Report: Optimising NYC Taxi Operations

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

1. Data Preparation

1.1. Loading the dataset

1.1.1. Sample the data and combine the files

Sample the data

df = pd.read_parquet('../trip_records/2023-1.parquet')
df.head()

Result

```
# Sample the data
# It is recommmended to not load all the files at once to avoid memory overload
df = pd.read_parquet('../trip_records/2023-1.parquet')
df.head()
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID
0	2	2023-01-01 00:32:10	2023-01-01 00:40:36	1.0	0.97	1.0	N	161
1	2	2023-01-01 00:55:08	2023-01-01 01:01:27	1.0	1.10	1.0	N	43
2	2	2023-01-01 00:25:04	2023-01-01 00:37:49	1.0	2.51	1.0	N	48
3	1	2023-01-01 00:03:48	2023-01-01 00:13:25	0.0	1.90	1.0	N	138
4	2	2023-01-01 00:10:29	2023-01-01 00:21:19	1.0	1.43	1.0	N	107

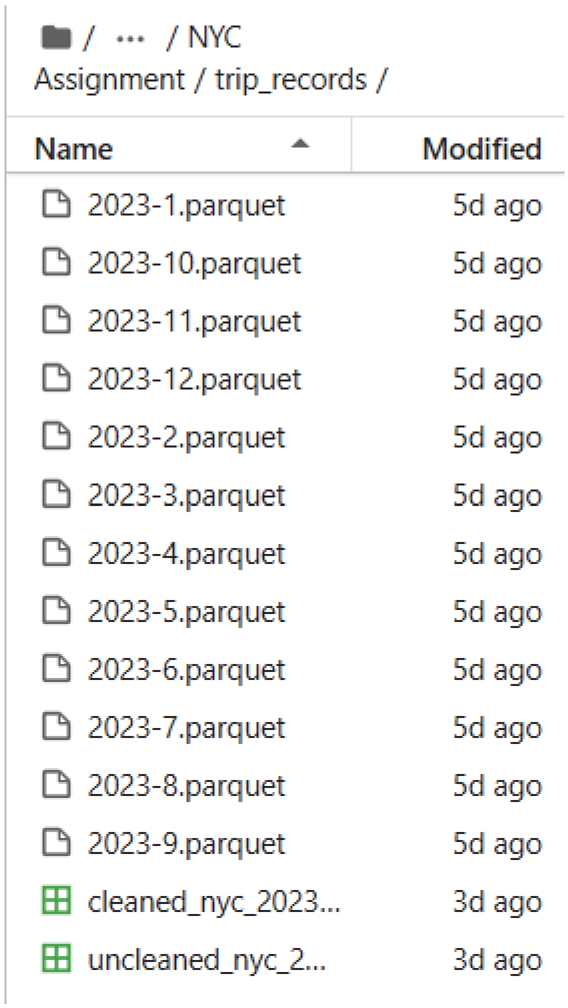
Combine the files

```
# 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



2. Data Cleaning

2.1. Fixing Columns

2.1.1. 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

2.1.2. 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)
```

2.2. Handling Missing Values

2.2.1. Find the proportion of missing values in each column

```
missing_proportion = final_df.isna().mean()
print(missing_proportion)
```

```
[90]: # Find the proportion of missing values in each column
      missing_proportion = final_df.isna().mean()
      print(missing_proportion)

VendorID                 0.00000
tpep_pickup_datetime     0.00000
tpep_dropoff_datetime    0.00000
passenger_count          0.03356
trip_distance            0.00000
RatecodeID               0.03356
store_and_fwd_flag       0.03356
PULocationID             0.00000
DOLocationID             0.00000
payment_type             0.00000
fare_amount              0.00000
extra                    0.00000
mta_tax                  0.00000
tip_amount               0.00000
tolls_amount             0.00000
improvement_surcharge    0.00000
total_amount             0.00000
congestion_surcharge     0.03356
airport_fee              0.03356
date                     0.00000
hour                     0.00000
dtype: float64
```

2.2.2. Handling missing values in passenger_count

Missing Values

final_df['passenger_count'].value_counts(dropna=False)

```
passenger_count
1.0    221052
2.0     44096
3.0     11004
NaN     10182
4.0      6066
0.0      4654
5.0      3771
6.0      2567
8.0         2
7.0         2
9.0         1
Name: count, dtype: int64
```

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)

2.2.3. Handle missing values in RatecodeID

Missing Values

final_df['RatecodeID'].value_counts(dropna=False)

```
RatecodeID
1.0     276834
2.0      11463
NaN      10182
99.0      1727
5.0       1650
3.0        961
4.0        580
Name: count, dtype: int64
```

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)

2.2.4. Impute NaN in congestion_surcharge

Missing Values
final_df['congestion_surcharge'].value_counts(dropna=False)

```
congestion_surcharge
2.5    270773
0.0     22442
NaN     10182
Name: count, dtype: int64
```

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)

2.3. Handling Outliers and Standardising Values

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

Payment type

final_df['payment_type'].value_counts()

```
payment_type
1    238799
2     50822
0     10105
4      2187
3      1389
Name: count, dtype: int64
```

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[:,]

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DOLocationID	payment_type	...
143217	2	2023-06-13 09:59:00	2023-06-13 10:12:00	1.0	22528.82	1.0	N	116	239	0	...
194295	2	2023-02-17 07:17:00	2023-02-17 07:25:00	1.0	8645.77	1.0	N	238	230	0	...
197483	2	2023-02-19 22:06:00	2023-02-19 22:22:00	1.0	6284.45	1.0	N	186	236	0	...

3 rows × 21 columns

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)]

3. Exploratory Data Analysis

3.1. General EDA: Finding Patterns and Trends

3.1.1. Classify variables into categorical and numerical

Categorical Variables

VendorID, RatecodeID, PULocationID, DOLocationID, payment_type

Numerical Variables

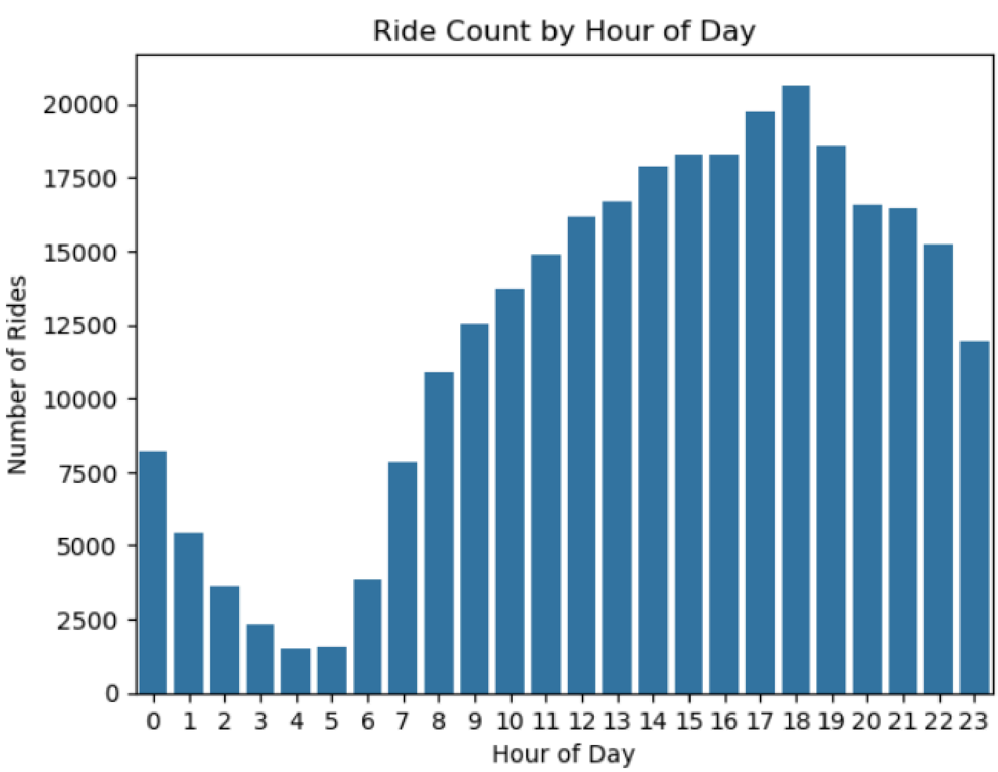
fare_amount, extra, mta_tax, tip_amount, tolls_amount, improvement_surcharge, total_amount, congestion_surcharge, airport_fee, tpep_pickup_datetime, tpep_dropoff_datetime, passenger_count, trip_distance, trip_duration, pickup_hour

3.1.2. 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()
```

