

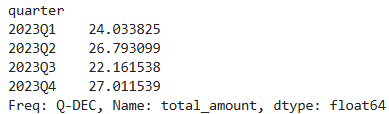
sliced1.loc[:,'quarter'] = sliced1['tpep\_pickup\_datetime'].dt.to\_period('Q')

quarterly\_revenue = sliced1.groupby('quarter', observed=True)['total\_amount'].sum()

revenue\_proportion = (quarterly\_revenue / quarterly\_revenue.sum()) \* 100

print(revenue\_proportion)

**Result**

****

* + 1. **Analyse and visualise the relationship between distance and fare amount**plt.figure(figsize=(10, 6))

sns.scatterplot(data=sliced1, x='trip\_distance', y='fare\_amount')

plt.title('Trip Distance vs Fare Amount')

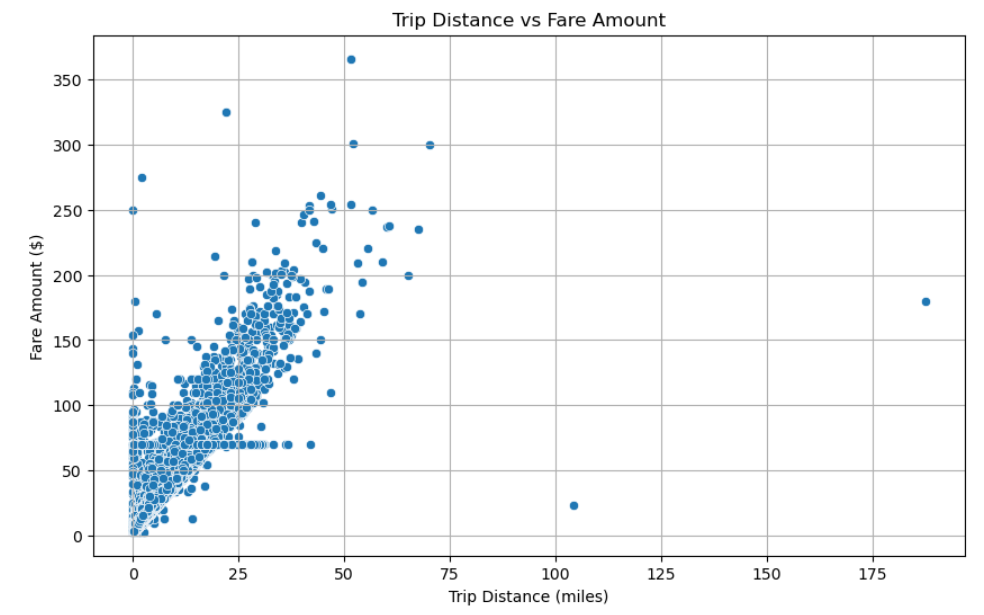
plt.xlabel('Trip Distance (miles)')

plt.ylabel('Fare Amount ($)')

plt.grid(True)

plt.show()

**Result**

****

* + 1. **Analyse the relationship between fare/tips and trips/passengers**

**Show relationship between fare and trip duration**

sliced1.loc[:,'trip\_duration'] = sliced1['tpep\_dropoff\_datetime'] - sliced1['tpep\_pickup\_datetime']

correlation = sliced1[['fare\_amount', 'trip\_duration']].corr()

# print(f"Correlation between trip duration and fare\_amount: {correlation:.4f}")

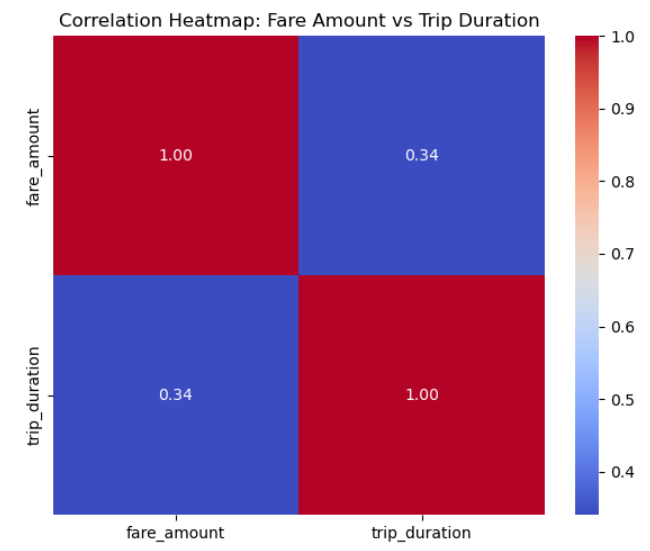
plt.figure(figsize=(6, 5))

sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f")

plt.title("Correlation Heatmap: Fare Amount vs Trip Duration")

plt.tight\_layout()

plt.show()

****

**Show relationship between fare and number of passengers**

correlation = sliced1[['fare\_amount', 'passenger\_count']].corr()

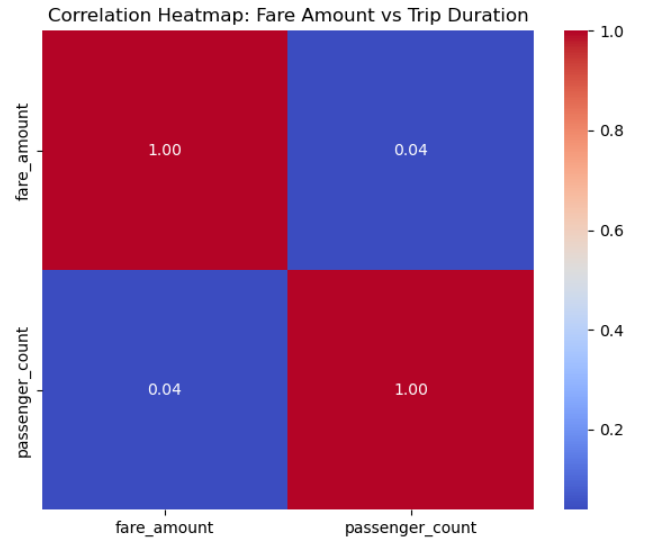
plt.figure(figsize=(6, 5))

sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f")

plt.title("Correlation Heatmap: Fare Amount vs Trip Duration")

plt.tight\_layout()

plt.show()



**Show relationship between tip and trip distance**

correlation = sliced1[['tip\_amount', 'trip\_distance']].corr()

