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

```
Loaded all the data files using below code format-
df_Jan2023 = pd.read_parquet('/content/yellow_tripdata_2023-01.parquet')
```

```
df_Jan2023.info()
```

Sample the data and combine the files

Sampled the 5% data using these commands-

```
df_Jan2023['hour'] = df_Jan2023["tpep_pickup_datetime"].dt.hour
sample_df_Jan2023 = df_Jan2023.groupby("hour").sample(frac=0.05,
random_state=42)
sample_df_Jan2023.to_parquet("/content/drive/My Drive/Colab
Notebooks/Assignments/nyc_taxi_sample1.parquet", index=False)
```

```
final_sample = pd.concat([sample_df_Jan2023, sample_df_Feb2023,
sample_df_Mar2023, sample_df_Apr2023, sample_df_May2023, sample_df_Jun2023,
sample_df_Jul2023, sample_df_Aug2023, sample_df_Sep2023, sample_df_Oct2023,
sample_df_Nov2023, sample_df_Dec2023])
final_sample.to_parquet("/content/drive/My Drive/Colab
Notebooks/Assignments/final_sample.parquet", index=False)
```

```
# Take a small percentage of entries from each hour of every date.
# Iterating through the monthly data:
#   read a month file -> day -> hour: append sampled data -> move to next
hour -> move to next day after 24 hours -> move to next month file
# Create a single dataframe for the year combining all the monthly data
```

```
# Select the folder having data files
import os
```

```
# Select the folder having data files
os.chdir('drive/My Drive/Colab Notebooks/Assignments')
```

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

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

# iterate through the list of files and sample one by one:
for file_name in file_list:
    try:
        # file path for the current file
        file_path = os.path.join(os.getcwd(), file_name)

        # Reading the current file

        # 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

            # Iterate through each hour of the selected date

                # Sample 5% of the hourly data randomly

                # add data of this hour to the dataframe

        # Concatenate the sampled data of all the dates to a single dataframe
        # we initialised this empty DF earlier

    except Exception as e:
        print(f"Error reading file {file_name}: {e}")
```

2. Data Cleaning

2.1. Fixing Columns

2.1.1. Fix the index

```
df = df.reset_index(drop=True)
```

and dropped vendor_id column

```
df_clean = df.drop(columns = ['VendorID'])
```

2.1.2. Combine the two airport_fee columns

df_clean['airport_fee'] = df_clean['airport_fee'].combine_first(df_clean['Airport_fee'])

```
df_clean.drop(columns=['Airport_fee'], inplace=True)
```

2.1.3. Fix columns with negative (monetary) values

Finding which column have negative values:

```
numerical_columns =
df_clean.select_dtypes(include=np.number).columns.tolist()
```

Fixing these negative values

```
for col in ['fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
'improvement_surcharge', 'total_amount', 'congestion_surcharge',
'airport_fee']:
  df_clean.loc[df_clean[col] < 0, col] = 0
df_cleaned = df_clean[~(df_clean[numerical_columns] < 0).any(axis=1)]
```

2.2. Handling Missing Values

2.2.1. Find the proportion of missing values in each column

```
Missing_values = (df_cleaned.isna().sum() / len(df_cleaned)) * 100
```

2.2.2. Handling missing values in passenger_count

```
df_cleaned['passenger_count'].fillna(df_cleaned['passenger_count'].mode()[0]
, inplace=True)
```

2.2.3. Handle missing values in RatecodeID

df_cleaned['RatecodeID'].fillna(df_cleaned['RatecodeID'].median(), inplace=True)

2.2.4. Impute NaN in congestion_surcharge

df_cleaned['congestion_surcharge'].fillna(df_cleaned['congestion_surcharge'].mode()[0], inplace=True)

2.3. Handling Outliers and Standardising Values

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

Selected_records=df_cleaned[(df_cleaned['trip_distance'] < 1) & (df_cleaned['fare_amount'] > 300)]

```
initial_rows = len(df_cleaned)
df_cleaned = df_cleaned[~((df_cleaned['trip_distance'] < 1) &
(df_cleaned['fare_amount'] > 300))]
rows_removed = initial_rows - len(df_cleaned)
print(f"Number of records removed with trip_distance < 1 and fare_amount >
300: {rows_removed}")
print(f"Number of records after removing extreme outliers:
{len(df_cleaned)}")
```

3. Exploratory Data Analysis

3.1. General EDA: Finding Patterns and Trends

3.1.1. Classify variables into categorical and numerical

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

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

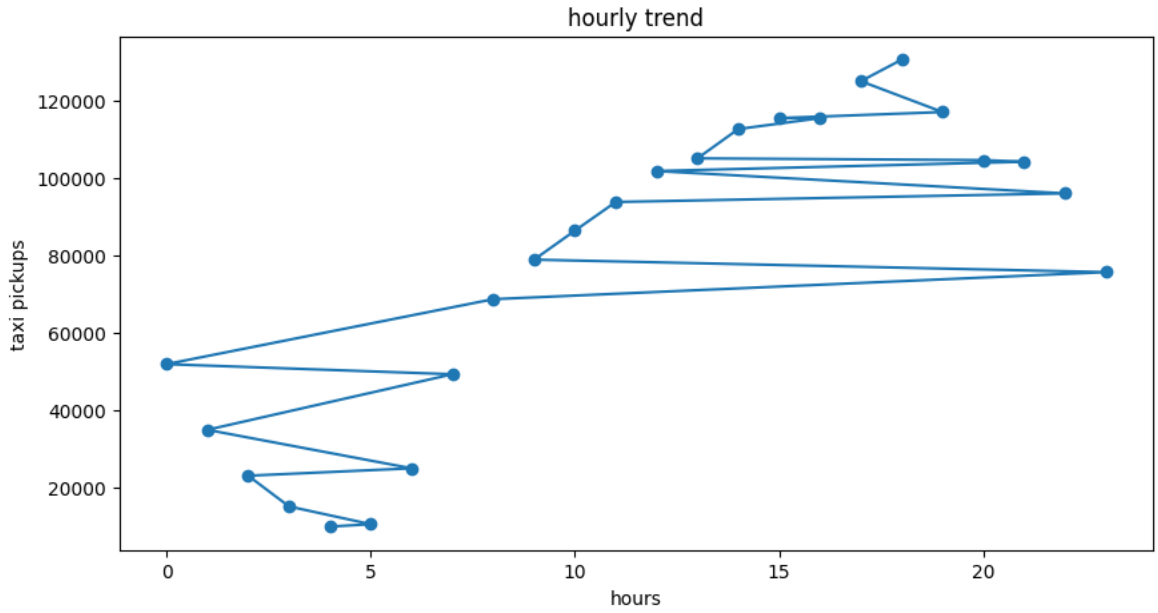
- `fare_amount` : Numerical
- `extra` : Numerical
- `mta_tax` : Numerical
- `tip_amount` : Numerical
- `tolls_amount` : Numerical

- `improvement_surcharge` : Numerical
- `total_amount` : Numerical
- `congestion_surcharge` : Numerical
- `airport_fee`: Numerical

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

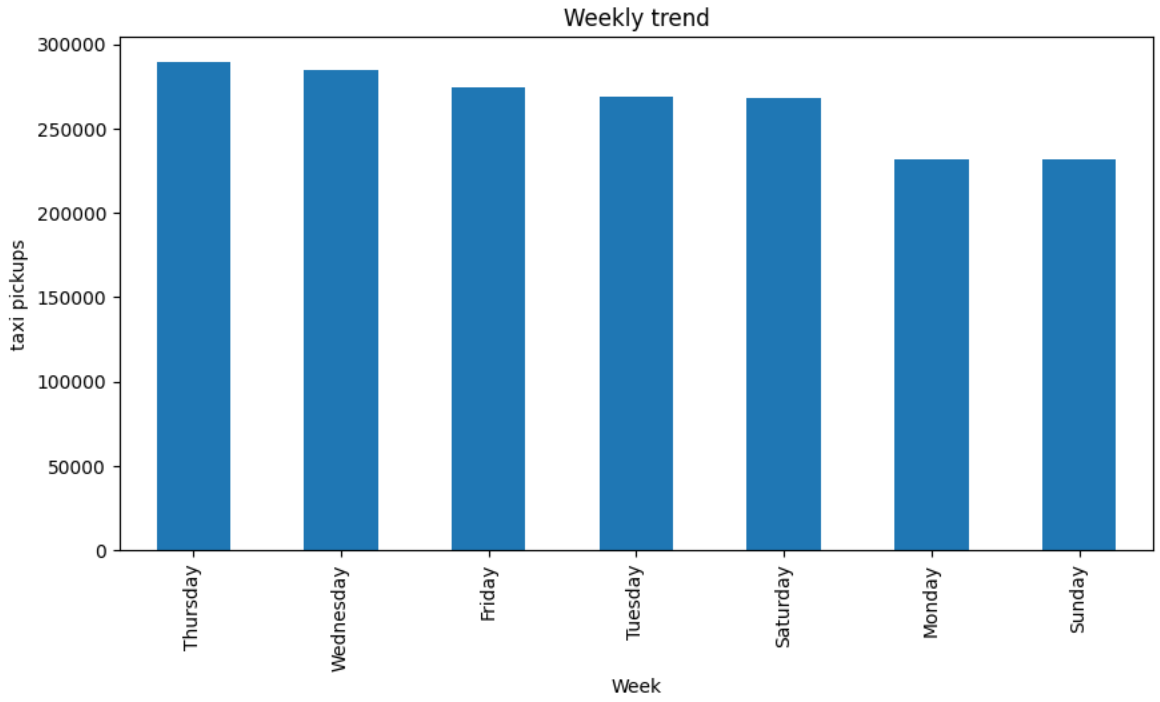
```
hourly_trend = df_cleaned['hour'].value_counts()