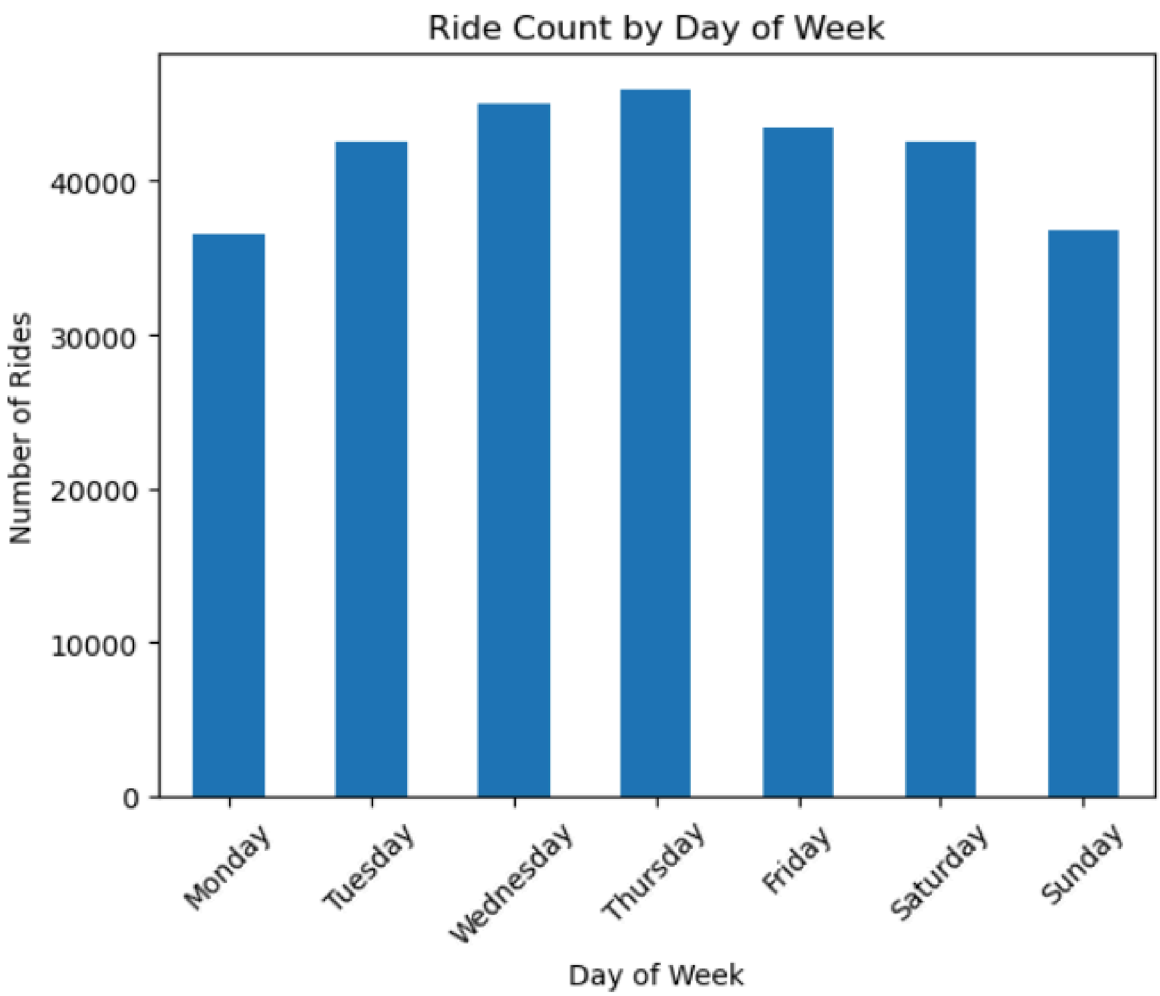
Result



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','W
ednesday','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()
```

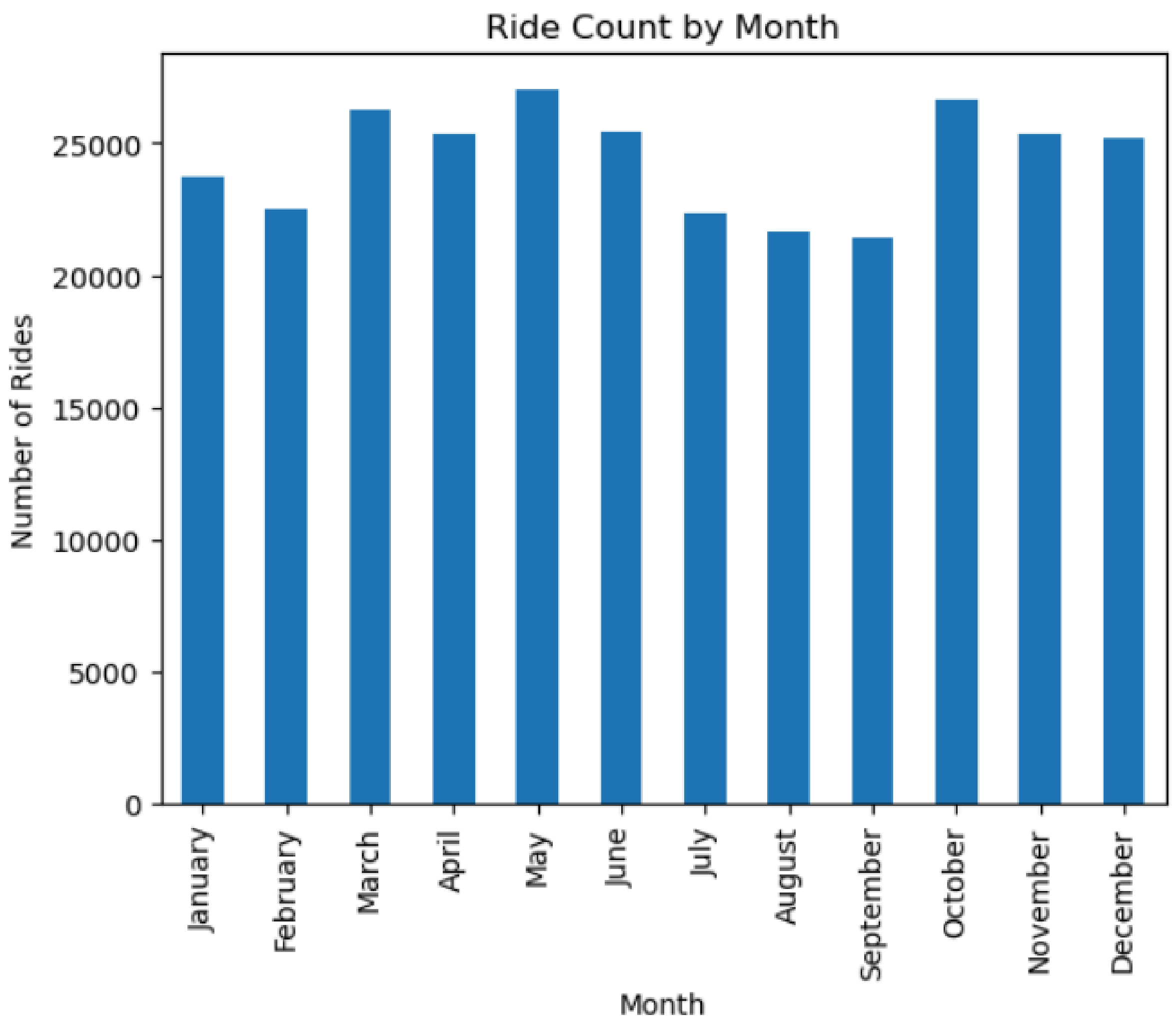
Result



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','Decem
ber']).plot(kind='bar')
plt.title('Ride Count by Month')
plt.xlabel('Month')
plt.ylabel('Number of Rides')
plt.show()
```