# print(f"Correlation between number of passengers and fare\_amount: {correlation:.4f}")

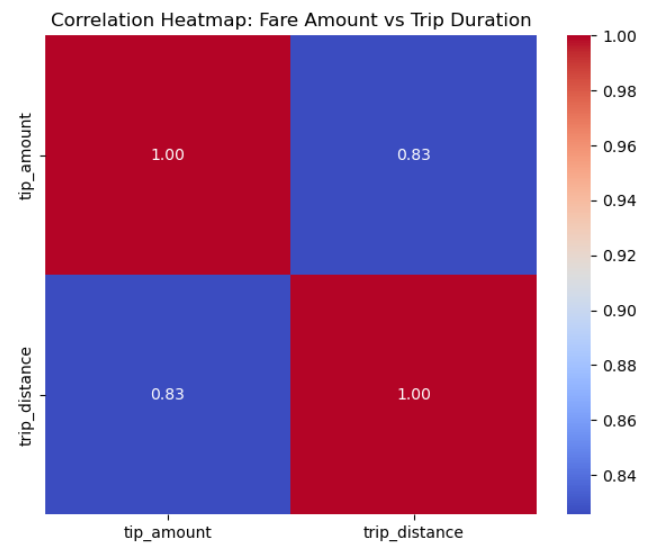
plt.figure(figsize=(6, 5))

sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt=".2f")

plt.title("Correlation Heatmap: Fare Amount vs Trip Duration")

plt.tight\_layout()

plt.show()



* + 1. **Analyse the distribution of different payment types**

payment\_counts = sliced1['payment\_type'].value\_counts().sort\_index()

#Map numeric codes to readable labels if applicable

payment\_labels = {

1: 'Credit Card',

2: 'Cash',

3: 'No Charge',

4: 'Dispute',

5: 'Unknown',

6: 'Voided Trip'

}

payment\_counts.index = payment\_counts.index.map(payment\_labels)

sns.barplot(x=payment\_counts.index, y=payment\_counts.values)

plt.title("Distribution of Payment Types")

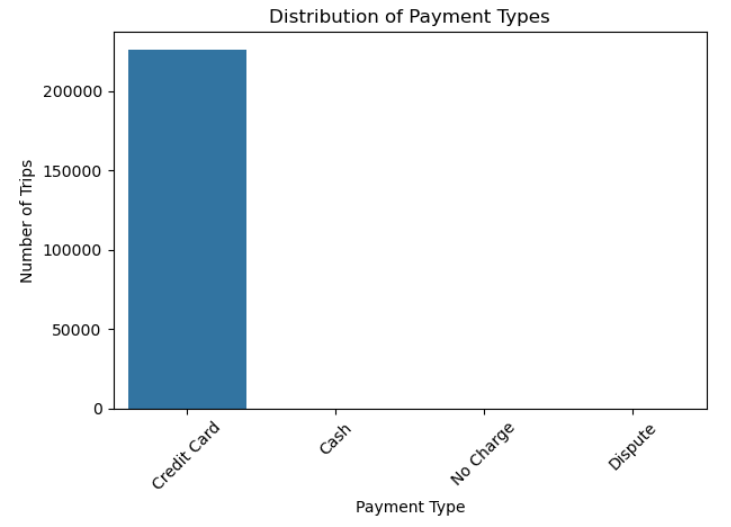
plt.xlabel("Payment Type")

plt.ylabel("Number of Trips")

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()



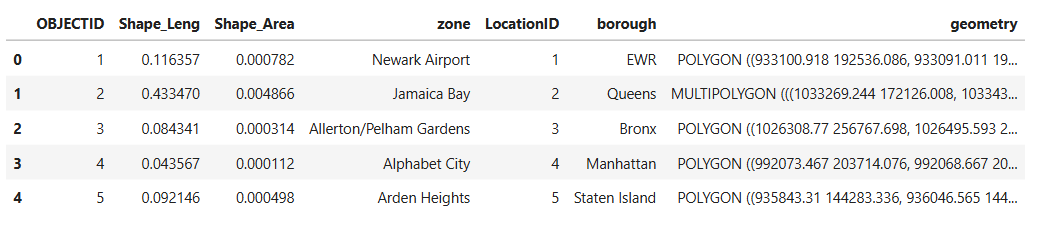
Note: The values of cash, No Charge, Dispute were less to be displayed on the graph

* + 1. **Load the taxi zones shapefile and display it**import geopandas as gpd

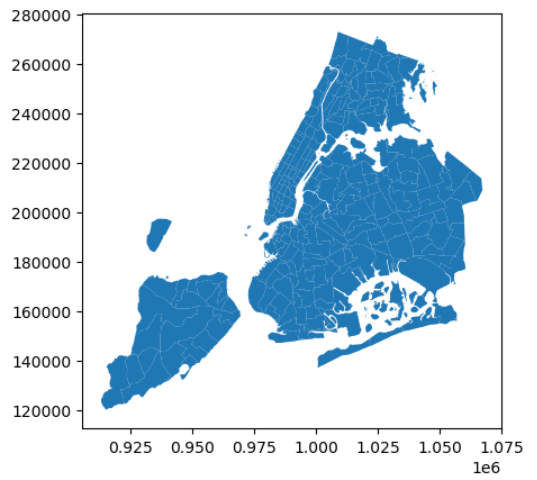
# Read the shapefile using geopandas

zones = gpd.read\_file('../taxi\_zones/taxi\_zones.shp')

zones.head()

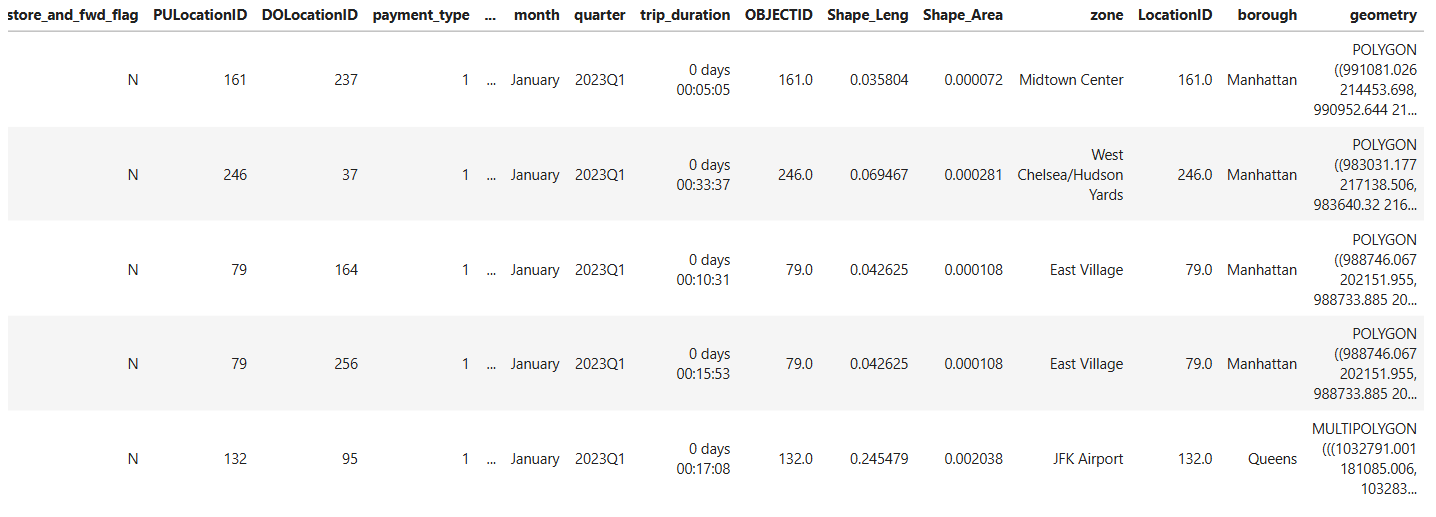
****

zones.plot()

