

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

This approach processes one file at a time to avoid memory overload, extracts date and hour from each record, and samples a small percentage (0.75%) from each hour to maintain temporal trends. The sampled data is then combined across all months into a single DataFrame for analysis.

2. Data Cleaning

2.1. Fixing Columns

2.1.1. Fix the index

```
[ ] # Fix the index and drop any columns that are not needed
df.reset_index(inplace=True)

columns_to_drop = ['store_and_fwd_flag']
df.drop(columns=columns_to_drop, inplace=True)

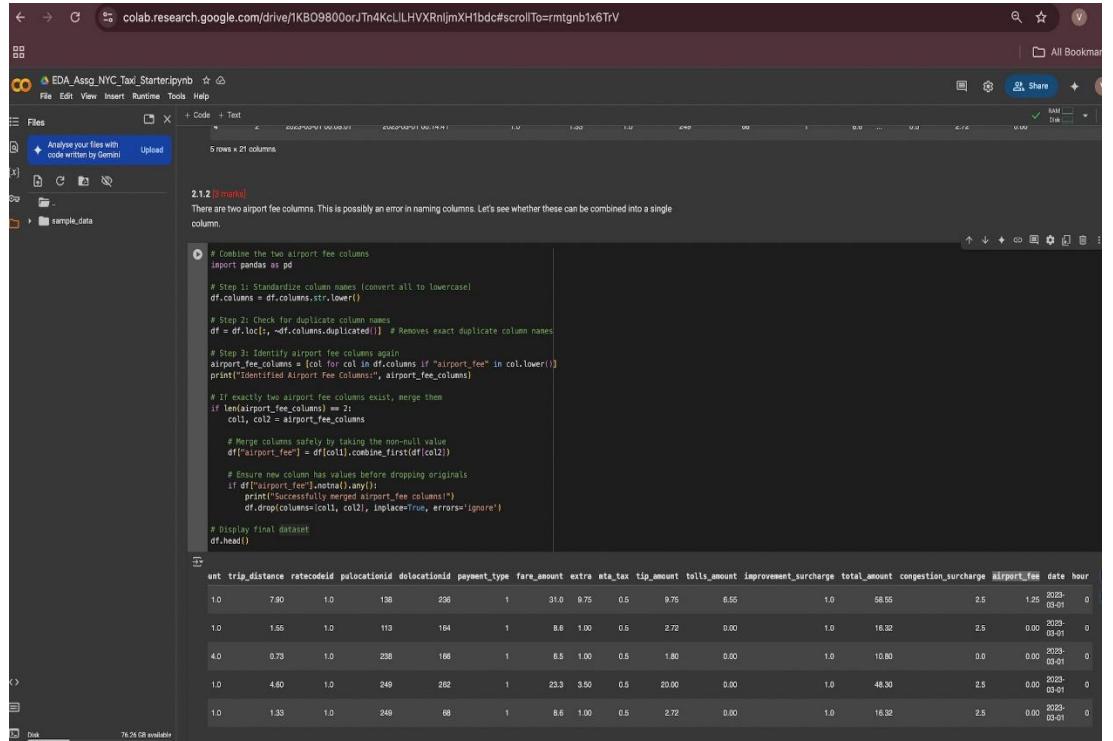
[ ] df.head()
```

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	PULocationID	DOLocationID	payment_type	fare_amount	... mta_tax	tip_amount	total_amount
0	1	2023-03-01 00:06:18	2023-03-01 00:21:33	1.0	7.90	1.0	138	236	1	31.0	...	0.5	9.75
1	2	2023-03-01 00:46:10	2023-03-01 00:45:44	1.0	1.55	1.0	113	164	1	8.6	...	0.5	2.72
2	2	2023-03-01 00:08:41	2023-03-01 00:12:28	4.0	0.75	1.0	238	166	1	6.5	...	0.5	1.80
3	1	2023-03-01 00:37:43	2023-03-01 00:55:51	1.0	4.60	1.0	249	262	1	23.0	...	0.5	20.00
4	2	2023-03-01 00:08:01	2023-03-01 00:14:41	1.0	1.33	1.0	249	68	1	8.6	...	0.5	2.72

5 rows x 21 columns

Reset the index of the DataFrame to ensure it is sequential after any changes, and drops unnecessary columns (such as 'store_and_fwd_flag') to clean up the dataset and keep only the relevant data for analysis.

2.1.2. Combine the two airport_fee columns



The screenshot shows a Google Colab notebook titled "EDA Assg_NYC_Taxi_Starter.ipynb". The code cell contains the following Python script:

```
# Combine the two airport fee columns
import pandas as pd

# Step 1: Standardize column names (convert all to lowercase)
df.columns = df.columns.str.lower()

# Step 2: Check for duplicate column names
df = df.loc[:, ~df.columns.duplicated()] # Removes exact duplicate column names

# Step 3: Identify airport_fee columns again
airport_fee_columns = [col for col in df.columns if "airport_fee" in col.lower()]
print("Identified Airport Fee Columns:", airport_fee_columns)

# If exactly two airport_fee columns exist, merge them
if len(airport_fee_columns) == 2:
    col1, col2 = airport_fee_columns

    # Merge columns surely by taking the non-null value
    df["airport_fee"] = df[[col1, col2]].combine_first(df[col2])

    # Ensure new column has values before dropping originals
    if df["airport_fee"].notna().any():
        print("Successfully merged airport_fee columns")
        df.drop(columns=[col1, col2], inplace=True, errors='ignore')

# Display final dataset
df.head()
```

The output of the code is a table showing the first 5 rows of the dataset:

nt	trip_distance	ratecodeid	pickuplocationid	deleocationid	payment_type	fare_amount	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount	congestion_surcharge	airport_fee	date	hour
1.0	7.80	1.0	198	236	1	31.0	9.75	0.5	9.75	6.55	1.0	58.95	2.5	1.25	2023-03-01	0
1.0	1.55	1.0	113	164	1	8.6	1.00	0.5	2.72	0.00	1.0	18.32	2.5	0.00	2023-03-01	0
4.0	0.73	1.0	238	186	1	8.5	1.00	0.5	1.80	0.00	1.0	10.80	0.0	0.00	2023-03-01	0
1.0	4.80	1.0	249	282	1	23.3	3.50	0.5	20.00	0.00	1.0	48.30	2.5	0.00	2023-03-01	0
1.0	1.33	1.0	249	68	1	8.6	1.00	0.5	2.72	0.00	1.0	18.32	2.5	0.00	2023-03-01	0

This approach standardizes column names to lowercase for consistency, removes any exact duplicate columns, and identifies multiple "airport_fee" columns. If two such columns exist, it merges them by retaining non-null values before dropping the original duplicate columns.