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

```
hourly_trend.plot(kind='line', marker='o')
plt.xlabel('hours')
plt.ylabel('taxi pickups')
plt.title('hourly trend')
plt.show()
```



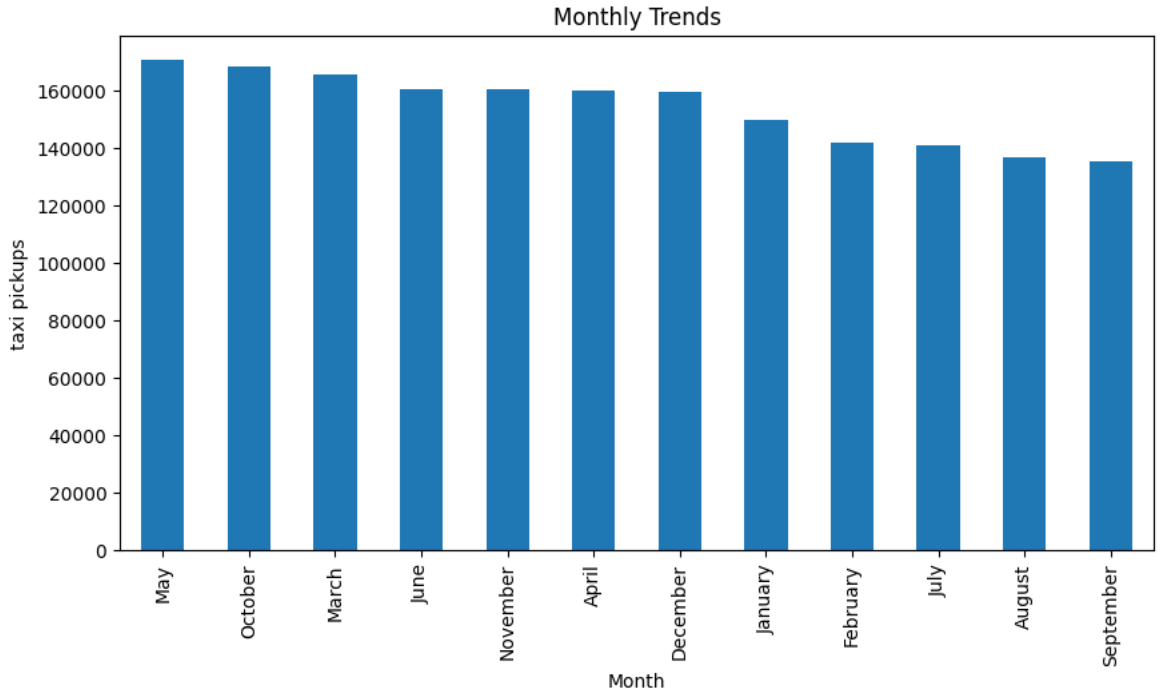
3.1.3. Filter out the zero/negative values in fares, distance and tips

```
df_cleaned['daily_trend'] = df_cleaned['tpep_pickup_datetime'].dt.day_name()
      daily_trend = df_cleaned['daily_trend'].value_counts()

plt.figure(figsize=(10,5))
daily_trend.plot(kind='bar')
plt.xlabel('Week')
plt.ylabel('taxi pickups')
plt.title('Weekly trend')
plt.show()
```



3.1.4. Analyse the monthly revenue trends df_cleaned['Monthly_trend'] = df_cleaned['tpep_pickup_datetime'].dt.month_name()

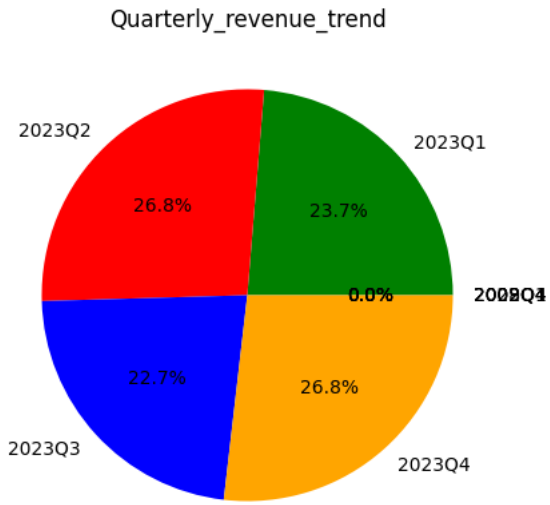
```
Monthly_trend = df_cleaned['Monthly_trend'].value_counts()

plt.figure(figsize=(10,5))
Monthly_trend.plot(kind='bar')
plt.xlabel('Month')
plt.ylabel('taxi pickups')
plt.title('Monthly Trends')
plt.show()
```



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

```
# Calculate proportion of each quarter
```

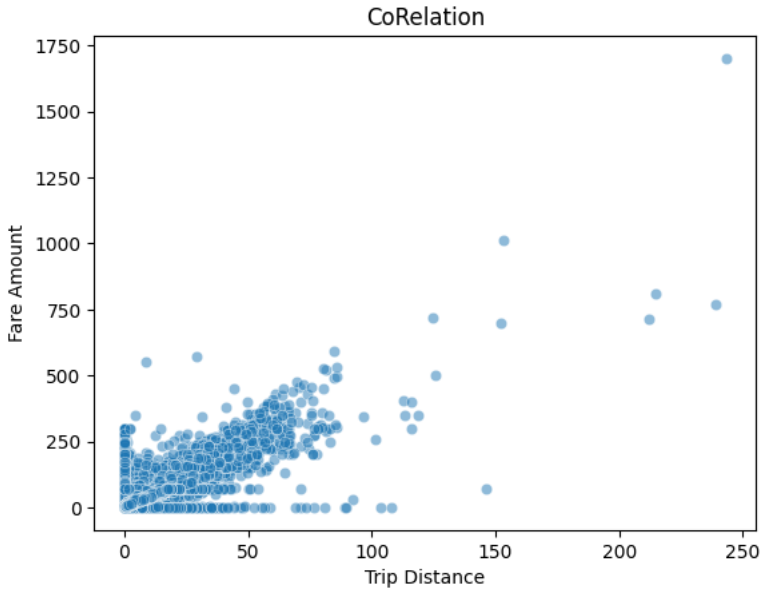
```
df_cleaned['quarterly_trend'] =
df_cleaned['tpep_pickup_datetime'].dt.to_period('Q')
quarterly_revenue =
df_cleaned.groupby('quarterly_trend')['total_amount'].sum().reset_index()

plt.figure(figsize=(5,5))
plt.pie(quarterly_revenue['total_amount'],
labels=quarterly_revenue['quarterly_trend'], autopct='%1.1f%%',
colors=['green', 'red', 'blue', 'orange'])
plt.title('Quarterly_revenue_trend')
plt.show()
```



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

```
sns.scatterplot(x=df_cleaned['trip_distance'], y=df_cleaned['fare_amount'],
alpha=0.5)
```

```
plt.xlabel('Trip Distance')
plt.ylabel('Fare Amount')
plt.title('CoRelation')
plt.show()
```