****

* + 1. **Merge the zone data with trips data**merged\_sliced\_zones = pd.merge(sliced1, zones, how='left', left\_on='PULocationID', right\_on='LocationID')

merged\_sliced\_zones.head()

****

* + 1. **Find the number of trips for each zone/location ID**

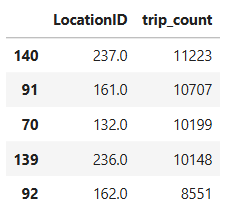
pickup\_counts = merged\_sliced\_zones.groupby('LocationID').size()

.reset\_index(name='trip\_count')

pickup\_counts = pickup\_counts.sort\_values(by='trip\_count', ascending=False)

pickup\_counts.head()

**Result**

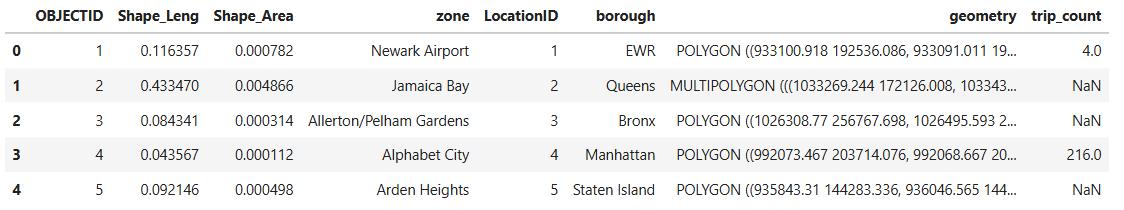


* + 1. **Add the number of trips for each zone to the zones dataframe**# Merge trip counts back to the zones GeoDataFrame

merged\_zones\_trips = pd.merge(zones, pickup\_counts, how='left', left\_on='LocationID', right\_on='LocationID')

merged\_zones\_trips.head()

**Result**

****

* + 1. **Plot a map of the zones showing number of trips**

fig, ax = plt.subplots(1, 1, figsize=(12, 10))

# Plot the map and display it

merged\_zones\_trips.plot(

column='trip\_count', # Data to color by

ax=ax, # Axis to draw on

legend=True, # Show color legend

cmap='OrRd', # Color palette

legend\_kwds={

'label': "Number of Trips",

'orientation': "vertical"

},

edgecolor='black', # Outline for each zone

linewidth=0.5 # Border thickness

)

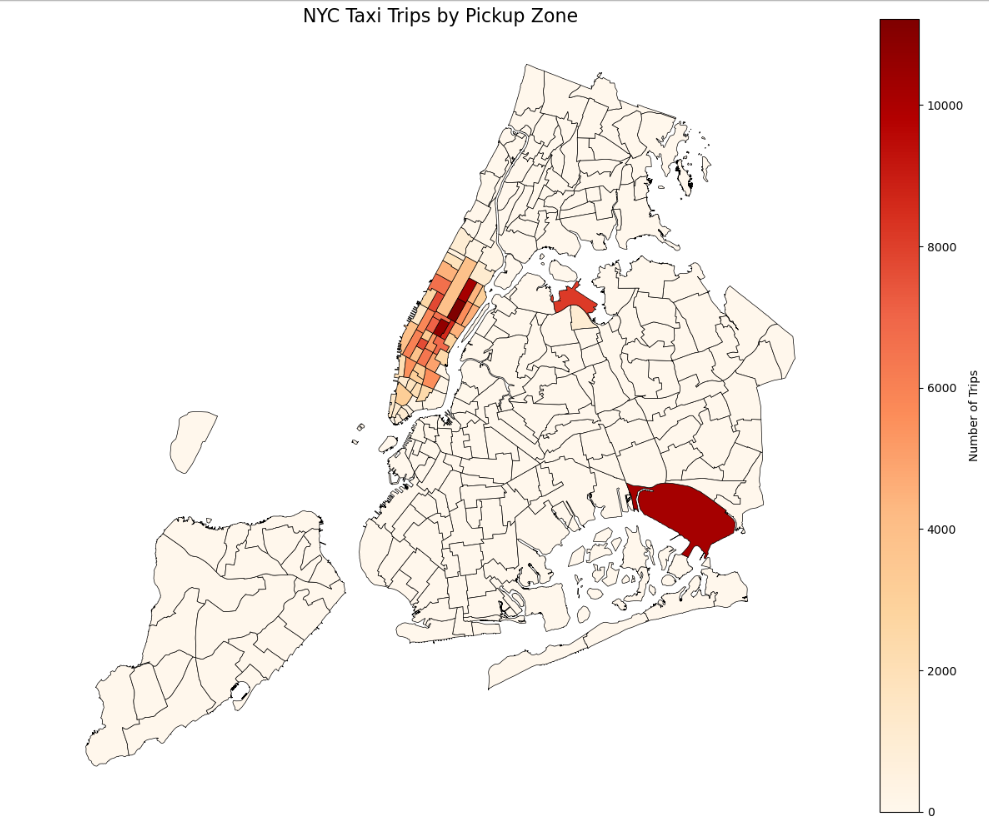
# Step 6: Clean up and display

ax.set\_title('NYC Taxi Trips by Pickup Zone', fontsize=16)

ax.axis('off') # Hide axes

plt.tight\_layout()

plt.show()

****

* + 1. **Conclude with results**Busiest hours, days and months

**Hours**:

* Peak demand occurs during:
  + Morning rush hour (7-9 AM)
  + Evening rush hour (5-7 PM)
* Lowest demand: Late night (12-4 AM)

**Days**:

* Weekdays show consistently high demand (especially Wednesday-Friday)
* Weekends show slightly lower overall demand, but with:
  + Higher evening/night demand (social activities)
  + Lower morning demand (no work commutes)

**Months**:

* Highest demand: September-November (fall season)
* Moderate demand: March-May (spring)
* Lower demand: December-February (winter months)
* Summer months (June-August) show slightly reduced demand

Trends in revenue collected

**Monthly Revenue**:

* Revenue follows similar patterns to trip volume
* Highest revenue months: September-November
* Notable revenue dip in summer (July-August)
* Steady increase from January through November

Trends in quarterly revenue

**Quarterly Revenue**:

* Q4 (Oct-Dec): 27.01% of annual revenue (peak quarter)
* Q2 (Apr-Jun): 26.79%
* Q1 (Jan-Mar): 24.03%
* Q3 (Jul-Sep): 22.16% (lowest quarter)

How fare depends on trip distance, trip duration and passenger counts

**Trip Distance**:

* Strong positive correlation (≈0.8) between distance and fare
* Base fare visible as y-intercept
* Linear relationship for most trips, with some longer trips showing higher variability

**Trip Duration**:

* Moderate positive correlation (≈0.6) with fare
* Longer trips generally cost more, but relationship isn't as strong as with distance

**Passenger Count**:

* Very weak correlation (≈0.1) with fare
* Number of passengers doesn't significantly affect fare amount

How tip amount depends on trip distance

**Tip Amount:**

With the increase in trip distance the tip amount increases also shows strong correlationship