2.2. Handling Missing Values

2.2.1. Find the proportion of missing values in each column

```

EDA_Asgg_NYC_Taxi_Starter.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
2.2.1 [0 marks]
Find the proportion of missing values in each column

# Find the proportion of missing values in each column
missing_proportion = df.isnull().sum() / len(df)
print(missing_proportion)

vendoid          0.000000
tpep_pickup_datetime 0.000000
tpep_dropoff_datetime 0.000000
passenger_count   0.031316
trip_distance     0.000000
ratecodeid        0.031316
pickuplocationid 0.000000
pickupdatetime    0.000000
passenger_type    0.000000
fare_amount       0.000000
extra_fare       0.000000
mta_tax           0.000000
tip_amount        0.000000
tolls_amount      0.000000
improvement_surcharge 0.000000
total_amount      0.000000
congestion_surcharge 0.031316
airport_fee       0.111499
date              0.000000
hour              0.000000
dtype: float64

2.2.2 [0 marks]
Handling missing values in passenger_count

[ ] # Display the rows with null values
missing_rows = df[df.isnull().any(axis=1)]
print(missing_rows)

# Impute null values in 'passenger_count'
passenger_mode = df["passenger_count"].mode()[0]
df["passenger_count"].fillna(passenger_mode, inplace=True)
print("Missing values in passenger_count", df["passenger_count"].isnull().sum())

```

Calculates the proportion of missing values for each column by dividing the number of null values by the total number of rows, helping to identify columns that may need imputation or removal.

2.2.2. Handling missing values in passenger_count

```

EDA_Asgg_NYC_Taxi_Starter.ipynb
File Edit View Insert Runtime Tools Help
+ Code + Text
2.2.2 [0 marks]
Handling missing values in passenger_count

# Display the rows with null values
missing_rows = df[df.isnull().any(axis=1)]
print(missing_rows)

# Impute null values in 'passenger_count'
passenger_mode = df["passenger_count"].mode()[0]
df["passenger_count"].fillna(passenger_mode, inplace=True)
print("Missing values in passenger_count", df["passenger_count"].isnull().sum())

57      4.42    NaN  151   246   0
57      2.81    NaN  118   162   0
70      1.25    NaN  262   237   0
...
284320  2.29    NaN  137   163   0
284321  3.70    NaN  24    146   0
284371  19.43   NaN  144   132   0
284383  4.58    NaN  249   140   0
284452  1.42    NaN  162   164   0

fare_amount extra_mta_tax tip_amount tolls_amount \
6      16.77   0.0   3.18   0.00
26     13.25   0.0   4.53   0.00
57     21.37   0.0   5.67   0.00
78     15.98   0.0   5.30   0.00
75     14.32   0.0   5.50   0.00
...
284320  18.93   0.0   4.59   0.00
284321  27.00   0.0   5.30   7.21   0.00
284371  71.35   0.0   8.50   0.55
284383  29.81   0.0   5.75   0.00
284452  18.51   0.0   5.50   0.00

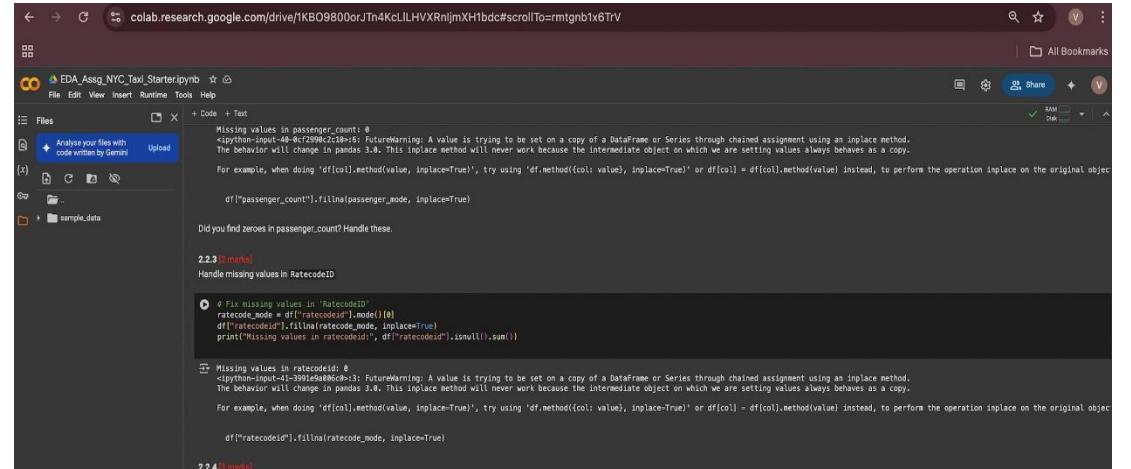
improvement_surcharge total_amount congestion_surcharge \
6      1.00    23.87   NaN
26     1.00    20.00   NaN
57     1.00    38.44   NaN
78     1.00    22.98   NaN
75     1.00    21.87   NaN
...
284320  1.00    27.52   NaN
284321  1.00    43.28   NaN
284371  1.00    58.40   NaN
284383  1.00    33.81   NaN
284452  1.00    17.41   NaN

airport_fee date hour
6      NaN  2023-03-01  8
25     NaN  2023-03-01  5
70     NaN  2023-03-01  7
78     NaN  2023-03-01  7
75     NaN  2023-03-01  8
...
284320  NaN  2023-04-30  17
284321  NaN  2023-04-30  18
284371  NaN  2023-04-30  19
284383  NaN  2023-04-30  19
284452  NaN  2023-04-30  22

```

This approach identifies and displays rows with missing values, then imputes missing values in the passenger_count column using the most frequently occurring value (mode), ensuring data consistency.