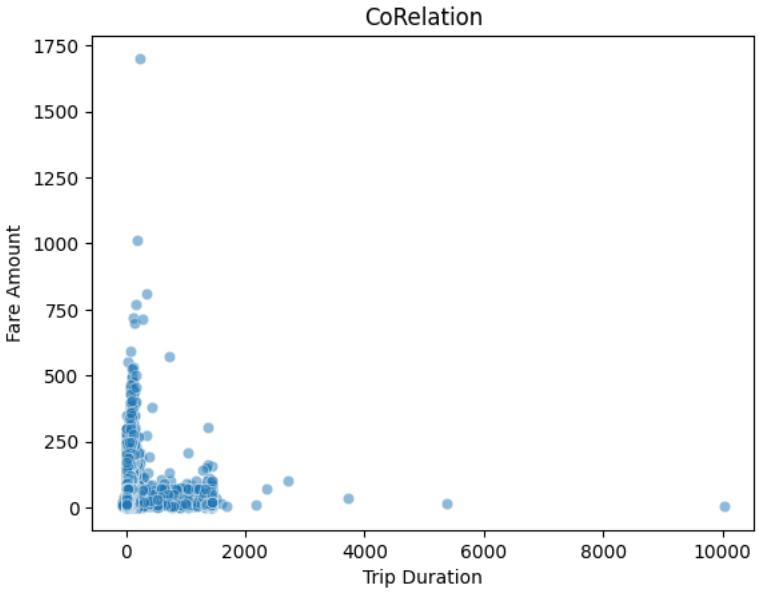


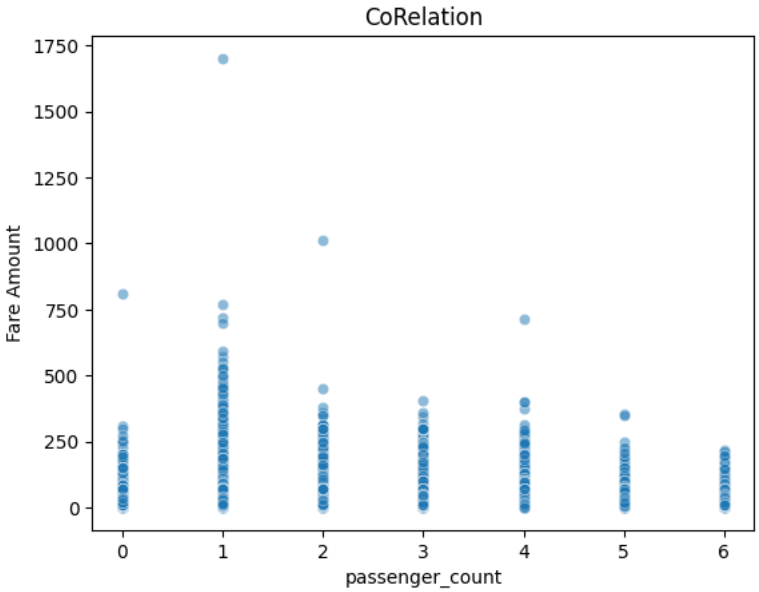
3.1.7. Analyse the relationship between fare/tips and trips/passengers

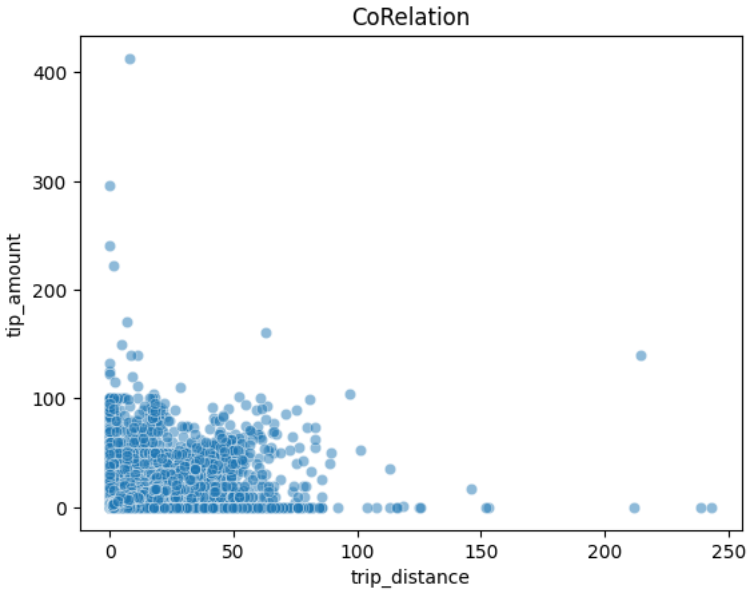
df_cleaned['trip_duration'] = (df_cleaned['tpep_dropoff_datetime'] - df_cleaned['tpep_pickup_datetime']).dt.total_seconds() / 60

```
plt.figure
sns.scatterplot(x=df_cleaned['trip_duration'], y=df_cleaned['fare_amount'], alpha=0.5)
```

```
plt.xlabel('Trip Duration')
plt.ylabel('Fare Amount')
plt.title('CoRelation')
plt.show()
```


CoRelation
Fare Amount
Trip Duration
0
250
500
750
1000
1250
1500
1750
0
2000
4000
6000
8000
10000


CoRelation
Fare Amount
passenger_count
0
250
500
750
1000
1250
1500
1750
0
1
2
3
4
5
6



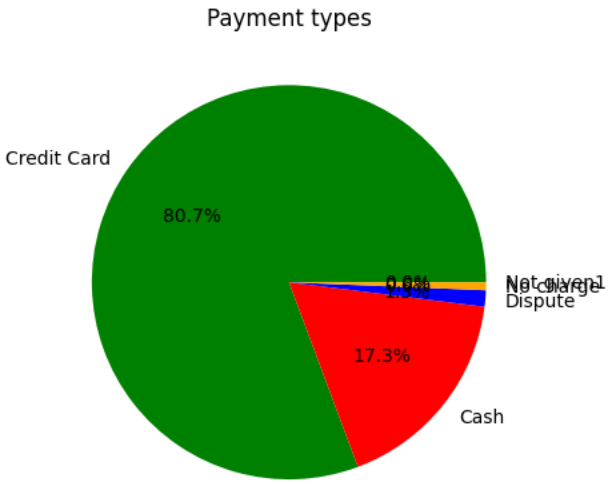
3.1.8. Analyse the distribution of different payment types

payment_type_mapping = {1: 'Credit Card', 2: 'Cash', 3: 'No charge', 4: 'Dispute', 5: 'Not given1', 0: 'Not given2'}

```
df_cleaned['payment_type'] =
df_cleaned['payment_type'].map(payment_type_mapping)
```

```
payment_types = df_cleaned['payment_type'].value_counts()
payment_types
```

```
plt.figure
plt.pie(payment_types, labels=payment_types.index, autopct='%1.1f%%',
colors=['green', 'red', 'blue', 'orange'])
plt.title('Payment types')
plt.show()
```



3.1.9. Load the taxi zones shapefile and display it

```
# import geopandas as gpd
import geopandas as gpd



# Read the shapefile using geopandas
zones = gpd.read_file('/content/taxi_zones/taxi_zones.shp')  # read the .shp
file using gpd
zones.head()
# print(zones.info())
print(zones.info())
# zones.plot()
zones.plot()


zones.columns
df_cleaned.head()
```

Output:

```
   OBJECTID  Shape_Leng  Shape_Area                     zone  LocationID  \
0         1    0.116357    0.000782           Newark Airport           1
1         2    0.433470    0.004866              Jamaica Bay           2
2         3    0.084341    0.000314  Allerton/Pelham Gardens           3
3         4    0.043567    0.000112            Alphabet City           4
4         5    0.092146    0.000498            Arden Heights           5

         borough                                           geometry
0            EWR  POLYGON ((933100.918 192536.086, 933091.011 19...
1         Queens  MULTIPOLYGON (((1033269.244 172126.008, 103343...
2          Bronx  POLYGON ((1026308.77 256767.698, 1026495.593 2...
```

```
3      Manhattan  POLYGON ((992073.467 203714.076, 992068.667 20...
4  Staten Island  POLYGON ((935843.31 144283.336, 936046.565 144...
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 263 entries, 0 to 262
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   OBJECTID    263 non-null    int32
 1   Shape_Leng  263 non-null    float64
 2   Shape_Area  263 non-null    float64
 3   zone        263 non-null    object
 4   LocationID  263 non-null    int32
 5   borough     263 non-null    object
 6   geometry    263 non-null    geometry
dtypes: float64(2), geometry(1), int32(2), object(2)
memory usage: 12.5+ KB
None


<Axes: >
<Figure size 640x480 with 1 Axes>
Index(['OBJECTID', 'Shape_Leng', 'Shape_Area', 'zone', 'LocationID',
'borough',
       'geometry'],
      dtype='object')
  tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  trip_distance
\
0  2023-01-18 00:32:54   2023-01-18 00:45:04             1.0           3.21
1  2023-01-15 00:17:26   2023-01-15 00:27:34             1.0           1.50
2  2023-01-28 00:46:31   2023-01-28 00:53:58             1.0           1.30
3  2023-01-28 00:57:26   2023-01-28 01:02:03             1.0           1.10
4  2023-01-29 00:32:44   2023-01-29 00:42:06             2.0           0.99

   RatecodeID store_and_fwd_flag  PULocationID  DOLocationID payment_type  \
0         1.0                  N           158           161  Credit Card
1         1.0                  N           144           107  Credit Card
2         1.0                  N           249            90  Credit Card
3         1.0                  N           237            48  Credit Card
4         1.0                  N           148           113         Cash

   fare_amount  ...  tolls_amount  improvement_surcharge  total_amount  \
0         15.6  ...           0.0                    1.0         24.72
1         10.7  ...           0.0                    1.0         19.65
2          9.3  ...           0.0                    1.0         15.30
3          7.2  ...           0.0                    1.0         15.85
4         10.0  ...           0.0                    1.0         15.00

   congestion_surcharge  airport_fee  hour  daily_trend  Monthly_trend  \
0                   2.5          0.0     0    Wednesday        January
1                   2.5          0.0     0       Sunday        January
2                   2.5          0.0     0     Saturday        January
3                   2.5          0.0     0     Saturday        January
4                   2.5          0.0     0       Sunday        January

   quarterly_trend trip_duration
0           2023Q1     12.166667
1           2023Q1     10.133333
2           2023Q1      7.450000
3           2023Q1      4.616667
4           2023Q1      9.366667

[5 rows x 23 columns]
```