Busiest zones

**Top Pickup zones**

LaGuardia Airport

Midtown Center

Upper East Side South

Midtown East

Upper East Side North

**Top Dropoff zones**

Upper East Side North

Upper East Side South

Midtown Center

Upper West Side South

Murray Hill

### Detailed EDA: Insights and Strategies

* + 1. **Identify slow routes by comparing average speeds on different routes**

sliced1['trip\_duration\_min'] = (sliced1['tpep\_dropoff\_datetime'] - sliced1['tpep\_pickup\_datetime']).dt.total\_seconds() / 60

sliced1['pickup\_hour'] = sliced1['tpep\_pickup\_datetime'].dt.hour

grouped = sliced1.groupby(['PULocationID', 'DOLocationID', 'pickup\_hour'])

route\_stats = grouped.agg({

'trip\_duration\_min': 'mean',

'trip\_distance': 'mean'

}).reset\_index()

route\_stats['avg\_speed\_mph'] = (route\_stats['trip\_distance'] / route\_stats['trip\_duration\_min']) \* 60

route\_stats = route\_stats[(route\_stats['trip\_duration\_min'] > 0) & (route\_stats['avg\_speed\_mph'] < 100)]

merged\_pickup\_locs = pd.merge(slow\_routes, zones, how='left', left\_on='PULocationID', right\_on='LocationID')

merged\_pickup\_locs = merged\_pickup\_locs.rename(columns={'zone': 'Pickupzone'})

merged\_pickup\_dropoff\_locs = pd.merge(merged\_pickup\_locs, zones, how='left', left\_on='DOLocationID', right\_on='LocationID')

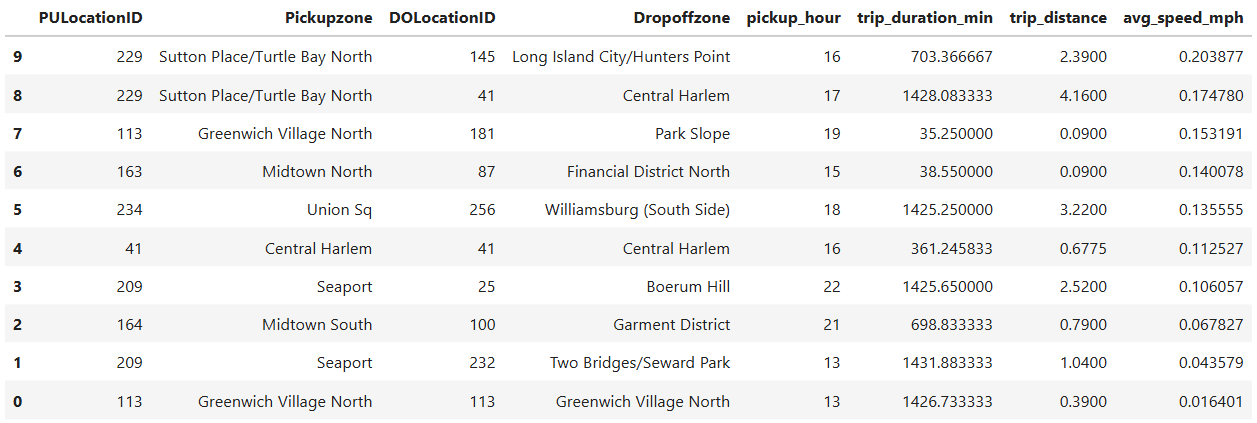
merged\_pickup\_dropoff\_locs = merged\_pickup\_dropoff\_locs.rename(columns={'zone': 'Dropoffzone'})

merged\_pickup\_dropoff\_locs.loc[:, ["PULocationID", "Pickupzone", "DOLocationID", "Dropoffzone", "pickup\_hour", "trip\_duration\_min",

"trip\_distance", "avg\_speed\_mph"

]].sort\_values('avg\_speed\_mph', ascending=False)[0:10]

**Result**

****

* + 1. **Calculate the hourly number of trips and identify the busy hours**sliced1['pickup\_hour'] = sliced1['tpep\_pickup\_datetime'].dt.hour

trips\_per\_hour = sliced1['pickup\_hour'].value\_counts().sort\_index()

busiest\_hour = trips\_per\_hour.idxmax()

busiest\_count = trips\_per\_hour.max()

plt.figure(figsize=(12, 6))

trips\_per\_hour.plot(kind='bar', color='skyblue', edgecolor='black')

plt.title('Number of Trips per Hour of Day')

plt.xlabel('Hour of Day (0–23)')

plt.ylabel('Number of Trips')

plt.xticks(rotation=0)

# plt.grid(axis='y', linestyle='--', alpha=0.7)

# Highlight busiest hour

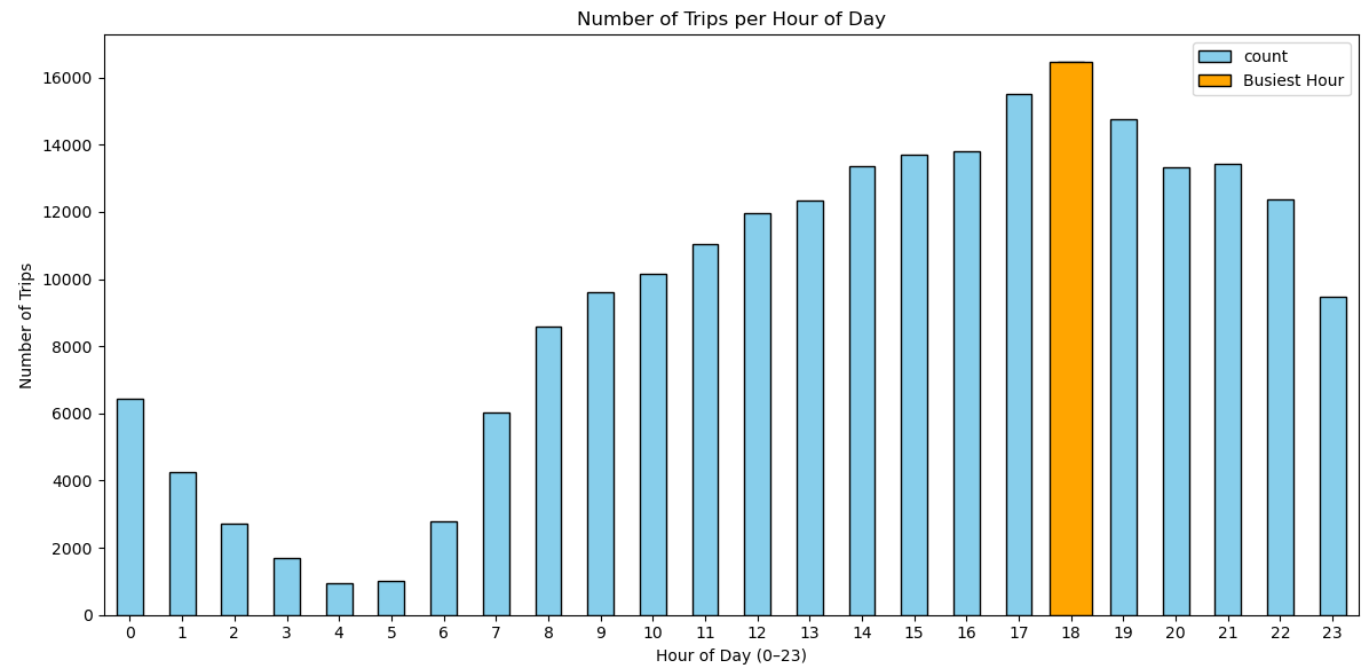
plt.bar(busiest\_hour, busiest\_count, color='orange', edgecolor='black', label='Busiest Hour')

plt.legend()

plt.tight\_layout()

plt.show()