Result



3.1.3. 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

```
Zero values in 'fare_amount': 88
Zero values in 'tip_amount': 65640
Zero values in 'total_amount': 47
Zero values in 'trip_distance': 3521
```

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

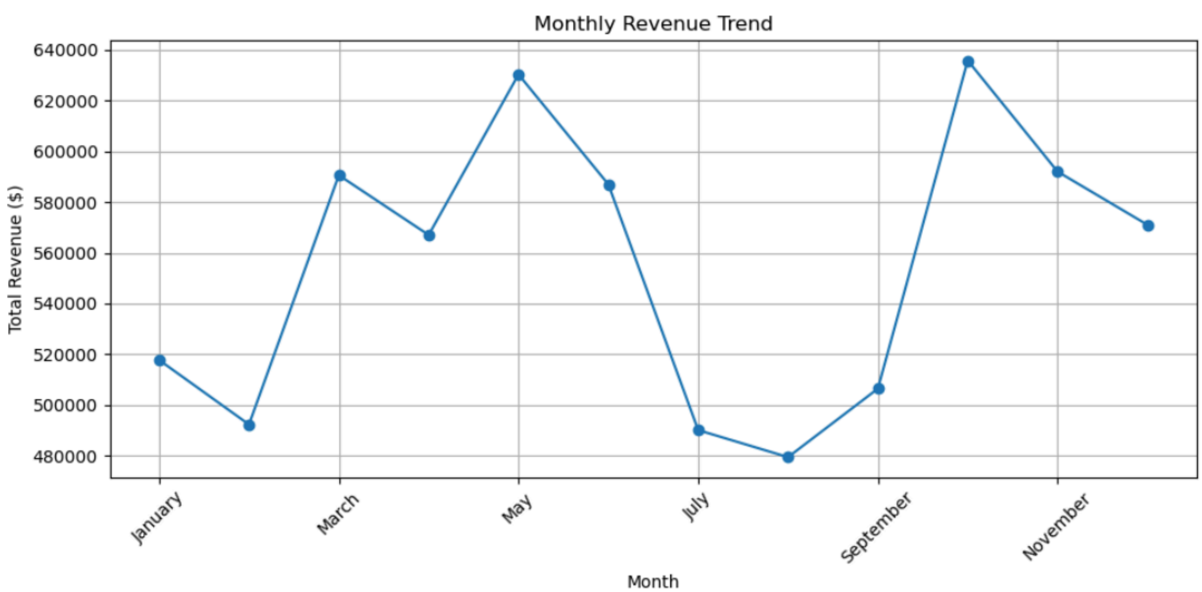
```
sliced1 = cleaned_df[(cleaned_df['fare_amount'] > 0) &
(cleaned_df['tip_amount'] > 0) & (cleaned_df['total_amount'] > 0) &
(cleaned_df['trip_distance'] > 0)]
```

3.1.4. 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

```
month
April          567056.31
August         479428.96
December       571065.45
February       492310.20
January        517727.50
July           490085.92
June           587013.13
March          590626.40
May            630363.64
November       592044.95
October        635870.89
September      506454.02
Name: total_amount, dtype: float64
```



3.1.5. 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)
```

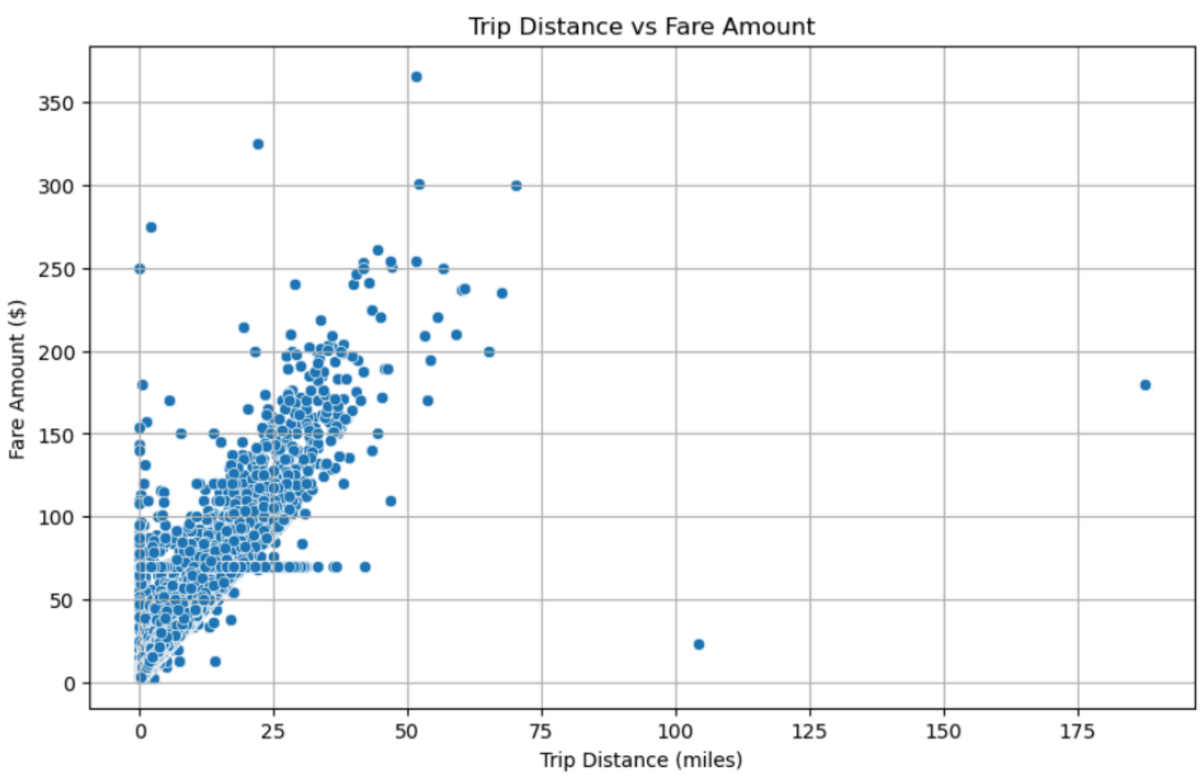
Result