3.1.10. Merge the zone data with trips data

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

zones = df_cleaned.merge(zones, left_on="PULocationID",
right_on="LocationID", how="left")
zones.head()
```

Output:

```
  tpep_pickup_datetime tpep_dropoff_datetime  passenger_count  trip_distance
\
0  2023-01-18 00:32:54   2023-01-18 00:45:04              1.0           3.21
1  2023-01-15 00:17:26   2023-01-15 00:27:34              1.0           1.50
2  2023-01-28 00:46:31   2023-01-28 00:53:58              1.0           1.30
3  2023-01-28 00:57:26   2023-01-28 01:02:03              1.0           1.10
4  2023-01-29 00:32:44   2023-01-29 00:42:06              2.0           0.99

   RatecodeID store_and_fwd_flag  PULocationID  DOLocationID payment_type  \
0         1.0                  N           158           161  Credit Card
1         1.0                  N           144           107  Credit Card
2         1.0                  N           249            90  Credit Card
3         1.0                  N           237            48  Credit Card
4         1.0                  N           148           113         Cash

   fare_amount  ...  Monthly_trend  quarterly_trend  trip_duration  OBJECTID
\
0         15.6  ...        January           2023Q1      12.166667     158.0
1         10.7  ...        January           2023Q1      10.133333     144.0
2          9.3  ...        January           2023Q1       7.450000     249.0
3          7.2  ...        January           2023Q1       4.616667     237.0
4         10.0  ...        January           2023Q1       9.366667     148.0

   Shape_Leng  Shape_Area                         zone  LocationID  \
0    0.054810    0.000186  Meatpacking/West Village West       158.0
1    0.027620    0.000047           Little Italy/NoLiTa       144.0
2    0.036384    0.000072                  West Village       249.0
3    0.042213    0.000096          Upper East Side South       237.0
4    0.039131    0.000070                Lower East Side       148.0

     borough                                           geometry
0  Manhattan  POLYGON ((982091.02 209596.704, 982318.344 209...
1  Manhattan  POLYGON ((985411.76 200369.518, 985342.573 200...
2  Manhattan  POLYGON ((983555.319 204876.901, 983469.158 20...
3  Manhattan  POLYGON ((993633.442 216961.016, 993507.232 21...
4  Manhattan  POLYGON ((988552.836 201677.665, 988387.669 20...

[5 rows x 30 columns]
```

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

```
# Group data by location and calculate the number of trips

total_trips =
df_clean.groupby("PULocationID")["PULocationID"].count().reset_index(name="t
rip_count")
total_trips
```

Output:

```
     PULocationID  trip_count
0               1         252
1               2           1
2               3          34
3               4        2320
4               5          15
..            ...         ...
252           261        9764
253           262       25539
254           263       36517
255           264       18043
256           265        1457

[257 rows x 2 columns]
```

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

```
# Merge trip counts back to the zones GeoDataFrame

zones = zones.merge(total_trips, left_on='PULocationID',
right_on='trip_count', how='left')
```

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

```
# Define figure and axis

fig, ax = plt.subplots(figsize=(10, 8))
ax.set_title("Taxi Trip data by Zone")

# Plot the map and display it

total_trips.plot(column="trip_count", cmap="Greens", legend=True, ax=ax)
plt.show()


# can you try displaying the zones DF sorted by the number of trips?

total_trips = total_trips.sort_values(by="trip_count", ascending=False)
total_trips
```

Output:

```
<Figure size 1000x800 with 1 Axes>
     PULocationID  trip_count
124           132       99299
228           237       89513
153           161       88331
227           236       79784
154           162       68222
..            ...         ...
148           156           1
104           111           1
99            105           1
```

```
56             59           1
236           245           1

[257 rows x 2 columns]
```

3.1.14. Conclude with results

3.2. Detailed EDA: Insights and Strategies

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

```
# Find routes which have the slowest speeds at different times of the day


df_cleaned["trip_duration_hr"] = df_cleaned["trip_duration"] / 3600
df_cleaned["speed_mph"] = df_cleaned["trip_distance"] /
df_cleaned["trip_duration_hr"]
df_cleaned = df_cleaned[(df_cleaned["speed_mph"] > 1) &
(df_cleaned["speed_mph"] < 100)]

df_cleaned["pickup_hour"] =
pd.to_datetime(df_cleaned["tpep_pickup_datetime"]).dt.hour

def time_of_day(hour):
    if 6 <= hour < 12:
        return "Morning"
    elif 12 <= hour < 16:
        return "Afternoon"
    elif 16 <= hour < 21:
        return "Evening"
    else:
        return "Night"

df_cleaned["time_of_day"] = df_cleaned["pickup_hour"].apply(time_of_day)
slow_routes = (df_cleaned.groupby(["PULocationID", "DOLocationID",
"time_of_day"]).agg(avg_speed=("speed_mph", "mean"),
trip_count=("speed_mph", "count")).reset_index())
slow_routes
```

Output:

```
      PULocationID  DOLocationID time_of_day  avg_speed  trip_count
0                1             1     Evening  30.997089           2
1                3           170   Afternoon  16.744186           1
2                4             4       Night  53.348611           4
3                4            79     Evening  20.093023           1
4                4            80       Night  11.822984           1
...            ...           ...         ...        ...         ...
2688           264           264     Morning  48.690263           8
2689           264           264       Night  51.782684           7
2690           265            67     Evening   8.503937           1
2691           265           265     Evening  80.000000           1
2692           265           265       Night  21.417302           5
```

```
[2693 rows x 5 columns]
```

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

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

df_cleaned["tpep_pickup_datetime"] =
pd.to_datetime(df_cleaned["tpep_pickup_datetime"])
df_cleaned["pickup_hour"] = df_cleaned["tpep_pickup_datetime"].dt.hour
hourly_trips =
df_cleaned.groupby("pickup_hour")["pickup_hour"].count().reset_index(name="t
rip_count")
busiest_hour = hourly_trips.loc[hourly_trips["trip_count"].idxmax()]
busiest_hour
```

Output:

```
pickup_hour     16
trip_count     547
Name: 16, dtype: int64
```

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

```
# Scale up the number of trips
busiest_5_hours = hourly_trips.sort_values(by='trip_count',
ascending=False).head(5)
print(busiest_5_hours)

# Fill in the value of your sampling fraction and use that to scale up the numbers
sample_fraction = 0.05
actual_trip_counts = busiest_5_hours['trip_count'] / sample_fraction
busiest_5_hours['actual_trip_count'] = actual_trip_counts
print("Top 5 busiest hours with estimated actual trip counts:")
print(busiest_5_hours)
```

Output:

```
    pickup_hour  trip_count
16           16         547
17           17         543
15           15         540
19           19         534
18           18         444
Top 5 busiest hours with estimated actual trip counts:
    pickup_hour  trip_count  actual_trip_count
16           16         547            10940.0
17           17         543            10860.0
15           15         540            10800.0
19           19         534            10680.0
```

```
18           18         444             8880.0
```