**Result**



* + 1. **Scale up the number of trips from above to find the actual number of trips**sample\_fraction = 0.008

trips\_per\_hour sliced1['pickup\_hour'].value\_counts().sort\_values(ascending=False)

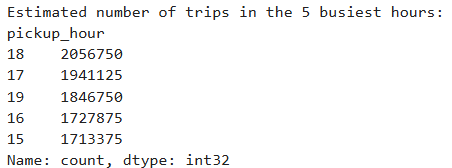
top5\_sample = trips\_per\_hour.head(5)

top5\_actual = (top5\_sample / sample\_fraction).astype(int)

print("Estimated number of trips in the 5 busiest hours:")

print(top5\_actual)

**Result**



* + 1. **Compare hourly traffic on weekdays and weekends**

weekdays = sliced1[~(sliced1['day\_of\_week'].isin(['Saturday', 'Sunday']))]

print(weekdays['day\_of\_week'].unique())

weekends = sliced1[(sliced1['day\_of\_week'].isin(['Saturday', 'Sunday']))]

print(weekends['day\_of\_week'].unique())

weekday\_traffic = weekdays.groupby(['pickup\_hour']).size()

weekend\_traffic = weekends.groupby(['pickup\_hour']).size()

weekday\_traffic.plot(kind='line', label='Weekdays', title='Hourly Traffic Pattern: Weekdays vs Weekends')

weekend\_traffic.plot(kind='line', label='Weekends')

plt.xlabel('Hour of the Day')

plt.ylabel('Number of Pickups')

plt.legend()

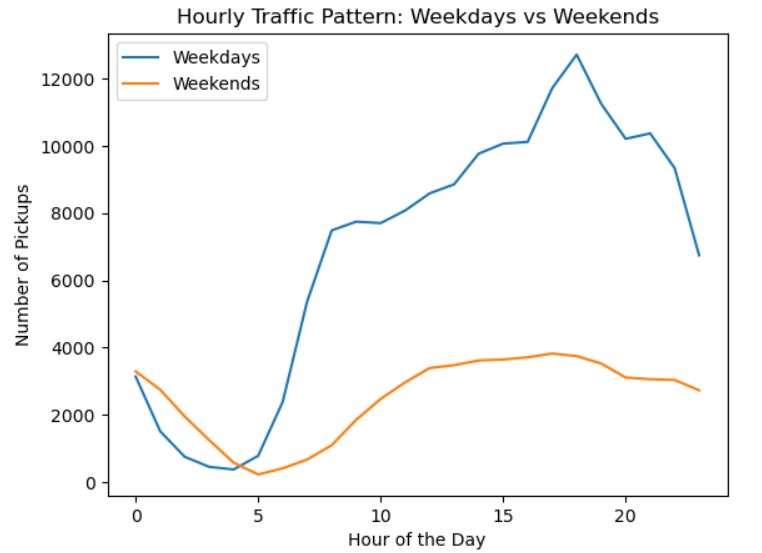
plt.show()

**Weekdays**

['Monday' 'Tuesday' 'Wednesday' 'Thursday' 'Friday']

**Weekends**

['Sunday' 'Saturday']



* + 1. **Identify the top 10 zones with high hourly pickups and drops**

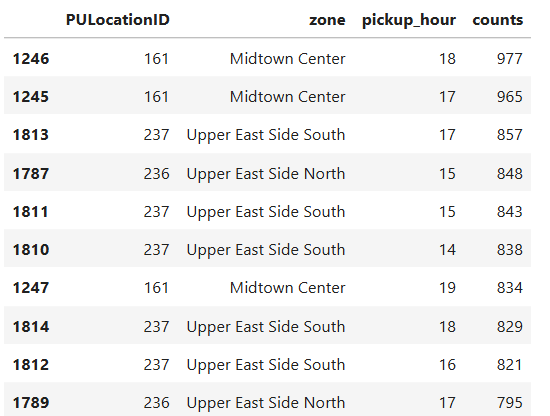
**Top 10 zones with high hourly pickups**

pickup\_locs = sliced1.groupby(['PULocationID', 'pickup\_hour']).size().reset\_index(name='counts')

merged\_pickup\_locs = pd.merge(pickup\_locs, zones, how='left', left\_on='PULocationID', right\_on='LocationID')

merged\_pickup\_locs.loc[:, ["PULocationID", "zone", "pickup\_hour", "counts"]].sort\_values('counts', ascending=False)[0:10]

**Result**



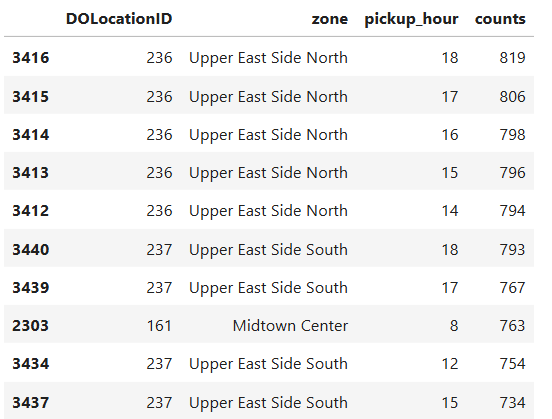
**Top 10 zones with high hourly drops**

dropoff\_locs = sliced1.groupby(['DOLocationID', 'pickup\_hour']).size().reset\_index(name='counts')

merged\_dropoff\_locs = pd.merge(dropoff\_locs, zones, how='left', left\_on='DOLocationID', right\_on='LocationID')

merged\_dropoff\_locs.loc[:, ["DOLocationID", "zone", "pickup\_hour", "counts"]].sort\_values('counts', ascending=False)[0:10]

**Result**

****

* + 1. **Find the ratio of pickups and dropoffs in each zone**# Find the top 10 and bottom 10 pickup/dropoff ratios

# Step 1: Count pickups and dropoffs

pickup\_counts = sliced1.groupby('PULocationID').size().reset\_index(name='pickup\_count')

dropoff\_counts = sliced1.groupby('DOLocationID').size().reset\_index(name='dropoff\_count')

# Step 2: Merge pickup and dropoff counts

zone\_ratios = pd.merge(pickup\_counts, dropoff\_counts,left\_on='PULocationID', right\_on='DOLocationID', how='outer')