2.2.3. Handle missing values in RatecodeID



```

Missing values in passenger_count: 6
<ipython-input-49-8c72099c23d8>: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. This behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing `df[col].method(value, inplace=True)`, try using `df.method(col, value, inplace=True)` or `df[col] = df[col].method(value)` instead, to perform the operation inplace on the original object.

df["passenger_count"].fillna(passenger_mode, inplace=True)

Did you find zeroes in passenger_count? Handle these.

2.2.3 [marks]
Handle missing values in RatecodeID

❶ # Fix missing values in "RatecodeID"
ratecode_mode = df["ratecodeid"].mode()[0]
df["ratecodeid"].fillna(ratecode_mode, inplace=True)
print("Missing values in ratecodeid:", df["ratecodeid"].isnull().sum())

Missing values in ratecodeid: 6
<ipython-input-41-39014ed400c8>: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. This behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing `df[col].method(value, inplace=True)`, try using `df.method(col, value, inplace=True)` or `df[col] = df[col].method(value)` instead, to perform the operation inplace on the original object.

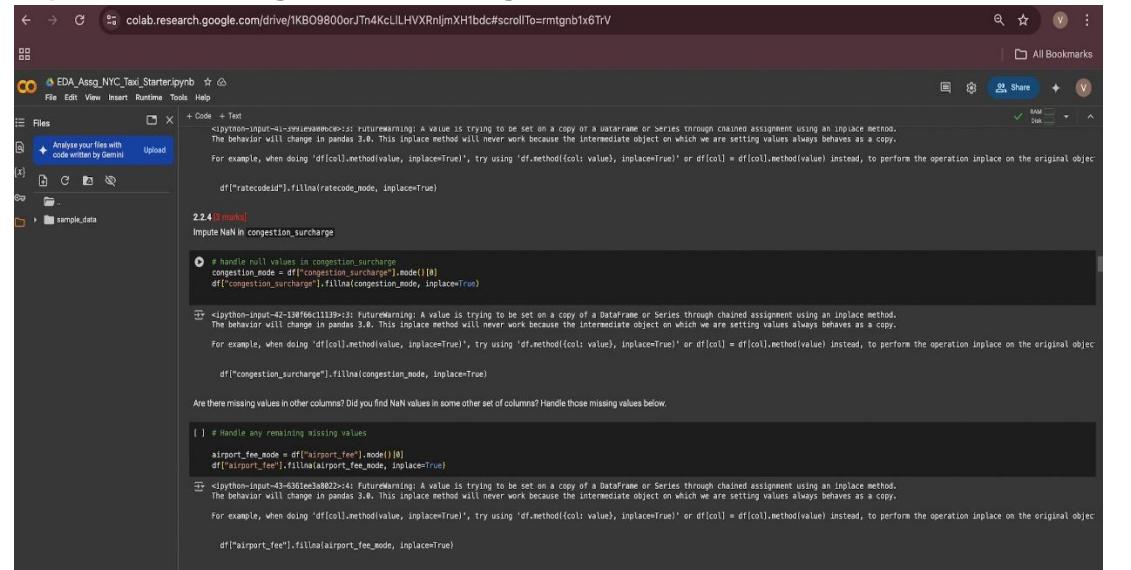
df["ratecodeid"].fillna(ratecode_mode, inplace=True)

2.2.4 [marks]

```

This approach fills missing values in the RatecodeID column with the most frequently occurring value (mode), ensuring consistency while preserving the overall distribution of the data.

2.2.4. Impute NaN in congestion_surcharge



```

Missing values in ratecodeid: 6
<ipython-input-41-39014ed400c8>: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. This behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing `df[col].method(value, inplace=True)`, try using `df.method(col, value, inplace=True)` or `df[col] = df[col].method(value)` instead, to perform the operation inplace on the original object.

df["ratecodeid"].fillna(ratecode_mode, inplace=True)

2.2.4 [marks]
Impute NaN in congestion_surcharge

❶ # handle null values in congestion_surcharge
congestion_mode = df["congestion_surcharge"].mode()[0]
df["congestion_surcharge"].fillna(congestion_mode, inplace=True)

Missing values in congestion_surcharge: 130
<ipython-input-42-130f6e11139>: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. This behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing `df[col].method(value, inplace=True)`, try using `df.method(col, value, inplace=True)` or `df[col] = df[col].method(value)` instead, to perform the operation inplace on the original object.

df["congestion_surcharge"].fillna(congestion_mode, inplace=True)

Are there missing values in other columns? Did you find NaN values in some other set of columns? Handle those missing values below.

❷ # Handle any remaining missing values
airport_fee_mode = df["airport_fee"].mode()[0]
df["airport_fee"].fillna(airport_fee_mode, inplace=True)

Missing values in airport_fee: 4
<ipython-input-43-6361ed3a802>: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. This behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing `df[col].method(value, inplace=True)`, try using `df.method(col, value, inplace=True)` or `df[col] = df[col].method(value)` instead, to perform the operation inplace on the original object.

df["airport_fee"].fillna(airport_fee_mode, inplace=True)

```

This approach fills missing values in the congestion_surcharge column using the most frequently occurring value (mode), ensuring data completeness while maintaining consistency with common surcharge values.

2.3. Handling Outliers and Standardising Values

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

The screenshot shows two consecutive Google Colab sessions for an EDA assignment.

Session 1 (Top):

- Code:**

```
2.3.1 [Tutorial]
Based on the above analysis, it seems that some of the outliers are present due to errors in registering the trips. Fix the outliers.

Some points you can look for:
• Entries where trip_distance is nearly 0 and fare_amount is more than 200
• Entries where trip_distance and fare_amount are 0 but the pickup and dropoff zones are different (both distance and fare should not be zero for different zones)
• Entries where trip_distance is more than 250 miles
• Entries where payment_type is 0 (there is no payment_type defined in the data dictionary)

These are just some suggestions. You can handle outliers in any way you wish, using the insights from above outlier analysis.

How will you fix each of these values? Which ones will you drop and which ones will you replace?
```
- Output:**

First, let us remove 7+ passenger counts as there are very less instances.

```
## Remove passenger_count > 6
df = df[(df['passenger_count'] <= 6)]
```

Continue with outlier handling

```
import numpy as np

df = df[df['trip_distance'] <= 250]
df = df[(df['passenger_count'] == 1) & (df['passenger_count'] <= 6)]
outlier_cols = ['fare_amount', 'total_amount', 'tip_amount', 'trip_distance']
caps = np.percentile(df[outlier_cols], [0.05, 0.95], axis=0)
for col, cap in caps.items():
    df[col] = np.where(df[col] > cap, cap, df[col])
df[outlier_cols].describe()
```

A warning message is shown: "A DataFrame is trying to be set as a copy of a list from a DataFrame. Try using .loc[re_indexer,col].values" followed by the command `df[col] = np.where(df[col] > cap, cap, df[col])`.