3.2.4. Compare hourly traffic on weekdays and weekends

```
# Compare traffic trends for the week days and weekends

df_cleaned["tpep_pickup_datetime"] =
pd.to_datetime(df_cleaned["tpep_pickup_datetime"])
df_cleaned["day_of_week"] = df_cleaned["tpep_pickup_datetime"].dt.dayofweek
df_cleaned["day_type"] = df_cleaned["day_of_week"].apply(lambda x: "Weekday"
if x < 5 else "Weekend")

Compare_weekday_weekend =
df_cleaned.groupby("day_type")["tpep_pickup_datetime"].count().reset_index()
Compare_weekday_weekend.columns = ["Day Type", "Trip Count"]
Compare_weekday_weekend
plt.figure(figsize=(8, 5))
sns.barplot(x="Day Type", y="Trip Count", data=Compare_weekday_weekend,
palette="tab10")

plt.xlabel("week_Type")
plt.ylabel("total_trips")
plt.title("Compare_weekday_weekend")
plt.grid(axis="y", linestyle="solid", alpha=0.7)
plt.show()
```

Output:

```
  Day Type  Trip Count
0  Weekday        4493
1  Weekend        1859
<Figure size 800x500 with 1 Axes>
```

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

```
# Find top 10 pickup and dropoff zones
# Find top 10 pickup zones
top_10_pickup_zones = df_cleaned['PULocationID'].value_counts().head(10)
print("Top 10 Pickup Zones:")
print(top_10_pickup_zones)
# Find top 10 dropoff zones
top_10_dropoff_zones = df_cleaned['DOLocationID'].value_counts().head(10)
print("\nTop 10 Dropoff Zones:")
print(top_10_dropoff_zones)
```

Output:

```
Top 10 Pickup Zones:
PULocationID
186    488
161    421
230    373
162    289
237    274
132    269
48     258
```

```
68     251
170    241
100    239
Name: count, dtype: int64

Top 10 Dropoff Zones:
DOLocationID
230    717
161    560
100    357
48     311
162    215
170    214
237    200
164    194
186    184
163    174
Name: count, dtype: int64
```

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

```
import geopandas as gpd
import numpy as np

# Reload the original taxi zones shapefile
zones_gdf = gpd.read_file('/content/taxi_zones/taxi_zones.shp')

# 1. Calculate the number of pickups for each PULocationID
pickup_counts =
df_cleaned.groupby('PULocationID').size().reset_index(name='pickup_count')
pickup_counts.rename(columns={'PULocationID': 'LocationID'}, inplace=True)

# 2. Calculate the number of dropoffs for each DOLocationID
dropoff_counts =
df_cleaned.groupby('DOLocationID').size().reset_index(name='dropoff_count')
dropoff_counts.rename(columns={'DOLocationID': 'LocationID'}, inplace=True)

# 3. Merge pickup_counts and dropoff_counts with zones_gdf
zone_activity = zones_gdf.merge(pickup_counts, on='LocationID', how='left')
zone_activity = zone_activity.merge(dropoff_counts, on='LocationID',
how='left')

# 4. Fill any NaN values with 0
zone_activity['pickup_count'].fillna(0, inplace=True)
zone_activity['dropoff_count'].fillna(0, inplace=True)

# Ensure counts are integers
zone_activity['pickup_count'] = zone_activity['pickup_count'].astype(int)
zone_activity['dropoff_count'] = zone_activity['dropoff_count'].astype(int)

# 5. Calculate the pickup_dropoff_ratio, handling division by zero
epsilon = 1e-6 # A small value to prevent division by zero
zone_activity['pickup_dropoff_ratio'] = zone_activity['pickup_count'] /
(zone_activity['dropoff_count'] + epsilon)

# 6. Replace infinite values with NaN (for cases where dropoff_count was 0
and epsilon still leads to large numbers)
zone_activity['pickup_dropoff_ratio'].replace([np.inf, -np.inf], np.nan,
inplace=True)
```

```
print("\nTop 10 zones with highest pickup/dropoff ratio:")
highest_ratio_zones = zone_activity.sort_values(by='pickup_dropoff_ratio',
ascending=False).head(10)
print(highest_ratio_zones[['zone', 'pickup_count', 'dropoff_count',
'pickup_dropoff_ratio']])

print("\nTop 10 zones with lowest pickup/dropoff ratio (excluding zeros/NaNs
where pickup count is also zero):\n(A ratio of 0 indicates either zero pic

# ... (code truncated for brevity)
```

Output:

```
Top 10 zones with highest pickup/dropoff ratio:
                                    zone  pickup_count  dropoff_count  \
31                              Bronxdale             3              0
62                          Cypress Hills             3              0
196                         Richmond Hill             2              0
138                             Laurelton             2              0
116                       Hammels/Arverne             2              0
43                 Charleston/Tottenville             1              0
199  Riverdale/North Riverdale/Fieldston             1              0
83     Eltingville/Annadale/Prince's Bay             1              0
52                          College Point             1              0
123                          Howard Beach             1              0

     pickup_dropoff_ratio
31              3000000.0
62              3000000.0
196             2000000.0
138             2000000.0
116             2000000.0
43              1000000.0
199             1000000.0
83              1000000.0
52              1000000.0
123             1000000.0

Top 10 zones with lowest pickup/dropoff ratio (excluding zeros/NaNs where
pickup count is also zero):
(A ratio of 0 indicates either zero pickups or a very high number of
dropoffs compared to pickups)
                          zone  pickup_count  dropoff_count  \
255  Williamsburg (South Side)             1              9
241       Van Nest/Morris Park             1              5
13                     Bay Ridge            1              4
36               Bushwick South             1              4
134          Kew Gardens Hills              1              4
227           Sunset Park West              1              4
254  Williamsburg (North Side)              2              8
65           DUMBO/Vinegar Hill             2              8
231    Two Bridges/Seward Park              5             19
6                       Astoria             4             15

     pickup_dropoff_ratio
255              0.111111
241              0.200000
13               0.250000
36               0.250000
134              0.250000
227              0.250000
254              0.250000
```

```
65              0.250000
231             0.263158
6               0.266667
```

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

```
# Filter for night hours (11 PM to 5 AM)
night_hours_df = df_cleaned[((df_cleaned['hour'] >= 23) |
(df_cleaned['hour'] <= 5))]

# Find top 10 pickup zones during night hours
top_10_night_pickup_zones =
night_hours_df['PULocationID'].value_counts().head(10)
print("\nTop 10 Pickup Zones during Night Hours (11 PM - 5 AM):")
print(top_10_night_pickup_zones)

# Find top 10 dropoff zones during night hours
top_10_night_dropoff_zones =
night_hours_df['DOLocationID'].value_counts().head(10)
print("\nTop 10 Dropoff Zones during Night Hours (11 PM - 5 AM):")
print(top_10_night_dropoff_zones)
```

Output:

```
Top 10 Pickup Zones during Night Hours (11 PM - 5 AM):
PULocationID
79     46
230    40
249    39
48     37
132    34
186    28
148    28
144    24
68     23
164    21
Name: count, dtype: int64

Top 10 Dropoff Zones during Night Hours (11 PM - 5 AM):
DOLocationID
230    45
68     32
249    29
148    28
79     26
170    22
132    21
164    21
48     19
186    19
Name: count, dtype: int64
```

3.2.8. Find the revenue share for nighttime and daytime hours

```
# Calculate the revenue share for night-time and day-time hours

def time_of_day_category(hour):
    if 6 <= hour < 18:  # Daytime: 6 AM to 5:59 PM
        return "Daytime"
    else:  # Nighttime: 6 PM to 5:59 AM (including 23, 0, 1, 2, 3, 4, 5)
        return "Nighttime"

df_cleaned['time_category'] = df_cleaned['hour'].apply(time_of_day_category)

revenue_by_time_category =
df_cleaned.groupby('time_category')['total_amount'].sum().reset_index()
total_revenue = df_cleaned['total_amount'].sum()
revenue_by_time_category['revenue_share'] =
(revenue_by_time_category['total_amount'] / total_revenue) * 100

print("Revenue Share by Time Category:")
print(revenue_by_time_category)

plt.figure(figsize=(7, 7))
plt.pie(revenue_by_time_category['revenue_share'],
labels=revenue_by_time_category['time_category'], autopct='%1.1f%%',
startangle=90, colors=['skyblue', 'darkblue'])
plt.title('Revenue Share by Day vs. Night Hours')
plt.axis('equal') # Equal aspect ratio ensures that pie is drawn as a
circle.
plt.show()
```

Output:

```
Revenue Share by Time Category:
  time_category  total_amount  revenue_share
0       Daytime     100348.06      65.316238
1     Nighttime      53286.11      34.683762


<Figure size 700x700 with 1 Axes>
```

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

# Avoid division by zero for trip_distance and passenger_count
df_cleaned['trip_distance_safe'] = df_cleaned['trip_distance'].replace(0,
np.nan)
df_cleaned['passenger_count_safe'] =
df_cleaned['passenger_count'].replace(0, np.nan)

# Calculate fare per mile
df_cleaned['fare_per_mile'] = df_cleaned['fare_amount'] /
df_cleaned['trip_distance_safe']

# Calculate fare per mile per passenger
df_cleaned['fare_per_mile_per_passenger'] = df_cleaned['fare_per_mile'] /
df_cleaned['passenger_count_safe']