```
quarter
2023Q1    24.033825
2023Q2    26.793099
2023Q3    22.161538
2023Q4    27.011539
Freq: Q-DEC, Name: total_amount, dtype: float64
```

3.1.6. 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()
```

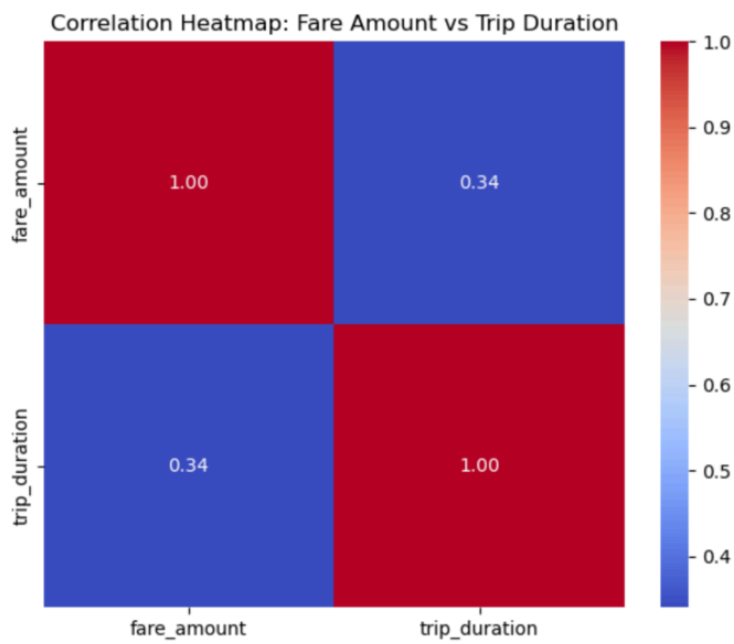
Result



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

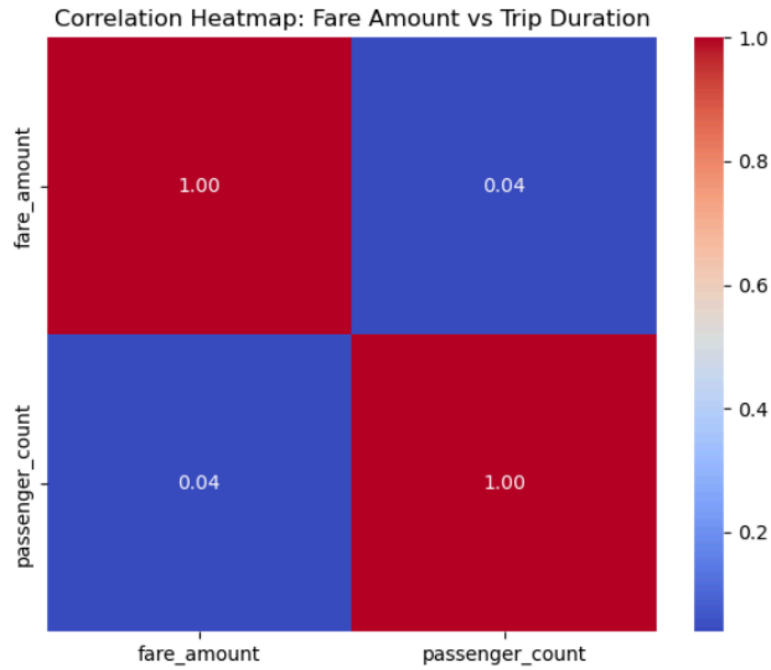
Show relationship between fare and trip duration

```
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()
```



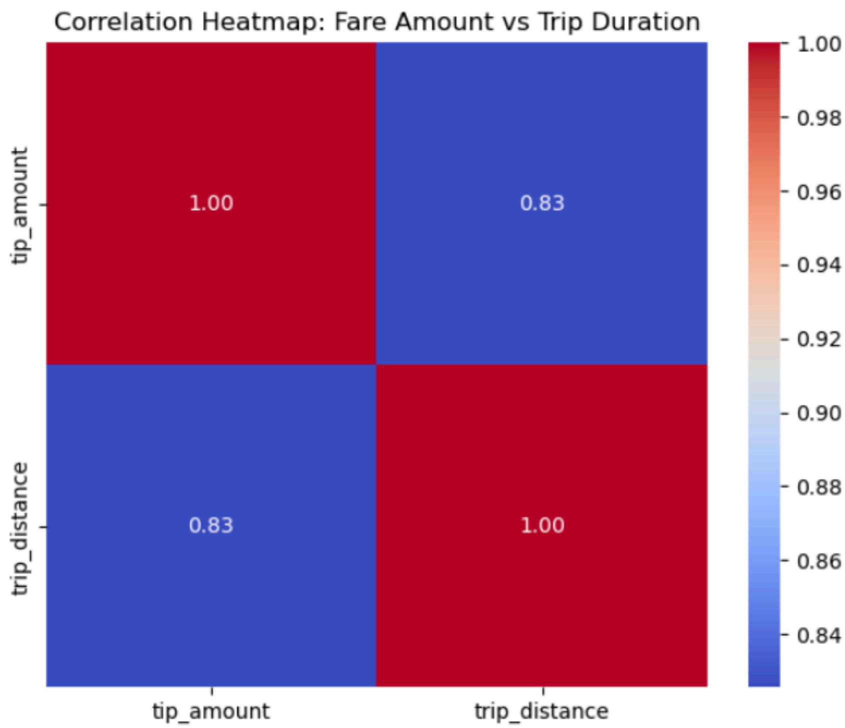
Show relationship between fare and number of passengers

```
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()
```



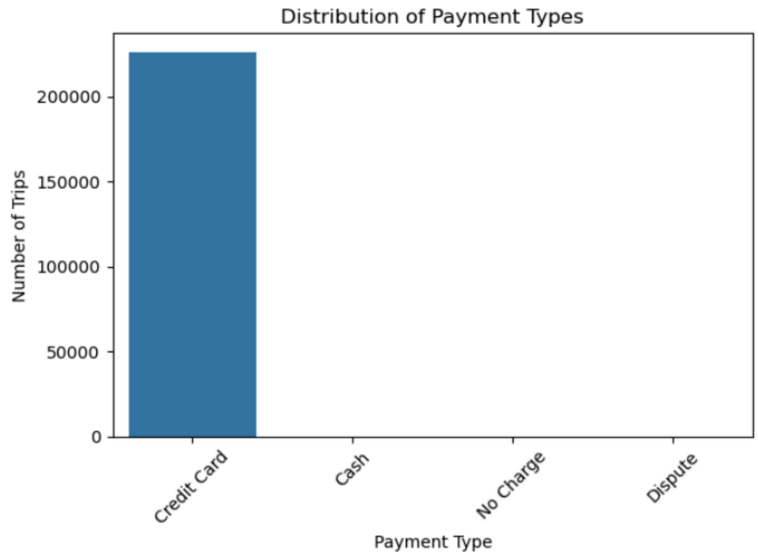
Show relationship between tip and trip distance

```
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()
```