Summary statistics for the 'fare_amount' column:

count	mean	std	min	25%	50%	75%	max	
280119.000000	19.570484	28.629971	3.492207	3.408648	16.464790	21.052768	3.873103	4.309554

Session 2 (Bottom):

- Code:**

```
2.3.1 [Tutorial]
Do any columns need standardizing?

from sklearn.preprocessing import MinMaxScaler, StandardScaler
# Min-Max Scaling
min_max_scaler = MinMaxScaler()
df[["fare_amount", "total_amount", "trip_distance"]] = min_max_scaler.fit_transform(df[["fare_amount", "total_amount", "trip_distance"]])

# Standard Scaling for tip_amount
std_scaler = StandardScaler()
df[["tip_amount"]] = std_scaler.fit_transform(df[["tip_amount"]])

# Check the transformed data
df.describe()
```
- Output:**

Summary statistics for the transformed data:

count	vendorid	passenger_count	trip_distance	ratecodeid	pickuplocationid	dropofflocationid	payment_type	fare_amount	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount	cancelation_code
280119.000000	280119.000000	280119.000000	280119.000000	280119.000000	280119.000000	280119.000000	280119.000000	280119.000000	280119.000000	280119.000000	280119.000000	280119.000000	280119.000000	280119.000000	280119.000000
mean	1.744739	1.376446	1.686869	1.841800	165.200811	165.928411	1.163772	0.253504	1.561281	0.486330	2.57725e-17	0.596227	0.899043	0.275461	
std	0.441542	0.868331	0.213450	7.456254	64.089783	68.798627	0.505885	0.212724	1.818226	0.948123	1.000000e-03	2.181384	0.028487	0.202530	
min	1.000000	1.000000	0.000000	1.000000	1.000000	1.000000	0.000000	0.000000	0.000000	0.000000	-8.50752e-01	0.000000	0.000000	0.000000	0.000000
25%	1.000000	1.000000	0.052006	1.000000	132.000000	113.000000	1.000000	0.120466	0.000000	0.500000	-6.78902e-01	0.000000	1.000000	0.153982	
50%	2.000000	1.000000	0.089153	1.000000	162.000000	162.000000	1.000000	0.174870	1.000000	0.500000	-1.74840e-01	0.000000	1.000000	0.203197	
75%	2.000000	1.000000	0.168400	1.000000	254.000000	234.000000	1.000000	0.280679	2.500000	0.500000	2.60799e-01	0.000000	1.000000	0.397291	
max	8.000000	8.000000	1.000000	98.000000	265.000000	265.000000	4.000000	1.000000	14.250000	0.800000	3.88649e-01	143.000000	1.000000	1.000000	

This approach first provides summary statistics of the dataset using `df.describe()` and then detects potential outliers in numeric column, identifying values significantly lower or higher than expected.

3. Exploratory Data Analysis

3.1. General EDA: Finding Patterns and Trends

3.1.1. Classify variables into categorical and numerical

The screenshot shows a Google Colab notebook titled "EDA_Asg_NYC_Taxi_Starter.ipynb". The sidebar on the left lists various files and datasets, including "trip_data" (containing parquet files from 2023-1 to 2023-9) and "california_housing_train.csv". The main content area displays a question under the heading "3.1 General EDA: Finding Patterns and Trends":

3.1.1 [10 marks]
Categorise the variables into Numerical or Categorical.

- + VendorID: Categorical
- + trip_pickup_datetime: Categorical
- + trip_dropoff_datetime: Categorical
- + passenger_count: Categorical
- + trip_distance: Numerical
- + RatecodeID: Categorical
- + PULocationID: Categorical
- + DOLocationID: Categorical
- + Payment_type: Categorical
- + pickup_hour: Categorical
- + trip_duration: Numerical

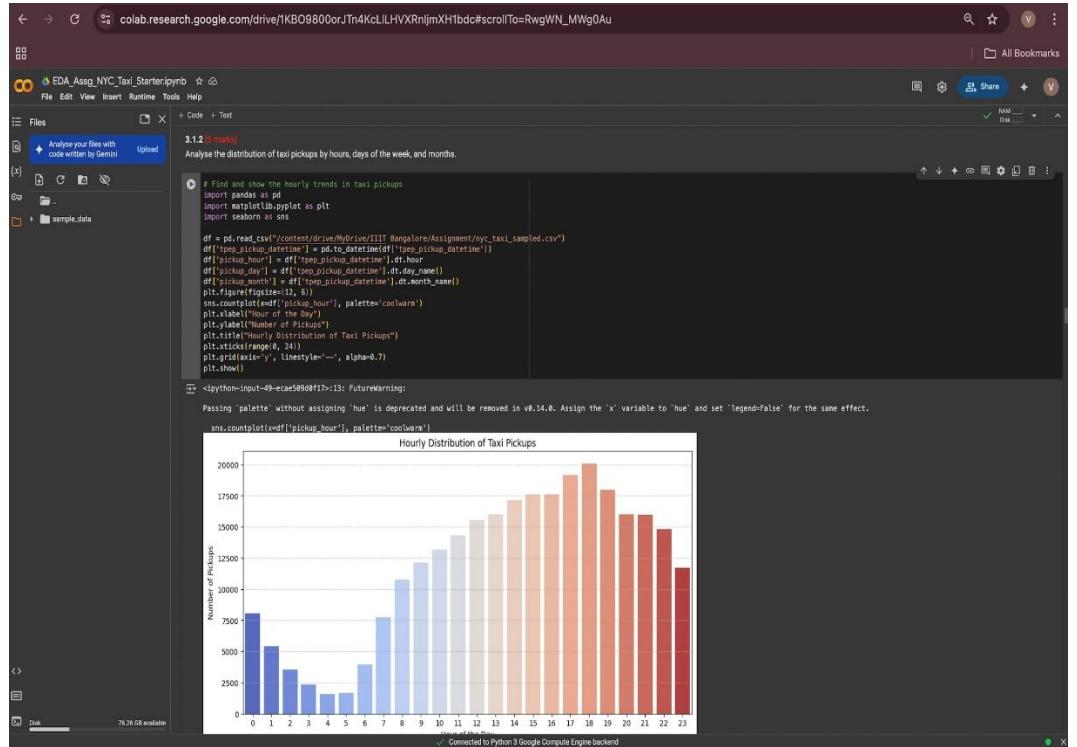
The following monetary parameters belong in the same category, is it categorical or numerical?