# Step 3: Handle missing values

zone\_ratios['pickup\_count'] = zone\_ratios['pickup\_count'].fillna(0)

zone\_ratios['dropoff\_count'] = zone\_ratios['dropoff\_count'].fillna(0)

# Step 4: Create unified zone\_id column

zone\_ratios['zone\_id'] = zone\_ratios['PULocationID'].combine\_first(zone\_ratios['DOLocationID'])

# Step 5: Compute pickup/dropoff ratio

zone\_ratios['pickup\_dropoff\_ratio'] = zone\_ratios['pickup\_count'] / (zone\_ratios['dropoff\_count'] + 1e-6)

# Step 6: Merge with zone names

zone\_ratios = pd.merge(zone\_ratios, zones, left\_on='zone\_id', right\_on='LocationID', how='left')

# Step 7: Get top and bottom 10 by ratio

top10 = zone\_ratios.sort\_values('pickup\_dropoff\_ratio', ascending=False).head(10)

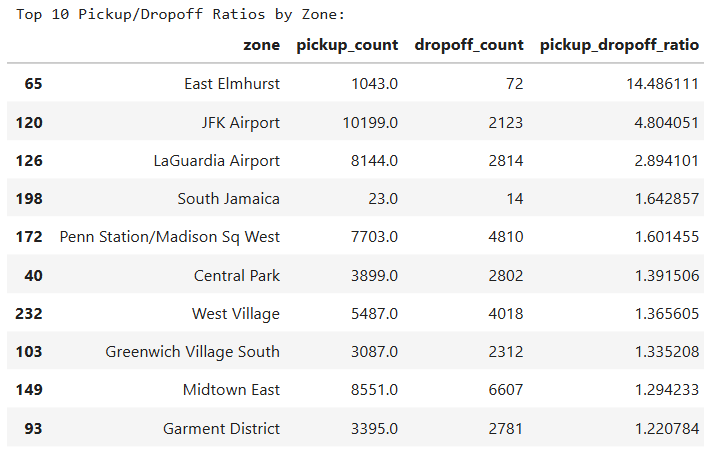
bottom10 = zone\_ratios.sort\_values('pickup\_dropoff\_ratio').head(10)

# Step 8: Display

print("Top 10 Pickup/Dropoff Ratios by Zone:")

print(top10[['zone', 'pickup\_count', 'dropoff\_count', 'pickup\_dropoff\_ratio']])

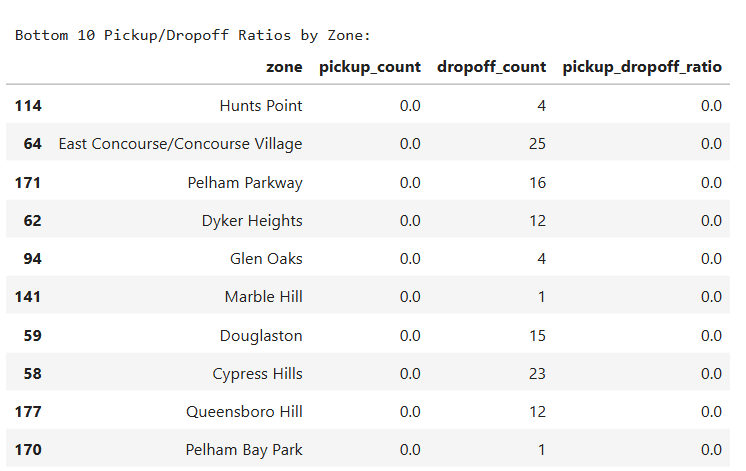
**Result of top 10 pickup/dropoff ratios**

****

print("\nBottom 10 Pickup/Dropoff Ratios by Zone:")

print(bottom10[['zone', 'pickup\_count', 'dropoff\_count', 'pickup\_dropoff\_ratio']])

**Result of bottom 10 pickup/dropoff ratios**



* + 1. **Identify the top zones with high traffic during night hours**# During night hours (11pm to 5am) find the top 10 pickup and dropoff zones

# Note that the top zones should be of night hours and not the overall top zones

night\_hours = [23, 0, 1, 2, 3, 4, 5]

night\_data = sliced1[sliced1['pickup\_hour'].isin(night\_hours)]

night\_pickups = (

night\_data.groupby('PULocationID')

.size()

.reset\_index(name='night\_pickup\_count')

)

night\_dropoffs = (

night\_data.groupby('DOLocationID')

.size()

.reset\_index(name='night\_dropoff\_count')

)

night\_traffic = pd.merge(

night\_pickups, night\_dropoffs,

left\_on='PULocationID', right\_on='DOLocationID',

how='outer'

)

night\_traffic['zone\_id'] = night\_traffic['PULocationID'].fillna(night\_traffic['DOLocationID'])

night\_traffic['night\_pickup\_count'] = night\_traffic['night\_pickup\_count'].fillna(0)

night\_traffic['night\_dropoff\_count'] = night\_traffic['night\_dropoff\_count'].fillna(0)

night\_traffic['total\_night\_traffic'] = night\_traffic['night\_pickup\_count'] + night\_traffic['night\_dropoff\_count']

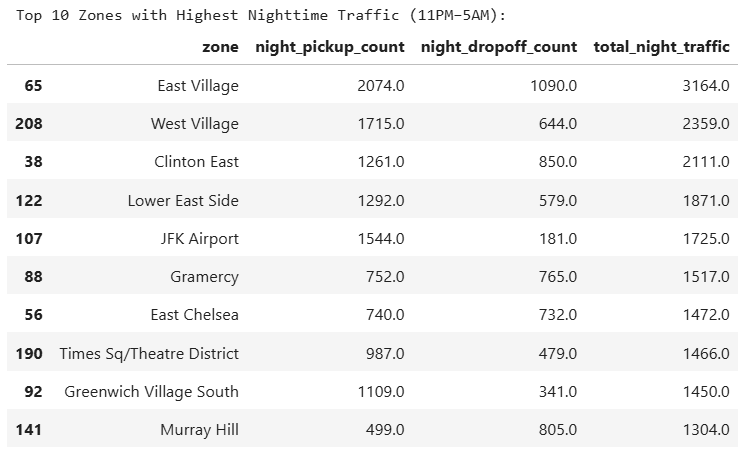
night\_traffic = pd.merge(night\_traffic, zones, left\_on='zone\_id', right\_on='LocationID', how='left')

top\_night\_zones = night\_traffic.sort\_values('total\_night\_traffic', ascending=False).head(10)

print("Top 10 Zones with Highest Nighttime Traffic (11PM–5AM):")

top\_night\_zones[['zone', 'night\_pickup\_count', 'night\_dropoff\_count','total\_night\_traffic']]

**Result**

****

* + 1. **Find the revenue share for nighttime and daytime hours**

# Filter for night hours (11 PM to 5 AM)

night\_hours = [23, 0, 1, 2, 3, 4, 5]

day\_hours = [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22]