3.1.8. 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()
```



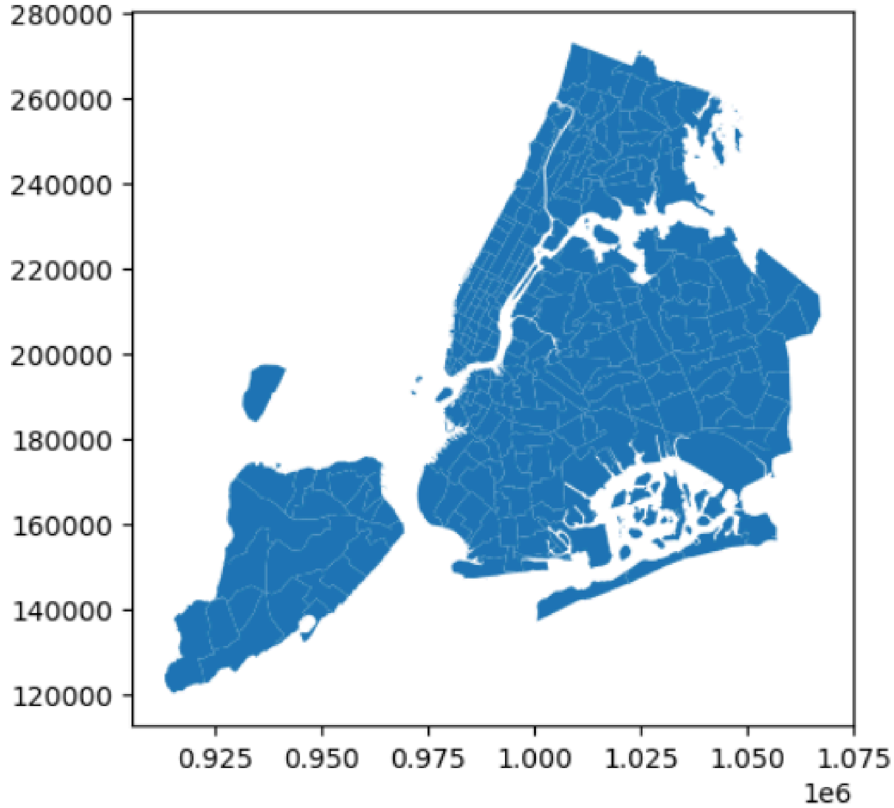
Note: The values of cash, No Charge, Dispute were less to be displayed on the graph

3.1.9. 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()
```

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry
0	1	0.116357	0.000782	Newark Airport	1	EWR	POLYGON ((933100.918 192536.086, 933091.011 19...
1	2	0.433470	0.004866	Jamaica Bay	2	Queens	MULTIPOLYGON (((1033269.244 172126.008, 103343...
2	3	0.084341	0.000314	Allerton/Pelham Gardens	3	Bronx	POLYGON ((1026308.77 256767.698, 1026495.593 2...
3	4	0.043567	0.000112	Alphabet City	4	Manhattan	POLYGON ((992073.467 203714.076, 992068.667 20...
4	5	0.092146	0.000498	Arden Heights	5	Staten Island	POLYGON ((935843.31 144283.336, 936046.565 144...

zones.plot()



3.1.10. 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()

store_and_fwd_flag	PULocationID	DOLocationID	payment_type	...	month	quarter	trip_duration	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry
N	161	237	1	...	January	2023Q1	0 days 00:05:05	161.0	0.035804	0.000072	Midtown Center	161.0	Manhattan	POLYGON ((991081.026 214453.698, 990952.644 21...
N	246	37	1	...	January	2023Q1	0 days 00:33:37	246.0	0.069467	0.000281	West Chelsea/Hudson Yards	246.0	Manhattan	POLYGON ((983031.177 217138.506, 983640.32 216...
N	79	164	1	...	January	2023Q1	0 days 00:10:31	79.0	0.042625	0.000108	East Village	79.0	Manhattan	POLYGON ((988746.067 202151.955, 988733.885 20...
N	79	256	1	...	January	2023Q1	0 days 00:15:53	79.0	0.042625	0.000108	East Village	79.0	Manhattan	POLYGON ((988746.067 202151.955, 988733.885 20...
N	132	95	1	...	January	2023Q1	0 days 00:17:08	132.0	0.245479	0.002038	JFK Airport	132.0	Queens	MULTIPOLYGON (((1032791.001 181085.006, 103283...

3.1.11. 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

	LocationID	trip_count
140	237.0	11223
91	161.0	10707
70	132.0	10199
139	236.0	10148
92	162.0	8551

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

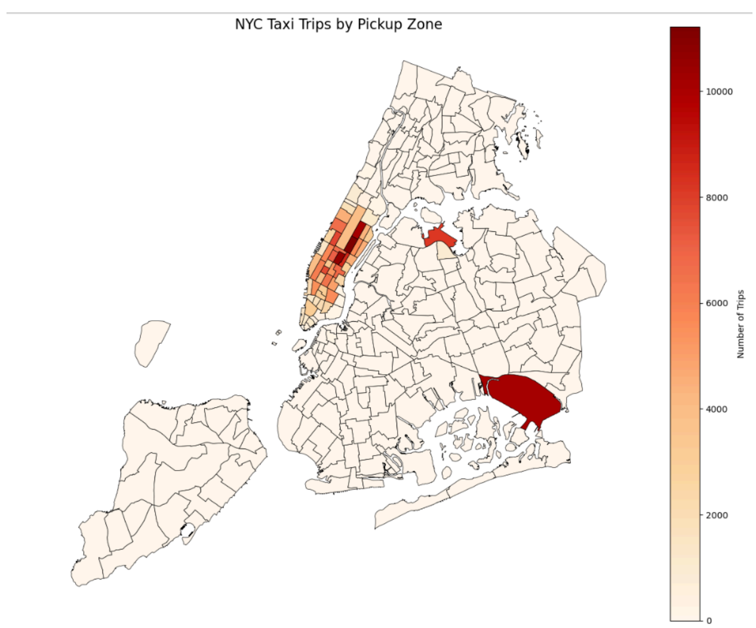
```
# Merge trip counts back to the zones GeoDataFrame
merged_zones_trips = pd.merge(zones, pickup_counts, how='left',
left_on='LocationID', right_on='LocationID')

merged_zones_trips.head()
```

Result

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry	trip_count
0	1	0.116357	0.000782	Newark Airport	1	EWR	POLYGON ((933100.918 192536.086, 933091.011 19...	4.0
1	2	0.433470	0.004866	Jamaica Bay	2	Queens	MULTIPOLYGON (((1033269.244 172126.008, 103343...	NaN
2	3	0.084341	0.000314	Allerton/Pelham Gardens	3	Bronx	POLYGON ((1026308.77 256767.698, 1026495.593 2...	NaN
3	4	0.043567	0.000112	Alphabet City	4	Manhattan	POLYGON ((992073.467 203714.076, 992068.667 20...	216.0
4	5	0.092146	0.000498	Arden Heights	5	Staten Island	POLYGON ((935843.31 144283.336, 936046.565 144...	NaN