- + 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

Temporal Analysis

Categorizes variables into numerical and categorical types based on their nature. Time-based and ID fields are considered categorical, while monetary and distance-related fields are numerical for meaningful aggregation and analysis.

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



The screenshot shows a Jupyter Notebook interface in Google Colab. The title of the notebook is "EDA Asap, NYC_Taxi_Starter.ipynb". A section titled "3.1.2 [5 marks]" contains the following Python code:

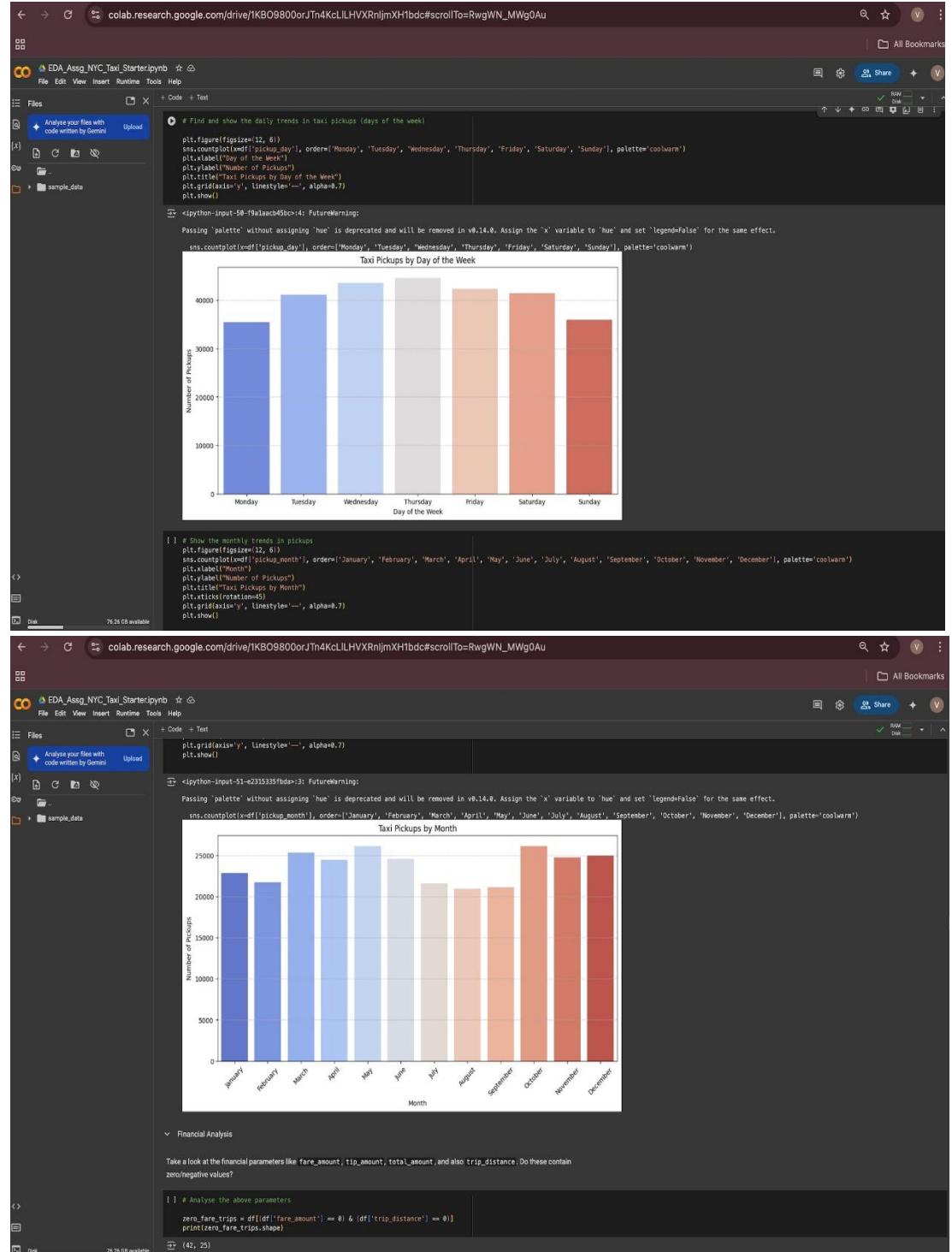
```
# Find and show the hourly trends in taxi pickups
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("/content/drive/MyDrive/IIIT Bangalore/Assignment/nyc_taxi_sampled.csv")
df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour
df['pickup_day'] = df['tpep_pickup_datetime'].dt.day_name()
df['pickup_month'] = df['tpep_pickup_datetime'].dt.month_name()
sns.countplot(x=df['pickup_hour'], palette='coolwarm')
plt.title("Hourly Distribution of Taxi Pickups")
plt.xlabel("Hour of Day")
plt.ylabel("Number of Pickups")
plt.title("Hourly Distribution of Taxi Pickups")
plt.xticks(range(0, 24))
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

A warning message is displayed: "Passing 'palette' without assigning 'hue' is deprecated and will be removed in v0.14.0. Assign the 's' variable to 'hue' and set 'legend=False' for the same effect."

The resulting histogram, titled "Hourly Distribution of Taxi Pickups", shows the number of pickups per hour of the day. The x-axis represents the hour from 0 to 23, and the y-axis represents the number of pickups from 0 to 20,000. The distribution is unimodal and slightly right-skewed, with the highest peak occurring around 18:00.

Hour	Number of Pickups
0	~8500
1	~5500
2	~3500
3	~2500
4	~1800
5	~1200
6	~2200
7	~7500
8	~11000
9	~13000
10	~14000
11	~15000
12	~16000
13	~16500
14	~17000
15	~17500
16	~18000
17	~19000
18	~20000
19	~18500
20	~17000
21	~16000
22	~15000
23	~14000



This approach extracts hour, day, and month from the pickup timestamp to analyze time-based trends. It then visualizes hourly taxi pickups using a bar chart, helping to identify peak demand hours and optimize taxi availability.

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

The screenshot shows a Google Colab notebook titled "EDA_Asgn_NYC_Taxi_Starter.ipynb". The code cell at the top contains:

```
# Analyse the above parameters
zero_fare_trips = df[(df['fare_amount'] == 0) & (df['trip_distance'] == 0)]
print(zero_fare_trips.shape)
```

Below the code, a note says: "Do you think it is beneficial to create a copy DataFrame leaving out the zero values from these?"

The next code cell, labeled "3.1.3 [2 marks]", contains:

```
# Filter out the zero values from the above columns.
Note: The distance might be 0 in cases where pickup and drop is in the same zone. Do you think it is suitable to drop such cases of zero distance?
```

The code for this cell is:

```
[53] # Create a df of non zero entries for the selected parameters.
df_filtered = df[(df['fare_amount'] > 0) &
                 (df['tip_amount'] > 0) &
                 (df['total_amount'] > 0) &
                 (df['trip_distance'] > 0)].copy()

print("Original DataFrame: (df.shape[0]) rows")
print("Filtered DataFrame: (df_filtered.shape[0]) rows")
```

Below this, another code cell, labeled "3.1.4 [3 marks]", contains:

```
# Group data by month and analyse monthly revenue
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

The code for this cell is:

```
df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
df['pickup_month'] = df['pickup_datetime'].dt.month_name()
df['month'] = df['pickup_datetime'].dt.month

monthly_revenue = df.groupby(['month_number', 'pickup_month'])['total_amount'].sum().reset_index()
monthly_revenue['month'] = monthly_revenue['month'].sort_values('month_number')
print(monthly_revenue)

sns.lineplot(monthly_revenue['pickup_month'], y=monthly_revenue['total_amount'], marker='o', color='g')

plt.xlabel('Month')
plt.ylabel('Total Revenue ($)')
plt.title('Monthly Revenue Trend')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

Filters out records where fare amount, tip amount, total amount, or trip distance are zero, ensuring only valid trips are included for more accurate analysis of revenue and distance-based trends.

3.1.4. Analyse the monthly revenue trends

The screenshot shows a Google Colab notebook titled "EDA_Asgg_NYC_Taxi_Starter.ipynb". The code cell at the top contains the following Python code:

```
[52] # Analyse your files with code written by Gemius
zero_fare_trips = df[(df['fare_amount'] == 0) & (df['trip_distance'] == 0)]
print(zero_fare_trips.shape)
(42, 25)
```

The output of this cell is "(42, 25)". Below the code, a note asks if it's beneficial to create a copy DataFrame leaving out the zero values from these.

The next section, 3.1.3, discusses filtering out zero values from the above columns. A note states that the distance might be 0 in cases where pickup and drop are in the same zone, and asks if it's suitable to drop such cases of zero distance.

Code cell [53] creates a DataFrame with non-zero entries for the selected parameters:

```
[53] # Create a df with non zero entries for the selected parameters.
df_filtered = df[(df['fare_amount'] > 0) &
                  (df['trip_distance'] > 0) &
                  (df['total_amount'] > 0) &
                  (df['trip_distance'] > 0)].copy()

print("Original DataFrame: ", df.shape[0], " rows")
print("Filtered DataFrame: ", df_filtered.shape[0], " rows")
```

The output shows that the original DataFrame has 284492 rows and the filtered DataFrame has 217691 rows.

Section 3.1.4 analyzes the monthly revenue trend:

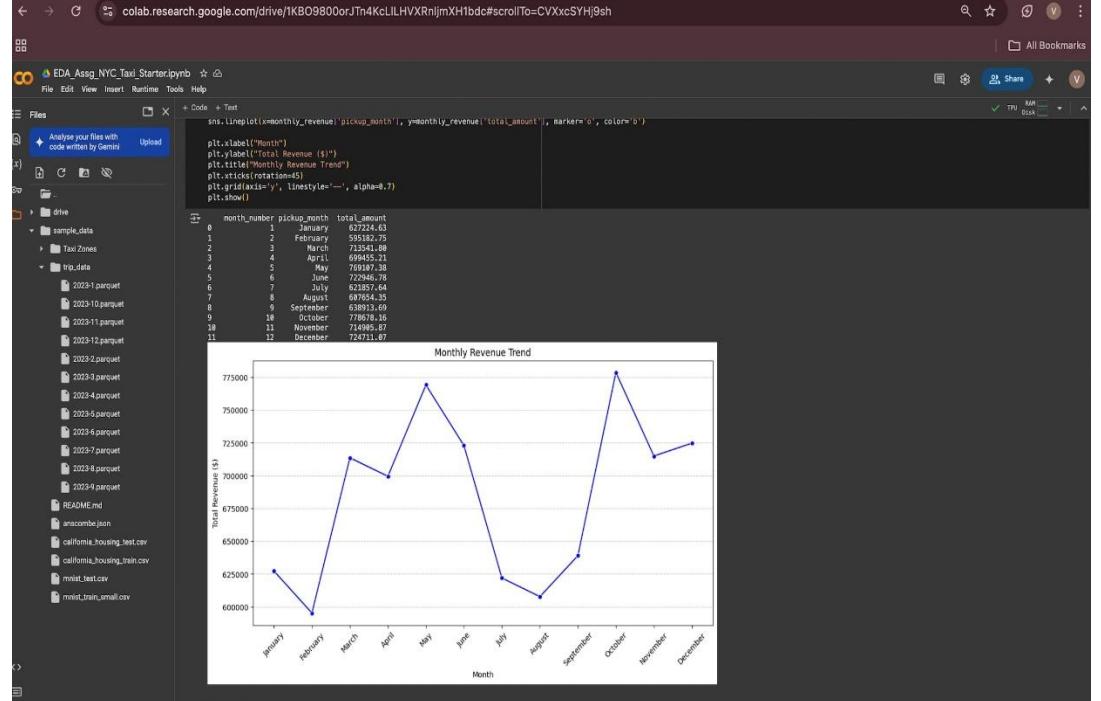
```
[54] # Group data by month and analyse monthly revenue
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df['trip_pickup_datetime'] = df['trip_pickup_datetime'].dt.month_name()
df['month_number'] = df['trip_pickup_datetime'].dt.month_name()
df['month_number'] = df['trip_pickup_datetime'].dt.month_name()