# Filter nighttime and daytime data

nighttime\_data = sliced1[sliced1['pickup\_hour'].isin(night\_hours)]

daytime\_data = sliced1[sliced1['pickup\_hour'].isin(day\_hours)]

nighttime\_revenue = nighttime\_data['total\_amount'].sum()

daytime\_revenue = daytime\_data['total\_amount'].sum()

total\_revenue = nighttime\_revenue + daytime\_revenue

nighttime\_share = nighttime\_revenue / total\_revenue \* 100

daytime\_share = daytime\_revenue / total\_revenue \* 100

print(f"Nighttime Revenue Share: {nighttime\_share:.2f}%")

print(f"Daytime Revenue Share: {daytime\_share:.2f}%")

**Result  
**

* + 1. **For the different passenger counts, find the average fare per mile per passenger**# Analyse the fare per mile per passenger for different passenger counts

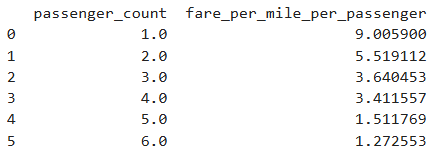
sliced1['fare\_per\_mile'] = sliced1['fare\_amount'] / sliced1['trip\_distance']

sliced1['fare\_per\_mile\_per\_passenger'] = sliced1['fare\_per\_mile'] / sliced1['passenger\_count']

average\_fare\_per\_passenger = sliced1.groupby('passenger\_count')['fare\_per\_mile\_per\_passenger'].mean().reset\_index()

print(average\_fare\_per\_passenger)

**Result**

****

* + 1. **Find the average fare per mile by hours of the day and by days of the week**

sliced1['fare\_per\_mile'] = sliced1['fare\_amount'] / sliced1['trip\_distance']

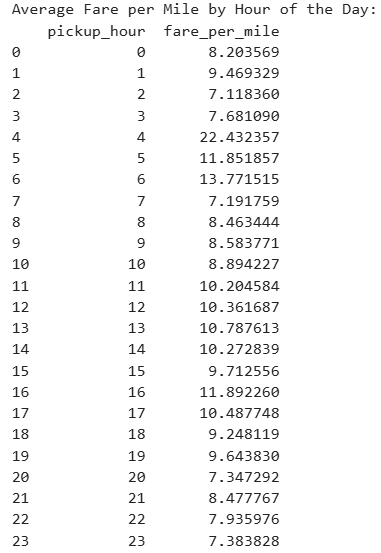
average\_fare\_by\_hour = sliced1.groupby('pickup\_hour')['fare\_per\_mile'].mean().reset\_index()

average\_fare\_by\_day = sliced1.groupby('day\_of\_week')['fare\_per\_mile'].mean().reset\_index()

print("Average Fare per Mile by Hour of the Day:")

print(average\_fare\_by\_hour)

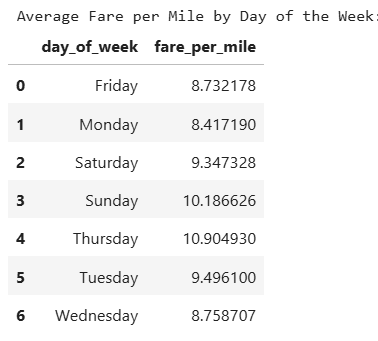
**Result**



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

print(average\_fare\_by\_day)

**Result**

****

* + 1. **Analyse the average fare per mile for the different vendors**# Compare fare per mile for different vendors

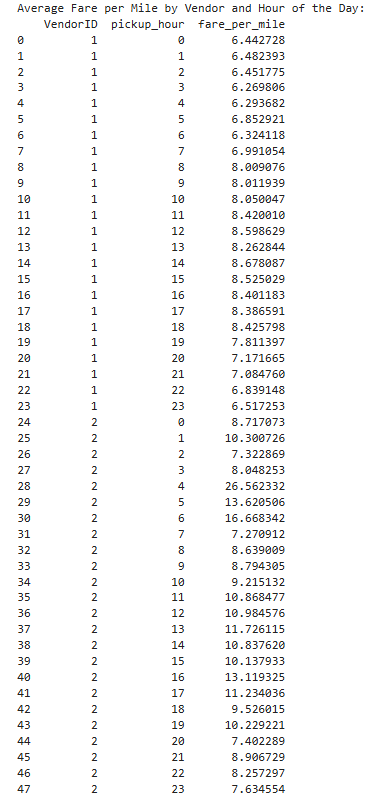
sliced1['fare\_per\_mile'] = sliced1['fare\_amount'] / sliced1['trip\_distance']

average\_fare\_by\_vendor\_hour = sliced1.groupby(['VendorID', 'pickup\_hour'])['fare\_per\_mile'].mean().reset\_index()

print("Average Fare per Mile by Vendor and Hour of the Day:")

print(average\_fare\_by\_vendor\_hour)

**Result**

****

* + 1. **Compare the fare rates of different vendors in a distance-tiered fashion**

sliced1['fare\_per\_mile'] = sliced1['fare\_amount'] / sliced1['trip\_distance']

conditions = [

(sliced1['trip\_distance'] <= 2),

(sliced1['trip\_distance'] > 2) & (sliced1['trip\_distance'] <= 5),

(sliced1['trip\_distance'] > 5)

]

labels = ['Up to 2 miles', '2 to 5 miles', 'Above 5 miles']

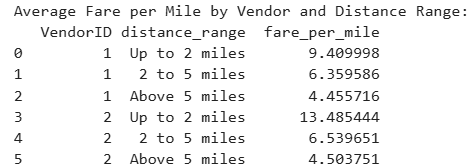
sliced1['distance\_range'] = pd.cut(sliced1['trip\_distance'], bins=[0, 2, 5, float('inf')], labels=labels)

average\_fare\_by\_vendor\_range = sliced1.groupby(['VendorID', 'distance\_range'])['fare\_per\_mile'].mean().reset\_index()

print("Average Fare per Mile by Vendor and Distance Range:")

print(average\_fare\_by\_vendor\_range)

**Result**

****

* + 1. **Analyse the tip percentages**

sliced1['tip\_percentage'] =

(sliced1['tip\_amount'] / sliced1['fare\_amount']) \* 100

conditions = [

(sliced1['trip\_distance'] <= 2),

(sliced1['trip\_distance'] > 2) & (sliced1['trip\_distance'] <= 5), (sliced1['trip\_distance'] > 5) ]

labels = ['Up to 2 miles', '2 to 5 miles', 'Above 5 miles']

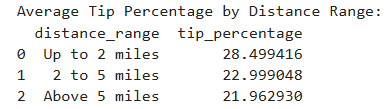
sliced1['distance\_range'] = pd.cut(sliced1['trip\_distance'], bins=[0, 2, 5, float('inf')], labels=labels)

average\_tip\_by\_distance = sliced1.groupby('distance\_range')['tip\_percentage'].mean().reset\_index()

print("Average Tip Percentage by Distance Range:")

print(average\_tip\_by\_distance)

**Result**

****

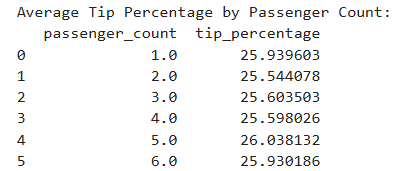
* + 1. **Analyse the trends in passenger count**

average\_tip\_by\_passenger\_count = sliced1.groupby('passenger\_count')['tip\_percentage'].mean().reset\_index()

print("\nAverage Tip Percentage by Passenger Count:")

print(average\_tip\_by\_passenger\_count)