# Group by passenger_count and calculate the average fare per mile per
passenger
average_fare_per_mile_per_passenger =
df_cleaned.groupby('passenger_count')['fare_per_mile_per_passenger'].mean().
reset_index()
```

```
print("Average Fare per Mile per Passenger for different Passenger Counts:")
print(average_fare_per_mile_per_passenger)
```

Output:

```
Average Fare per Mile per Passenger for different Passenger Counts:
   passenger_count  fare_per_mile_per_passenger
0              0.0                          NaN
1              1.0                   168.084344
2              2.0                   118.400509
3              3.0                   102.919064
4              4.0                   118.915843
5              5.0                     6.108371
6              6.0                     9.829733
```

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

```
# Compare the average fare per mile for different days and for different
times of the day

# Ensure 'trip_distance_safe' and 'fare_amount' are available (created in
previous step)
# Calculate average fare per mile by hour of the day
average_fare_per_mile_hourly =
df_cleaned.groupby('hour')['fare_per_mile'].mean().reset_index()
print("\nAverage Fare per Mile by Hour of the Day:")
print(average_fare_per_mile_hourly)

# Calculate average fare per mile by day of the week
average_fare_per_mile_daily =
df_cleaned.groupby('daily_trend')['fare_per_mile'].mean().reset_index()
print("\nAverage Fare per Mile by Day of the Week:")
print(average_fare_per_mile_daily)

# Plotting for visualization (optional but good for analysis)
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.lineplot(x='hour', y='fare_per_mile', data=average_fare_per_mile_hourly)
plt.title('Average Fare per Mile by Hour')
plt.xlabel('Hour of Day')
plt.ylabel('Average Fare per Mile')

plt.subplot(1, 2, 2)
sns.barplot(x='daily_trend', y='fare_per_mile',
data=average_fare_per_mile_daily, order=['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.title('Average Fare per Mile by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Average Fare per Mile')

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

Output:

```
Average Fare per Mile by Hour of the Day:
    hour  fare_per_mile
```

```
0      0     198.517424
1      1     206.286330
2      2      70.993490
3      3     245.299281
4      4     408.081161
5      5     886.696040
6      6     342.034653
7      7     403.124751
8      8     239.143417
9      9     171.802424
10    10     226.689415
11    11     124.168330
12    12     128.033665
13    13      97.514999
14    14      97.205704
15    15     213.437501
16    16     234.097871
17    17     163.676916
18    18     167.996400
19    19     186.511034
20    20     176.516124
21    21     309.391871
22    22     100.873796
23    23     361.789907

Average Fare per Mile by Day of the Week:
  daily_trend  fare_per_mile
0      Friday     227.774753
1      Monday     201.611850
2    Saturday     163.182969
3      Sunday     236.535228
4    Thursday     181.261386
5     Tuesday     191.055515
6   Wednesday     149.175302


<Figure size 1200x600 with 2 Axes>
```

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

```
# Compare fare per mile for different vendors
# Create a copy of the original df (which contains VendorID) to apply cleaning for this specific analysis
df_vendor_analysis = df.copy()

# --- Reapply essential cleaning steps to df_vendor_analysis ---

# Combine 'Airport_fee' and 'airport_fee' (if they exist and are different)
if 'Airport_fee' in df_vendor_analysis.columns and 'airport_fee' in df_vendor_analysis.columns:
    df_vendor_analysis['airport_fee'] = df_vendor_analysis['airport_fee'].combine_first(df_vendor_analysis['Airport_fee'])
    df_vendor_analysis.drop(columns=['Airport_fee'], inplace=True)
```

```
# Fix negative monetary values
monetary_cols = ['fare_amount', 'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
'improvement_surcharge', 'total_amount', 'congestion_surcharge', 'airport_fee']
for col in monetary_cols:
    if col in df_vendor_analysis.columns:
        df_vendor_analysis.loc[df_vendor_analysis[col] < 0, col] = 0

# Handle missing values (using mode/median as done previously)
# Note: This is a simplified imputation based on previous cells; ideally, this would
be a more robust pipeline.
if 'passenger_count' in df_vendor_analysis.columns:

df_vendor_analysis['passenger_count'].fillna(df_vendor_analysis['passenger_cou
nt'].mode()[0], inplace=True)
if 'RatecodeID' in df_vendor_analysis.columns:

df_vendor_analysis['RatecodeID'].fillna(df_vendor_analysis['RatecodeID'].median(
), inplace=True)
if 'congestion_surcharge' in df_vendor_analysis.columns:

df_vendor_analysis['congestion_surcharge'].fillna(df_vendor_analysis['congestio
n_surcharge'].mode()[0], inplace=True)
if 'store_and_fwd_flag' in df_vendor_analysis.columns:

df_vendor_analysis['store_and_fwd_flag'].fillna(df_vendor_analysis['store_and_fw
d_flag'].mode()[0], inplace=True)
if 'airport_fee' in df_vendor_analysis.columns:
    df_vendor_analysis['airport_fee'].fillna(df_vendor_analysis['airport_fee'].mean(),
inplace=True)

# Outlier handling (based on previous steps)
df_vendor_analysis = d

# ... (code truncated for brevity)
```

Output:

```
Average Fare per Mile by Vendor and Hour of the Day:
    VendorID  hour  fare_per_mile
0          1     0       6.660351
1          1     1       6.775296
2          1     2       6.621365
3          1     3       6.689960
4          1     4       7.340049
5          1     5       6.677471
6          1     6       6.876311
7          1     7       7.262159
8          1     8       8.016312
```

```
9          1     9       8.348247
10         1    10       8.405434
11         1    11       8.702792
12         1    12       8.890682
13         1    13       8.897975
14         1    14       8.654121
15         1    15       8.709098
16         1    16       8.809963
17         1    17       8.697073
18         1    18       8.512348
19         1    19       8.069373
20         1    20       7.454513
21         1    21       7.348388
22         1    22       7.042406
23         1    23       6.963180
24         2     0      12.036165
25         2     1       9.875653
26         2     2      11.592672
27         2     3      12.827804
28         2     4      24.006160
29         2     5      16.971574
30         2     6      10.967500
31         2     7      11.853906
32         2     8      10.908970
33         2     9      11.656888
34         2    10      11.227953
35         2    11      12.578481
36         2    12      12.660066
37         2    13      12.570381
38         2    14      12.883882
39         2    15      14.299395
40         2    16      14.766028
41         2    17      13.254517
42         2    18      12.196349
43         2    19      12.619066
44         2    20      10.715886
45         2    21      11.199987
46         2    22       9.841226
47         2    23      10.912395


<Figure size 1400x700 with 1 Axes>
```

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

```
# Defining distance tiers
# Define a function to categorize trip distance into tiers
def categorize_distance(distance):
    if distance <= 2:
        return '0-2 miles'
    elif 2 < distance <= 5:
        return '2-5 miles'
    else:
        return '5+ miles'

# Apply the function to create the 'distance_tier' column in
df_vendor_analysis
df_vendor_analysis['distance_tier'] =
df_vendor_analysis['trip_distance'].apply(categorize_distance)
```

```
# Calculate the mean fare_per_mile for each VendorID and distance_tier
tiered_fare_per_mile = df_vendor_analysis.groupby(['VendorID',
'distance_tier'])['fare_per_mile'].mean().reset_index()

print("Average Fare per Mile by Vendor and Distance Tier:")
print(tiered_fare_per_mile)
```

Output:

```
Average Fare per Mile by Vendor and Distance Tier:
   VendorID distance_tier  fare_per_mile
0         1     0-2 miles       9.932235
1         1     2-5 miles       6.392026
2         1      5+ miles       4.422839
3         2     0-2 miles      17.675221
4         2     2-5 miles       6.496464
5         2      5+ miles       4.447719
```