3.1.13. 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()
```



3.1.14. 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

3.2. Detailed EDA: Insights and Strategies

3.2.1. 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

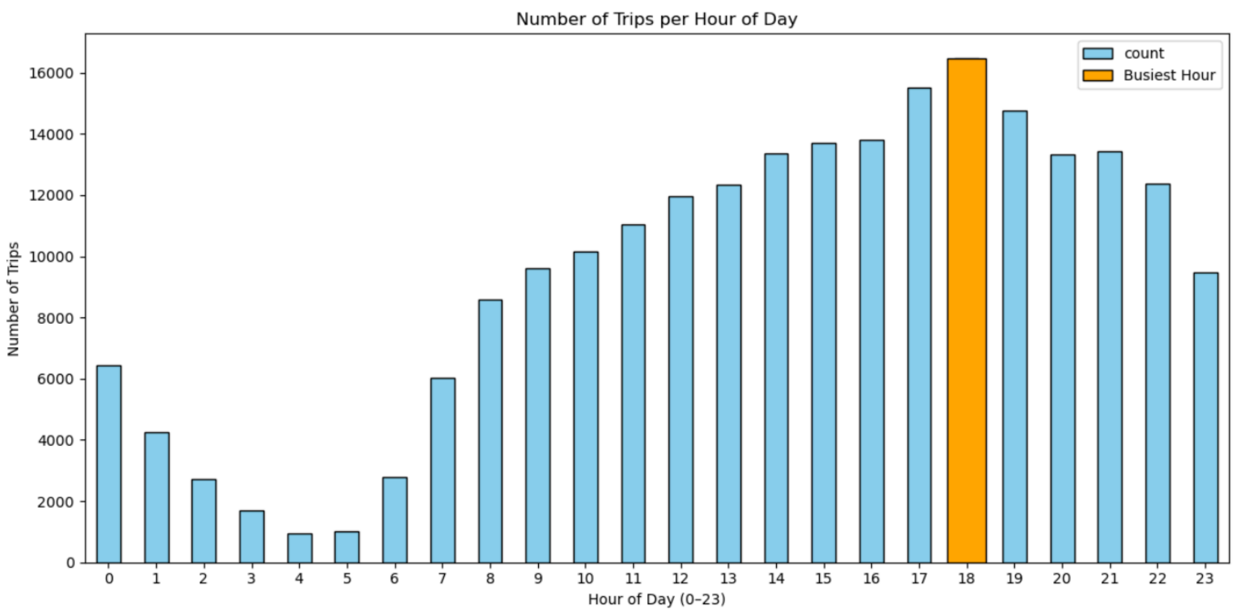
	PULocationID	Pickupzone	DOLocationID	Dropoffzone	pickup_hour	trip_duration_min	trip_distance	avg_speed_mph
9	229	Sutton Place/Turtle Bay North	145	Long Island City/Hunters Point	16	703.366667	2.3900	0.203877
8	229	Sutton Place/Turtle Bay North	41	Central Harlem	17	1428.083333	4.1600	0.174780
7	113	Greenwich Village North	181	Park Slope	19	35.250000	0.0900	0.153191
6	163	Midtown North	87	Financial District North	15	38.550000	0.0900	0.140078
5	234	Union Sq	256	Williamsburg (South Side)	18	1425.250000	3.2200	0.135555
4	41	Central Harlem	41	Central Harlem	16	361.245833	0.6775	0.112527
3	209	Seaport	25	Boerum Hill	22	1425.650000	2.5200	0.106057
2	164	Midtown South	100	Garment District	21	698.833333	0.7900	0.067827
1	209	Seaport	232	Two Bridges/Seward Park	13	1431.883333	1.0400	0.043579
0	113	Greenwich Village North	113	Greenwich Village North	13	1426.733333	0.3900	0.016401

3.2.2. 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



3.2.3. 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

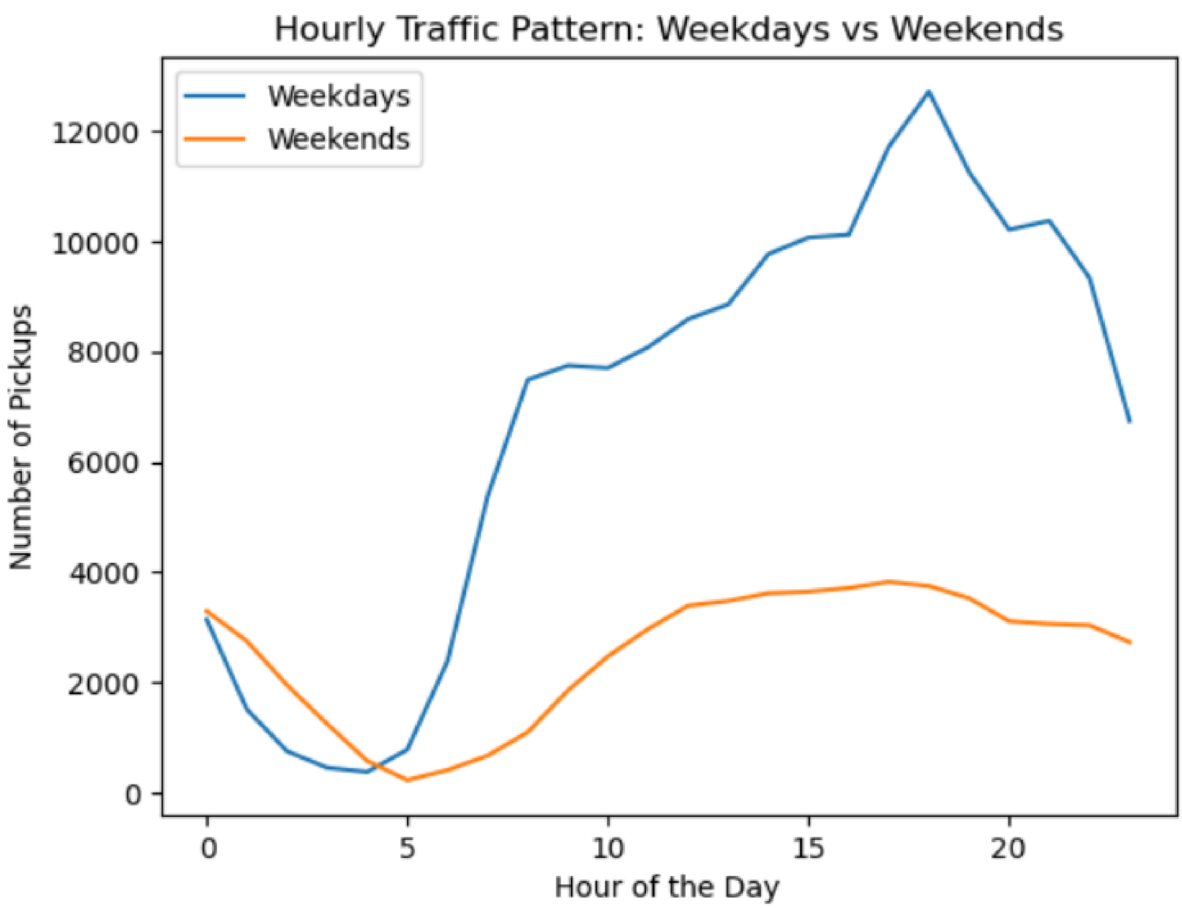
```
Estimated number of trips in the 5 busiest hours:
pickup_hour
18    2056750
17    1941125
19    1846750
16    1727875
15    1713375
Name: count, dtype: int32
```

3.2.4. 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']



3.2.5. 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')

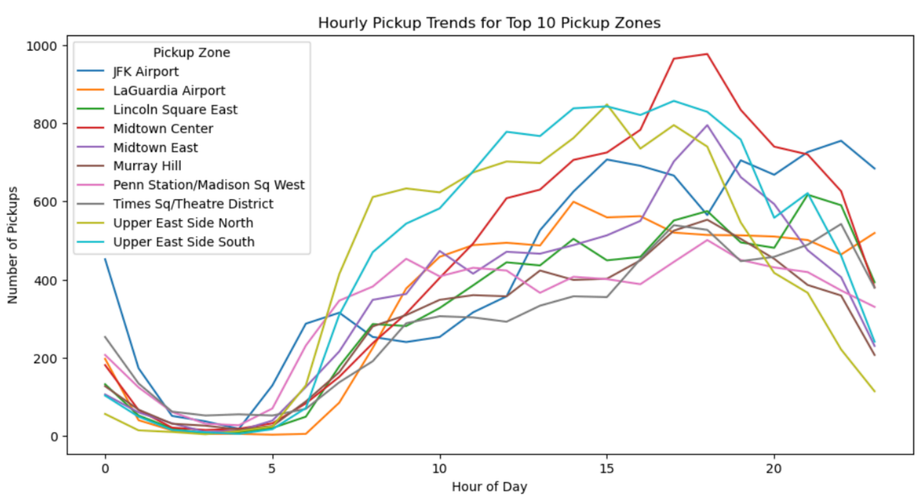
merged_pickup_locs = pd.merge(pickup_locs, zones, how='left', left_on='PULocationID', right_on='LocationID')

Plot_pickups = merged_pickup_locs.loc[:, ["PULocationID", "zone", "pickup_hour", "counts"]].sort_values('counts', ascending=False)[0:10]

Result

	PULocationID	zone	pickup_hour	counts
1246	161	Midtown Center	18	977
1245	161	Midtown Center	17	965
1813	237	Upper East Side South	17	857
1787	236	Upper East Side North	15	848
1811	237	Upper East Side South	15	843
1810	237	Upper East Side South	14	838
1247	161	Midtown Center	19	834
1814	237	Upper East Side South	18	829
1812	237	Upper East Side South	16	821
1789	236	Upper East Side North	17	795