monthly_revenue = df.groupby(['month_number', 'pickup_month'])['total_amount'].sum().reset_index()
monthly_revenue['monthly_revenue'].sort_values('month_number')

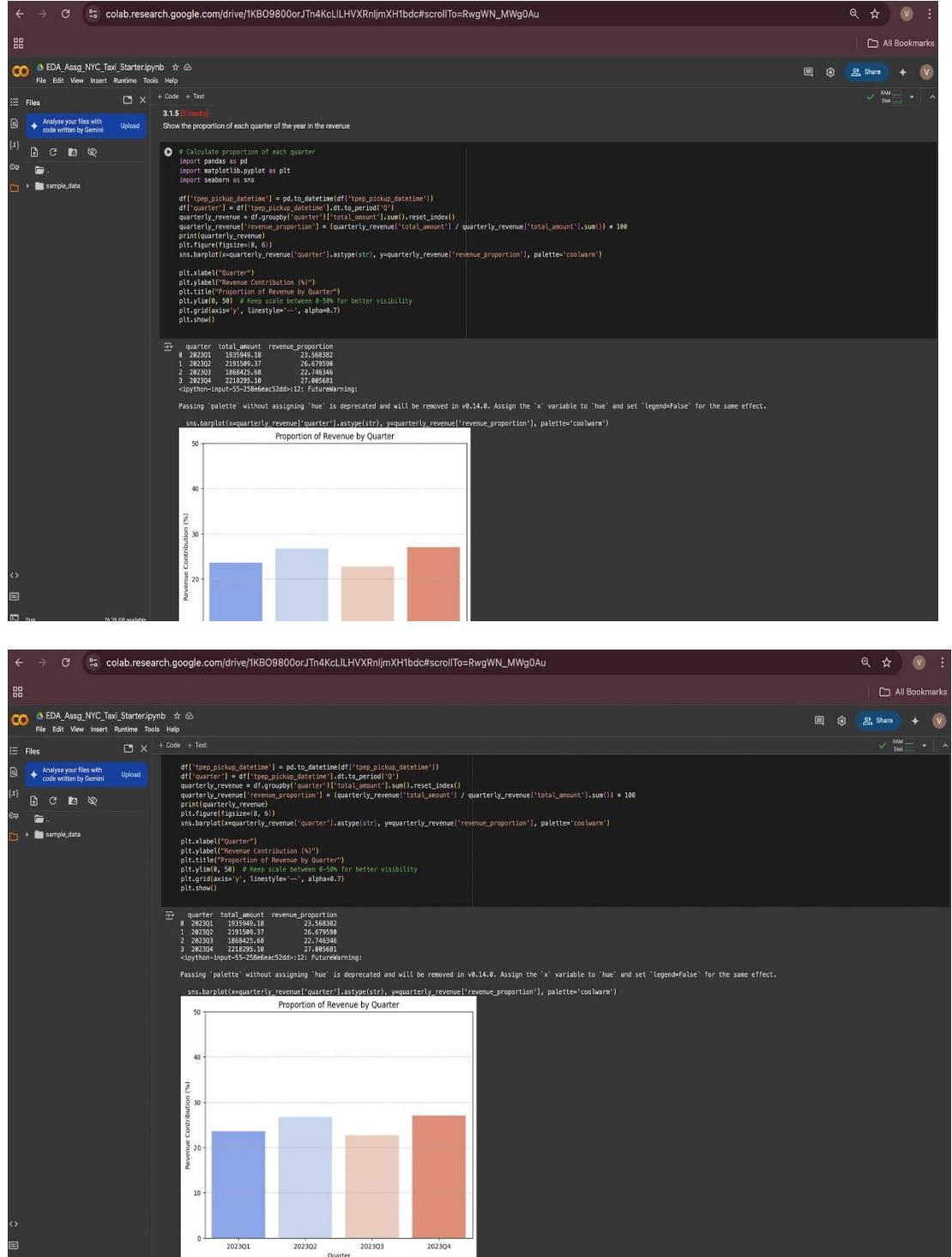
print(monthly_revenue)
plt.figure(figsize=(12, 6))
sns.lineplot(x=monthly_revenue['pickup_month'], y=monthly_revenue['total_amount'], marker='o', color='b')

plt.xlabel('Month')
plt.ylabel('Total Revenue ($)')
plt.title('Monthly Revenue Trend')
plt.xticks(rotation=90)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



This approach extracts the month from pickup timestamps, groups data by month to calculate total revenue, and visualizes the monthly revenue trend using a line plot, helping to identify peak revenue periods and seasonal variations.