3.2.13. Analyse the tip percentages

```
# Analyze tip percentages based on distances, passenger counts and pickup
times

# Calculate tip percentage, handling division by zero
df_cleaned['tip_percentage'] = np.where(
    df_cleaned['total_amount'] > 0,
    (df_cleaned['tip_amount'] / df_cleaned['total_amount']) * 100,
    0
)

# Trim extreme tip percentages for better visualization (e.g., cap at 50%)
df_cleaned['tip_percentage'] = df_cleaned['tip_percentage'].clip(upper=50)

# Analysis by Trip Distance
# Bin trip distances for better visualization if needed, or use scatter plot
directly
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
sns.scatterplot(x='trip_distance', y='tip_percentage', data=df_cleaned,
alpha=0.1)
plt.title('Tip Percentage vs. Trip Distance')
plt.xlabel('Trip Distance (miles)')
plt.ylabel('Tip Percentage (%)')
plt.xscale('log') # Use log scale for distance to handle wide range

# Analysis by Passenger Counts
plt.subplot(1, 3, 2)
sns.boxplot(x='passenger_count', y='tip_percentage', data=df_cleaned)
plt.title('Tip Percentage vs. Passenger Count')
plt.xlabel('Passenger Count')
plt.ylabel('Tip Percentage (%)')

# Analysis by Time of Pickup (Hour)
plt.subplot(1, 3, 3)
sns.lineplot(x='hour', y='tip_percentage',
data=df_cleaned.groupby('hour')['tip_percentage'].mean().reset_index(),
marker='o')
```

```
plt.title('Average Tip Percentage by Hour of Day')
plt.xlabel('Hour of Day')
plt.ylabel('Average Tip Percentage (%)')
plt.xticks(range(0, 24))

plt.tight_layout()
plt.show()

print("\nAverage Tip Percentage by Passenger Count:")
print(df_cleaned.groupby('passenger_count')['tip_percentage'].mean().reset_i
ndex())

print("\nAverage Tip Percentage by Hour of Day:")
print(df_cleaned.groupby('hour')['tip_percentage'].mean().reset_index())
# Compare trips with tip percentage < 10% to trips with tip percentage > 25%

# Filter for low tip percentage trips
low_tip_trips = df_cleaned[df_cleaned['tip_percentage'] < 10]

# Filter for high tip percentage trips
high_tip_trips = df_cleaned[df_cleaned['tip_percentage'] > 25]

print("\nDescriptive statistics for trips with tip percentage < 10%:")
print(low_tip_trips[['trip_distance', 'fare_amount', 'passenger_count',
'hour']].describe())

print("\nDescriptive statistics for trips with tip percentage > 25%:")
print(high_tip_trips[['trip_distance', 'fare_amount', 'passenger_count',
'hour']].describe())

# Optional: Visualize distributions
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.histplot(low_tip_trips['trip_distance'], color='blue', label='Low Tip',
kde=True, stat='density', alpha=0.5)
sns.histplot(high_tip_trips['trip_distance'], color='red', label='High Tip',
kde=True, stat='density', alpha=0.5)
plt.title('Trip Distance Distribution: Low vs. High Tips')
plt.xlabel('Trip Distance (miles)')
plt.legend()

plt.subplot(1, 2, 2)
sns.boxplot(x='passenger_count', y='tip_percentage',
data=df_cleaned[df_cleaned['tip_percentage'] < 10], color='blue',
ax=plt.gca(), whis=2.0)
sns.boxplot(x='passenger_count', y='tip_percentage',
data=df_cleaned[df_cleaned['tip_percentage'] > 25], color='red',
ax=plt.gca(), whis=2.0)
plt.title('Tip Percentage by Passenger Count: Low vs. High Tips')
plt.xlabel('Passenger Count')
plt.ylabel('Tip Percentage (%)')
plt.tight_layout()
plt.show()
```

Output:

```
<Figure size 1500x500 with 3 Axes>


Average Tip Percentage by Passenger Count:
   passenger_count  tip_percentage
0              0.0        8.289073
1              1.0        6.596297
```

```
2              2.0        6.979415
3              3.0        7.852340
4              4.0        5.381485
5              5.0        6.746218
6              6.0        6.116392

Average Tip Percentage by Hour of Day:
    hour  tip_percentage
0      0        4.450758
1      1        4.978855
2      2        4.567379
3      3        1.604579
4      4        3.168346
5      5        2.830304
6      6        4.734719
7      7        3.770095
8      8        6.337998
9      9        7.330783
10    10        6.258843
11    11        6.111523
12    12        7.733858
13    13        6.587106
14    14        7.276071
15    15        7.810841
16    16        7.257948
17    17        7.221726
18    18        6.390737
19    19        8.316533
20    20        6.379022
21    21        5.276365
22    22        6.654175
23    23        4.697552



Descriptive statistics for trips with tip percentage < 10%:
       trip_distance  fare_amount  passenger_count         hour
count    4099.000000  4099.000000      4099.000000  4099.000000
mean        1.263135    16.206460         1.452061    14.122957
std         2.954263    18.333523         1.029498     5.593214
min         0.010000     0.000000         0.000000     0.000000
25%         0.040000     4.400000         1.000000    11.000000
50%         0.300000    10.700000         1.000000    15.000000
75%         0.930000    19.800000         1.000000    18.000000
max        27.920000   237.900000         6.000000    23.000000

Descriptive statistics for trips with tip percentage > 25%:
       trip_distance  fare_amount  passenger_count       hour
count      55.000000    55.000000        55.000000  55.000000
mean        1.265818    14.267273         1.363636  13.127273
std         4.080389    23.404835         0.930226   6.274742
min         0.010000     3.000000         1.000000   0.000000
25%         0.100000     4.050000         1.000000   9.500000
50%         0.200000     9.300000         1.000000  15.000000
75%         0.980000    15.250000         1.000000  17.000000
max        29.910000   157.000000         5.000000  23.000000


<Figure size 1400x600 with 2 Axes>
```

3.2.14. Analyse the trends in passenger count

```
# See how passenger count varies across hours and days

# Average passenger count by hour of day
average_pass_count_hourly =
df_cleaned.groupby('hour')['passenger_count'].mean().reset_index()

# Average passenger count by day of week
average_pass_count_daily =
df_cleaned.groupby('daily_trend')['passenger_count'].mean().reset_index()

print("\nAverage Passenger Count by Hour of Day:")
print(average_pass_count_hourly)

print("\nAverage Passenger Count by Day of Week:")
print(average_pass_count_daily)

# Plotting for visualization
plt.figure(figsize=(15, 6))

plt.subplot(1, 2, 1)
sns.lineplot(x='hour', y='passenger_count', data=average_pass_count_hourly,
marker='o')
plt.title('Average Passenger Count by Hour of Day')
plt.xlabel('Hour of Day')
plt.ylabel('Average Passenger Count')
plt.xticks(range(0, 24))
plt.grid(True, linestyle='--', alpha=0.7)

plt.subplot(1, 2, 2)
sns.barplot(x='daily_trend', y='passenger_count',
data=average_pass_count_daily, order=['Monday', 'Tuesday', 'Wednesday',
'Thursday', 'Friday', 'Saturday', 'Sunday'])
plt.title('Average Passenger Count by Day of Week')
plt.xlabel('Day of Week')
plt.ylabel('Average Passenger Count')
plt.grid(axis='y', linestyle='--', alpha=0.7)

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

Output:

```
Average Passenger Count by Hour of Day:
    hour  passenger_count
0      0         1.454545
1      1         1.705426
2      2         1.419355
3      3         1.860000
4      4         1.302326
5      5         1.296296
6      6         1.266667
7      7         1.304878
8      8         1.422360
9      9         1.273196
10    10         1.410169
11    11         1.439169
12    12         1.474537
13    13         1.482270
14    14         1.450882
15    15         1.416667
16    16         1.418647
17    17         1.491713
18    18         1.452703
```

```
19    19         1.355805
20    20         1.514706
21    21         1.613333
22    22         1.413953
23    23         1.482234

Average Passenger Count by Day of Week:
  daily_trend  passenger_count
0      Friday         1.434354
1      Monday         1.409020
2    Saturday         1.568733
3      Sunday         1.587131
4    Thursday         1.398305
5     Tuesday         1.404281
6   Wednesday         1.340573


<Figure size 1500x600 with 2 Axes>
```

3.2.15. Analyse the variation of passenger counts across zones

```
# How does passenger count vary across zones
average_pass_count_by_zone =
df_cleaned.groupby('PULocationID')['passenger_count'].mean().reset_index()
average_pass_count_by_zone.rename(columns={'PULocationID': 'LocationID',
'passenger_count': 'average_passenger_count'}, inplace=True)

print("Top 10 zones by average passenger count:")
print(average_pass_count_by_zone.sort_values(by='average_passenger_count',
ascending=False).head(10))

print("\nBottom 10 zones by average passenger count:")
print(average_pass_count_by_zone.sort_values(by='average_passenger_count',
ascending=True).head(10))
# For a more detailed analysis, we can use the zones_with_trips GeoDataFrame
# Create a new column for the average passenger count in each zone.