```
# Top 10 pickup zones overall
top_pickup_zones = sliced1['PULocationID'].value_counts().head(10).index
pickup_trends = pickup_locs[pickup_locs['PULocationID'].isin(top_pickup_zones)]
pickup_trends = pd.merge(pickup_trends, zones, how='left',
left_on='PULocationID', right_on='LocationID')

plt.figure(figsize=(12,6))
sns.lineplot(data=pickup_trends, x='pickup_hour', y='counts', hue='zone')
plt.title('Hourly Pickup Trends for Top 10 Pickup Zones')
plt.xlabel('Hour of Day')
plt.ylabel('Number of Pickups')
plt.legend(title='Pickup Zone')
plt.show()
```



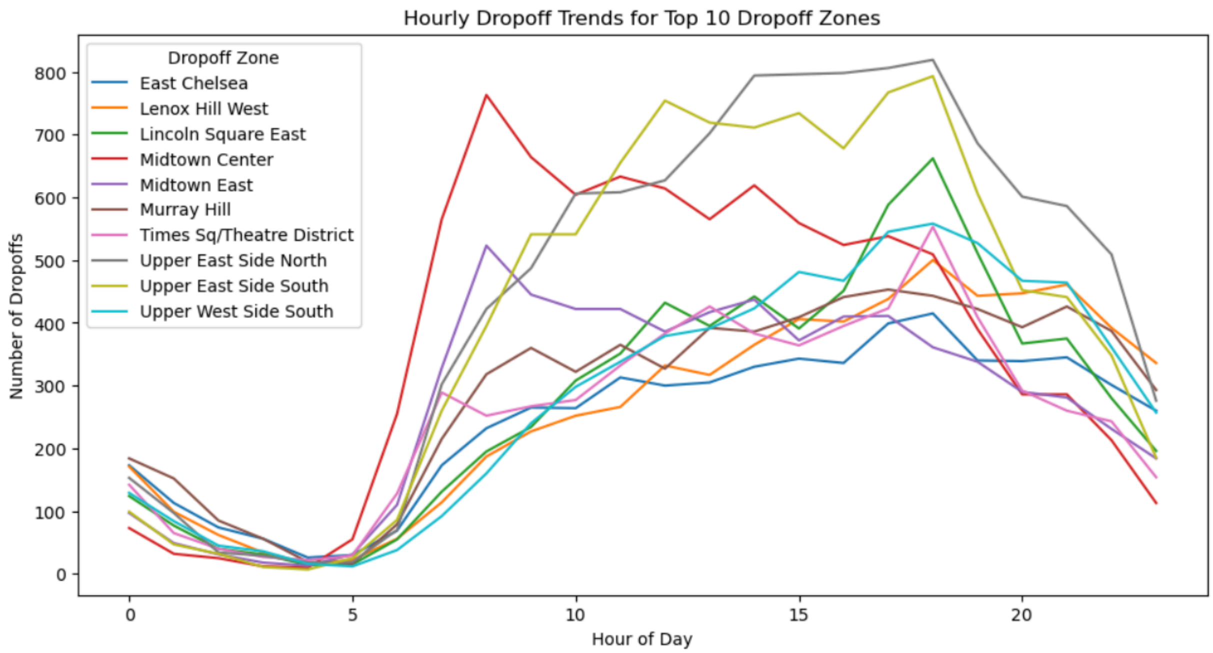
Top 10 zones with high hourly drops

```
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')
Plot_dropoffs = merged_dropoff_locs.loc[:, ["DOLocationID", "zone",
"pickup_hour", "counts"]].sort_values('counts', ascending=False)[0:10]
```

Result

	DOLocationID	zone	pickup_hour	counts
3416	236	Upper East Side North	18	819
3415	236	Upper East Side North	17	806
3414	236	Upper East Side North	16	798
3413	236	Upper East Side North	15	796
3412	236	Upper East Side North	14	794
3440	237	Upper East Side South	18	793
3439	237	Upper East Side South	17	767
2303	161	Midtown Center	8	763
3434	237	Upper East Side South	12	754
3437	237	Upper East Side South	15	734

```
# Top 10 dropoff zones overall
top_dropoff_zones = sliced1['DOLocationID'].value_counts().head(10).index
dropoff_trends =
dropoff_locs[dropoff_locs['DOLocationID'].isin(top_dropoff_zones)]
dropoff_trends = pd.merge(dropoff_trends, zones, how='left',
left_on='DOLocationID', right_on='LocationID')

plt.figure(figsize=(12,6))
sns.lineplot(data=dropoff_trends, x='pickup_hour', y='counts', hue='zone')
plt.title('Hourly Dropoff Trends for Top 10 Dropoff Zones')
plt.xlabel('Hour of Day')
plt.ylabel('Number of Dropoffs')
plt.legend(title='Dropoff Zone')
plt.show()
```



3.2.6. 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