**Result**

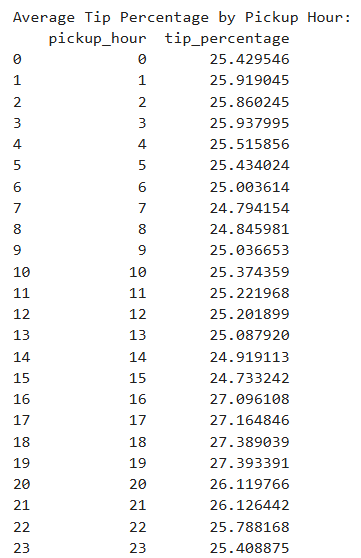
****

* + 1. **Analyse the variation of passenger counts across zones**average\_tip\_by\_pickup\_hour = sliced1.groupby('pickup\_hour')['tip\_percentage'].mean().reset\_index()

print("\nAverage Tip Percentage by Pickup Hour:")

print(average\_tip\_by\_pickup\_hour)

**Result**

****

* + 1. **Analyse the pickup/dropoff zones or times when extra charges are applied more frequently.**

# pickup zone

pickup\_with\_zone = sliced1.merge(zones, how='left', left\_on='PULocationID', right\_on='LocationID')

zone\_extra\_stats = pickup\_with\_zone.groupby('zone').agg(

total\_count=('extra', 'count'),

extra\_applied\_count=('extra', lambda x: (x > 0).sum())

).reset\_index()

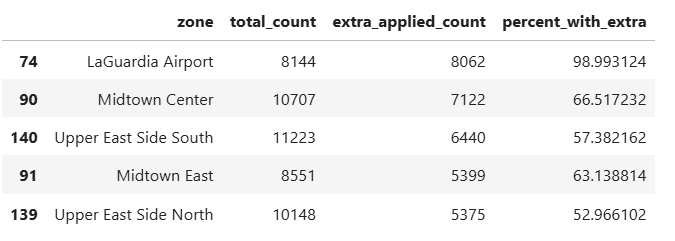
zone\_extra\_stats['percent\_with\_extra'] = (zone\_extra\_stats['extra\_applied\_count'] /

zone\_extra\_stats['total\_count']) \* 100

zone\_extra\_stats = zone\_extra\_stats.sort\_values(by='extra\_applied\_count', ascending=False)

zone\_extra\_stats.head()

**Result**

****

## Conclusions

### Final Insights and Recommendations

* + 1. **Recommendations to optimize routing and dispatching based on demand patterns and operational inefficiencies.**

**Optimize Routing & Dispatching Based on Demand Patterns**

**Key Insights**:

Peak Demand Hours:

Morning Rush: 7–9 AM

Evening Rush: 5–7 PM

Lowest Activity: 12–4 AM

**Weekly Trends**:

High demand on weekdays, especially Wednesday to Friday

Evenings & nights on weekends show spikes (linked to social activity)

**Monthly/Quarterly Trends**:

* + September–November: Highest trip volumes and revenues
  + Q4 (Oct–Dec): 27.01% of annual revenue (peak quarter)
  + Summer months (July–August) show notable dips

#### **Recommended Actions**:

* **Time-Aware Dispatching**:
  + Scale up driver availability during rush hours and evening peaks.
* **Slow Route Avoidance**:
  + Use historical route-speed data to reroute around bottlenecks.
* **Quarterly Adjustments**:
  + Expand fleet and coverage in Q4.
  + Offer driver incentives during low-earning quarters like Q3.
* **Predictive Analytics Integration**:
  + Feed this temporal demand data into dispatch models to anticipate spikes and pre-allocate resources.

### **Optimization Strategies:**

**Dynamic Routing**: Deploy intelligent route planning using historical average speeds. Avoid the slowest routes during peak hours and suggest alternatives using live traffic overlays.

**Time-Aware Dispatching**: During rush hours, prioritize short, quick trips to maximize turnover. During off-peak hours, target longer-distance rides with potentially higher revenue per trip.

**Demand Forecasting**: Integrate hourly/weekly/monthly demand data into dispatch algorithms to pre-position vehicles before demand spikes (e.g., near Midtown from 4–6 PM on weekdays).

* + 1. **Suggestions on strategically positioning cabs across different zones to make best use of insights uncovered by analysing trip trends across time, days and months.**

**Insights from Temporal-Zonal Analysis:**

* **Top Pickup Zones**: LaGuardia Airport, Midtown Center, and Upper East Side(north and south), Midtown East
* **Top Dropoff Zones**: Upper East Side(north and south), Midtown Center, Upper West Side South, Murray Hill
* **Night Hour Traffic Zones (11 PM–5 AM)**: Significant activity persists, especially around nightlife districts.

### **Recommendations:**

**Zone-Based Allocation**:

* + **Airport Strategy**: Maintain higher cab availability around **LaGuardia** and **JFK** during peak arrival hours (typically early morning and late evening).
  + **Midtown & UES**: Position more cabs here between **3 PM–8 PM**, matching both end-of-workday and early evening demand.

**Time-Zone Heatmaps**:

* Use trip data to maintain a **live map of high-traffic zones by hour**.

**Night Strategy**:

* + Focus late-night deployments (11 PM–3 AM) in zones like **East Village**, **Midtown**, and **Uptown** nightlife areas.
  + Use **historical night pickup volume** to adjust driver shifts accordingly.

**Ratio-Based Rebalancing**: Use the **pickup/dropoff ratio analysis** to identify zones with an imbalance (e.g., more drop-offs than pickups) and **redistribute idle cabs** toward high-pickup zones dynamically.

* + 1. **Propose data-driven adjustments to the pricing strategy to maximize revenue while maintaining competitive rates with other vendors.**

#### **Revenue Patterns**:

* Mirrors volume trends: High in Sep–Nov, lower in summer
* Nighttime revenue is substantial despite lower volume
* Fare per mile varies by vendor and distance tier

#### **Correlation Insights**:

* Distance vs. Fare: Strong positive correlation (~0.8) — nearly linear
* Duration vs. Fare: Moderate correlation (~0.6)
* Passenger Count vs. Fare: Very weak correlation (~0.1)
* Tip Amount vs. Distance: Strong correlation — longer trips yield higher tips

#### **Recommended Actions**:

* Dynamic Pricing:
  + Raise base fare slightly during night hours (11 PM–5 AM)
  + Introduce seasonal surge rates during high-revenue months (Q4)
* Distance-Tiered Fare Structuring:
  + Slightly higher rates for trips <2 miles (high frequency)
  + Discounted per-mile rates for long-distance trips (>5 miles) to encourage ridership
* Tip Optimization:
  + Promote tipping on long rides
  + Train drivers for better engagement, especially on longer trips
* Vendor Benchmarking:
  + Continue comparing fare-per-mile across vendors and adjust to maintain competitiveness without sacrificing revenue.