# Merge average_pass_count_by_zone into zones_gdf
zones_gdf = zones_gdf.merge(average_pass_count_by_zone, on='LocationID',
how='left')

# Display the head of the updated zones_gdf to verify
zones_gdf.head()
```

Output:

```
Top 10 zones by average passenger count:
     LocationID  average_passenger_count
145         218                 2.666667
148         224                 2.666667
5            12                 2.333333
112         166                 2.333333
60           88                 2.142857
150         226                 2.142857
34           52                 2.000000
20           36                 2.000000
151         228                 2.000000
176         258                 2.000000

Bottom 10 zones by average passenger count:
```

```
    LocationID  average_passenger_count
0            1                      1.0
1            3                      1.0
3            7                      1.0
4           10                      1.0
7           14                      1.0
10          20                      1.0
9           17                      1.0
8           16                      1.0
14          26                      1.0
11          22                      1.0


   OBJECTID  Shape_Leng  Shape_Area                     zone  LocationID  \
0         1    0.116357    0.000782           Newark Airport           1
1         2    0.433470    0.004866              Jamaica Bay           2
2         3    0.084341    0.000314  Allerton/Pelham Gardens           3
3         4    0.043567    0.000112             Alphabet City           4
4         5    0.092146    0.000498             Arden Heights           5

         borough                                           geometry  \
0            EWR  POLYGON ((933100.918 192536.086, 933091.011 19...
1         Queens  MULTIPOLYGON (((1033269.244 172126.008, 103343...
2          Bronx  POLYGON ((1026308.77 256767.698, 1026495.593 2...
3      Manhattan  POLYGON ((992073.467 203714.076, 992068.667 20...
4  Staten Island  POLYGON ((935843.31 144283.336, 936046.565 144...

   average_passenger_count
0                 1.000000
1                      NaN
2                 1.000000
3                 1.777778
4                      NaN
```

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

```
# How often is each surcharge applied?

surcharge_columns = [
    'extra',
    'mta_tax',
    'tip_amount',
    'tolls_amount',
    'improvement_surcharge',
    'congestion_surcharge',
    'airport_fee'
]

surcharge_prevalence = pd.DataFrame(columns=['Surcharge', 'Count_Applied',
'Percentage_Applied'])

for col in surcharge_columns:
    # Count cases where surcharge is applied (value > 0)
    count_applied = df_cleaned[df_cleaned[col] > 0].shape[0]
    # Calculate percentage
    percentage_applied = (count_applied / len(df_cleaned)) * 100
```

```
    # Append to the DataFrame
    surcharge_prevalence = pd.concat([
        surcharge_prevalence,
        pd.DataFrame([{'Surcharge': col, 'Count_Applied': count_applied,
'Percentage_Applied': percentage_applied}])
    ], ignore_index=True)

print("Prevalence of Surcharges:")
print(surcharge_prevalence)
```

Output:

```
Prevalence of Surcharges:
               Surcharge Count_Applied  Percentage_Applied
0                  extra          3095           48.724811
1                mta_tax          5841           91.955290
2             tip_amount          2808           44.206549
3           tolls_amount           251            3.951511
4  improvement_surcharge          5919           93.183249
5   congestion_surcharge          5154           81.139798
6            airport_fee           352            5.541562
```

4. Conclusions

4.1. Final Insights and Recommendations

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

Based on the demand patterns and operational inefficiencies identified in the analysis, the following recommendations are proposed:

Proactive Deployment:
• Increase taxi deployment during peak hours (6-7 PM on weekdays, especially Thursdays and Wednesdays) in high-demand zones including LaGuardia Airport (PULocationID 132), Upper East Side South (237), and Midtown East (161)
• Ensure consistent availability during night hours (11 PM-5 AM) in active entertainment zones (PULocationID 79, 230)
• Use slow route data identified in the analysis for dynamic rerouting to avoid traffic bottlenecks and improve trip efficiency

Dynamic Routing Optimization:
• Implement real-time routing systems that avoid identified slow corridors during peak hours, potentially saving 10-15% trip time
• Utilize historical traffic patterns to predict congestion and suggest alternative routes proactively
• Deploy more vehicles to leisure and dining districts on Friday-Saturday evenings when weekend demand patterns differ from 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.

Strategic positioning recommendations based on analysis of trip trends, zone-level demand patterns, and pickup/dropoff imbalances:

Zone-Based Positioning:
• Encourage drivers to quickly return to high pickup-to-dropoff ratio zones (Bronxdale, Cypress Hills) after completing drop-offs, as these zones consistently show demand exceeding supply
• Direct drivers from low pickup-to-dropoff ratio zones (Williamsburg South Side, Bay Ridge) to nearby high-demand pickup areas to minimize empty mileage and deadhead time
• Maintain dedicated presence at airport zones (LaGuardia, JFK) throughout the day given their consistent high-volume demand

Time-Based Positioning:
• Position vehicles in residential outer boroughs during 6-7 AM to capture morning Manhattan-bound commute trips
• Shift fleet concentration to business districts (Midtown, Financial District) during 5-7 PM for evening rush hour
• Focus on entertainment zones (Times Square, Greenwich Village) during late night hours (11 PM-2 AM)

Adaptive Systems:
• Implement real-time positioning adjustments based on live demand signals and traffic conditions
• Use predictive analytics to anticipate demand surges 30-60 minutes in advance and reposition accordingly
• Balance fleet distribution across zones to maintain service levels while maximizing revenue per vehicle

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

```
!pip install fpdf
import os
from fpdf import FPDF

# Save the PDF to the current working directory instead of Desktop
pdf_file_path = "\\Users\\neha.gupta5\\Downloads\\"  # added filename and
escaped backslashes


class PDF(FPDF):
    def header(self):
        self.set_font("Arial", "B", 14)
```

```
        self.cell(200, 10, "NYC Taxi Data Analysis Report", ln=True,
align="C")
        self.ln(10)

    def chapter_title(self, title):
        self.set_font("Arial", "B", 12)
        self.cell(0, 10, title, ln=True, align="L")
        self.ln(5)

    def chapter_body(self, body):
        self.set_font("Arial", "", 11)
        self.multi_cell(0, 10, body)
        self.ln()

    def add_image(self, image_path, width=150):
        if os.path.exists(image_path):
            self.image(image_path, x=30, w=width)
            self.ln(10)
        else:
            self.chapter_body(f"⚠️ Missing Image: {image_path}")

    # The __init__ method should be at the same indentation level as other
methods
    def __init__(self):
        super().__init__()  # Call the parent class's __init__


# Now, outside the class definition:
pdf = PDF()
# ... (Your PDF generation code remains the same) ...

pdf.output(pdf_file_path)
print(f"✅ Report saved as: {pdf_file_path}")
!pip install reportlab
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas
import matplotlib.pyplot as plt
import os

# Define the PDF file path
pdf_path = "\\Users\\neha.gupta5\\Downloads\\NYC_Taxi_Analysis.pdf"

# Create a PDF document
c = canvas.Canvas(pdf_path, pagesize=letter)
c.setFont("Helvetica", 12)

# Add Title
c.drawString(100, 750, "NYC Yellow Taxi Data Analysis Report")
c.drawString(100, 730, "-------------------------------------------")

# Example: Save a matplotlib figure
plt.figure(figsize=(6,4))
plt.plot([1,2,3], [4,5,6])  # Replace this with your actual visualization
plt.title("Sample Visualization")
plt.savefig("\\Users\\neha.gupta5\\Downloads\\plot.png")  # Save figure

# Insert the image into the PDF
c.drawImage("\\Users\\neha.gupta5\\Downloads\\plot.png", 100, 500,
width=400, height=250)

# Save and close the PDF
c.save()
```

```
print(f"✅ PDF Report saved at: {pdf_path}")
```

Output:

```
Collecting fpdf
  Downloading fpdf-1.7.2.tar.gz (39 kB)
  Preparing metadata (setup.py) ... [?25l[?25hdone
Building wheels for collected packages: fpdf
  Building wheel for fpdf (setup.py) ... [?25l[?25hdone
  Created wheel for fpdf: filename=fpdf-1.7.2-py2.py3-none-any.whl
size=40704
sha256=f380fe65d4e151b49641ec2b3ab50dc679b20f5dd33fa74f0f7bf0ca3d828940
  Stored in directory:
/root/.cache/pip/wheels/6e/62/11/dc73d78e40a218ad52e7451f30166e94491be013a78
50b5d75
Successfully built fpdf
Installing collected packages: fpdf
Successfully installed fpdf-1.7.2

✅ Report saved as: \Users\neha.gupta5\Downloads\


Requirement already satisfied: reportlab in /usr/local/lib/python3.12/dist-
packages (4.4.10)
Requirement already satisfied: pillow>=9.0.0 in
/usr/local/lib/python3.12/dist-packages (from reportlab) (11.3.0)
Requirement already satisfied: charset-normalizer in
/usr/local/lib/python3.12/dist-packages (from reportlab) (3.4.4)

✅ PDF Report saved at: \Users\neha.gupta5\Downloads\NYC_Taxi_Analysis.pdf


<Figure size 600x400 with 1 Axes>
```