```
Top 10 Pickup/Dropoff Ratios by Zone:
```

	zone	pickup_count	dropoff_count	pickup_dropoff_ratio
65	East Elmhurst	1043.0	72	14.486111
120	JFK Airport	10199.0	2123	4.804051
126	LaGuardia Airport	8144.0	2814	2.894101
198	South Jamaica	23.0	14	1.642857
172	Penn Station/Madison Sq West	7703.0	4810	1.601455
40	Central Park	3899.0	2802	1.391506
232	West Village	5487.0	4018	1.365605
103	Greenwich Village South	3087.0	2312	1.335208
149	Midtown East	8551.0	6607	1.294233
93	Garment District	3395.0	2781	1.220784

```
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

```
Bottom 10 Pickup/Dropoff Ratios by Zone:
```

	zone	pickup_count	dropoff_count	pickup_dropoff_ratio
114	Hunts Point	0.0	4	0.0
64	East Concourse/Concourse Village	0.0	25	0.0
171	Pelham Parkway	0.0	16	0.0
62	Dyker Heights	0.0	12	0.0
94	Glen Oaks	0.0	4	0.0
141	Marble Hill	0.0	1	0.0
59	Douglaston	0.0	15	0.0
58	Cypress Hills	0.0	23	0.0
177	Queensboro Hill	0.0	12	0.0
170	Pelham Bay Park	0.0	1	0.0

3.2.7. 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

```
Top 10 Zones with Highest Nighttime Traffic (11PM-5AM):
```

	zone	night_pickup_count	night_dropoff_count	total_night_traffic
65	East Village	2074.0	1090.0	3164.0
208	West Village	1715.0	644.0	2359.0
38	Clinton East	1261.0	850.0	2111.0
122	Lower East Side	1292.0	579.0	1871.0
107	JFK Airport	1544.0	181.0	1725.0
88	Gramercy	752.0	765.0	1517.0
56	East Chelsea	740.0	732.0	1472.0
190	Times Sq/Theatre District	987.0	479.0	1466.0
92	Greenwich Village South	1109.0	341.0	1450.0
141	Murray Hill	499.0	805.0	1304.0

3.2.8. 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

```
Nighttime Revenue Share: 11.96%
Daytime Revenue Share: 88.04%
```

3.2.9. 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'].mea
n().reset_index()

print(average_fare_per_passenger)
```

Result

	passenger_count	fare_per_mile_per_passenger
0	1.0	9.005900
1	2.0	5.519112
2	3.0	3.640453
3	4.0	3.411557
4	5.0	1.511769
5	6.0	1.272553

3.2.10. 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

```
Average Fare per Mile by Hour of the Day:
    pickup_hour  fare_per_mile
0             0       8.203569
1             1       9.469329
2             2       7.118360
3             3       7.681090
4             4      22.432357
5             5      11.851857
6             6      13.771515
7             7       7.191759
8             8       8.463444
9             9       8.583771
10           10       8.894227
11           11      10.204584
12           12      10.361687
13           13      10.787613
14           14      10.272839
15           15       9.712556
16           16      11.892260
17           17      10.487748
18           18       9.248119
19           19       9.643830
20           20       7.347292
21           21       8.477767
22           22       7.935976
23           23       7.383828
```

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

Result

Average Fare per Mile by Day of the Week:

	day_of_week	fare_per_mile
0	Friday	8.732178
1	Monday	8.417190
2	Saturday	9.347328
3	Sunday	10.186626
4	Thursday	10.904930
5	Tuesday	9.496100
6	Wednesday	8.758707

3.2.11. 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

```
Average Fare per Mile by Vendor and Hour of the Day:
    VendorID  pickup_hour  fare_per_mile
0          1            0       6.442728
1          1            1       6.482393
2          1            2       6.451775
3          1            3       6.269806
4          1            4       6.293682
5          1            5       6.852921
6          1            6       6.324118
7          1            7       6.991054
8          1            8       8.009076
9          1            9       8.011939
10         1           10       8.050047
11         1           11       8.420010
12         1           12       8.598629
13         1           13       8.262844
14         1           14       8.678087
15         1           15       8.525029
16         1           16       8.401183
17         1           17       8.386591
18         1           18       8.425798
19         1           19       7.811397
20         1           20       7.171665
21         1           21       7.084760
22         1           22       6.839148
23         1           23       6.517253
24         2            0       8.717073
25         2            1      10.300726
26         2            2       7.322869
27         2            3       8.048253
28         2            4      26.562332
29         2            5      13.620506
30         2            6      16.668342
31         2            7       7.270912
32         2            8       8.639009
33         2            9       8.794305
34         2           10       9.215132
35         2           11      10.868477
36         2           12      10.984576
37         2           13      11.726115
38         2           14      10.837620
39         2           15      10.137933
40         2           16      13.119325
41         2           17      11.234036
42         2           18       9.526015
43         2           19      10.229221
44         2           20       7.402289
45         2           21       8.906729
46         2           22       8.257297
47         2           23       7.634554
```

3.2.12. 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

```
Average Fare per Mile by Vendor and Distance Range:
   VendorID distance_range  fare_per_mile
0         1   Up to 2 miles       9.409998
1         1    2 to 5 miles       6.359586
2         1   Above 5 miles       4.455716
3         2   Up to 2 miles      13.485444
4         2    2 to 5 miles       6.539651
5         2   Above 5 miles       4.503751
```

3.2.13. 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

```
Average Tip Percentage by Distance Range:
  distance_range  tip_percentage
0  Up to 2 miles       28.499416
1   2 to 5 miles       22.999048
2  Above 5 miles       21.962930
```

3.2.14. 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

```
Average Tip Percentage by Passenger Count:
   passenger_count  tip_percentage
0              1.0       25.939603
1              2.0       25.544078
2              3.0       25.603503
3              4.0       25.598026
4              5.0       26.038132
5              6.0       25.930186
```

3.2.15. 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

```
Average Tip Percentage by Pickup Hour:
    pickup_hour  tip_percentage
0             0       25.429546
1             1       25.919045
2             2       25.860245
3             3       25.937995
4             4       25.515856
5             5       25.434024
6             6       25.003614
7             7       24.794154
8             8       24.845981
9             9       25.036653
10           10       25.374359
11           11       25.221968
12           12       25.201899
13           13       25.087920
14           14       24.919113
15           15       24.733242
16           16       27.096108
17           17       27.164846
18           18       27.389039
19           19       27.393391
20           20       26.119766
21           21       26.126442
22           22       25.788168
23           23       25.408875
```

3.2.16. 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

	zone	total_count	extra_applied_count	percent_with_extra
74	LaGuardia Airport	8144	8062	98.993124
90	Midtown Center	10707	7122	66.517232
140	Upper East Side South	11223	6440	57.382162
91	Midtown East	8551	5399	63.138814
139	Upper East Side North	10148	5375	52.966102

4. Conclusions

4.1. Final Insights and Recommendations

4.1.1. 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).

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

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.

4.1.3. 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.