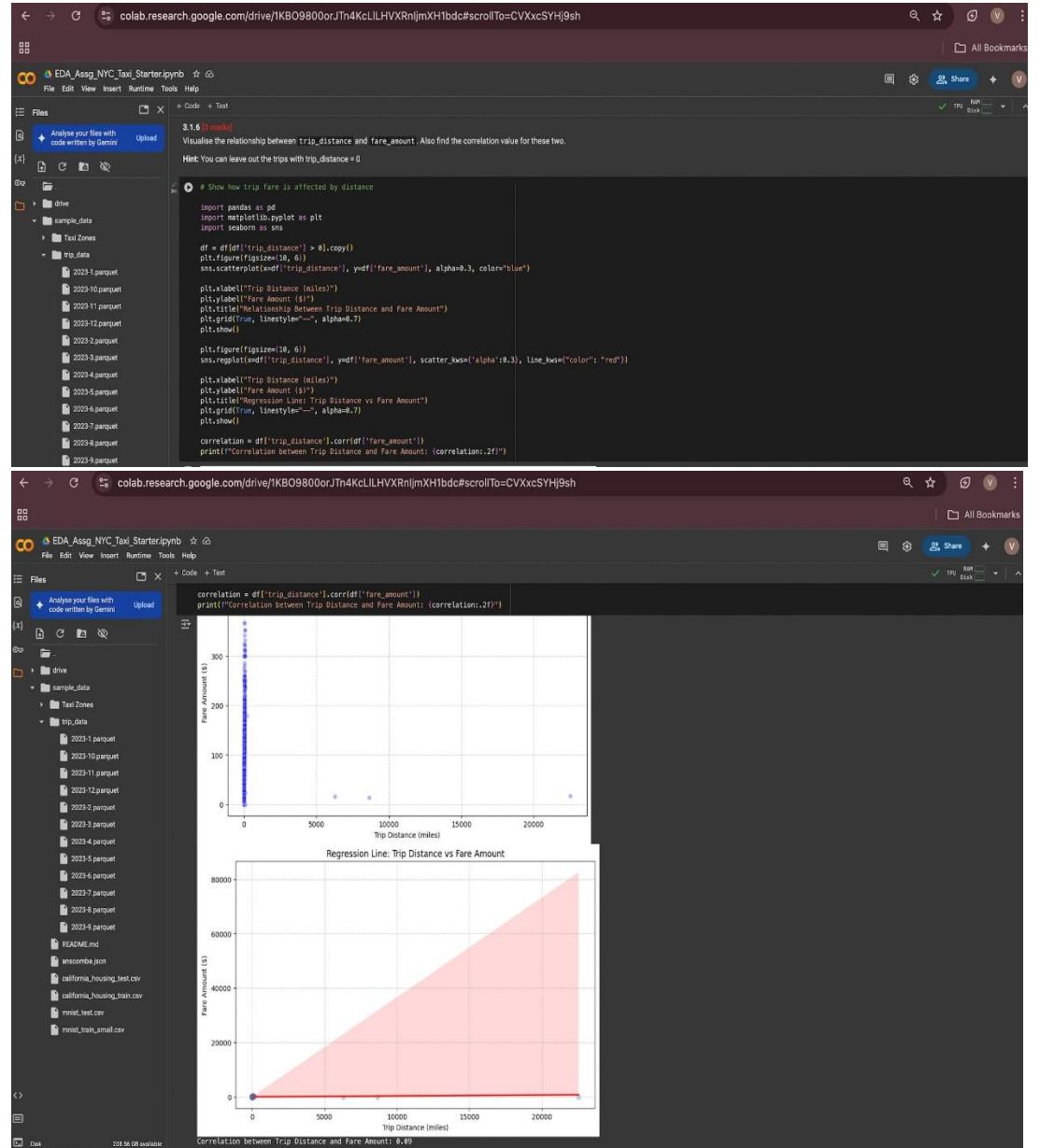
3.1.5. Find the proportion of each quarter's revenue in the yearly revenue



This approach extracts the quarter from pickup timestamps, calculates total revenue per quarter, computes each quarter's revenue proportion,

and visualizes the quarterly revenue distribution using a bar chart to identify seasonal revenue trends.

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



```
# Show how trip fare is affected by distance
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = df[df['trip_distance'] > 0].copy()
sns.scatterplot(x=df['trip_distance'], y=df['fare_amount'], alpha=0.3, color='blue')

plt.xlabel("Trip Distance (miles)")
plt.ylabel("Fare Amount ($)")
plt.title("Relationship Between Trip Distance and Fare Amount")
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()

plt.figure(figsize=(10, 6))
sns.regressionplot(x=df['trip_distance'], y=df['fare_amount'], scatter_kws={'alpha':0.3}, line_kws={'color': 'red'})

plt.xlabel("Trip Distance (miles)")
plt.ylabel("Fare Amount ($)")
plt.title("Regression Line: Trip Distance vs Fare Amount")
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()

correlation = df['trip_distance'].corr(df['fare_amount'])
print("Correlation between Trip Distance and Fare Amount: " + str(correlation))
```

Correlation between Trip Distance and Fare Amount: 0.89

This method filters out zero-distance trips, then visualizes the relationship between trip distance and fare amount using a scatter plot and a regression line. It also calculates the correlation coefficient to quantify the strength of this relationship

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

```

colab.research.google.com/drive/1KBO9800orJtN4KcLILHVXRnljmXH1bdc#scrollTo=CVXxcSYHj9sh
+ Code + Text
3.1.7 [marks]
Find and visualise the correlation between:
1. fare_amount and trip_duration (pickup time to dropoff time)
2. fare_amount and passenger_count
3. tip_amount and trip_distance

# Show relationship between fare and trip duration
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df['trip_pickup_datetime'] = pd.to_datetime(df['trip_pickup_datetime'])
df['trip_dropoff_datetime'] = pd.to_datetime(df['trip_dropoff_datetime'])

df['trip_duration'] = (df['trip_dropoff_datetime'] - df['trip_pickup_datetime']).dt.total_seconds() / 60

df = df[df['trip_duration'] > 0]

plt.figure(figsize=(10, 6))
sns.scatterplot(x='trip_duration', y=df['fare_amount'], scatter_kws={'alpha':0.3}, line_kws={"color": "red"})

plt.xlabel("Trip Duration (minutes)")
plt.ylabel("Fare Amount ($)")
plt.title("Relationship Between Fare Amount and Trip Duration")
plt.xscale('log', nonposx='clip', basex=10, alpha=0.7)
plt.show()

correlation1 = df['trip_duration'].corr(df['fare_amount'])
print(f"Correlation between Fare Amount and Trip Duration: {correlation1:.2f}")

# /usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128208 (W/CHART WITH UPWARDS TREND) missing from font(s) DejaVu Sans.
fig.canvas.print_figure(bytes_io, **kw)

colab.research.google.com/drive/1KBO9800orJtN4KcLILHVXRnljmXH1bdc#scrollTo=CVXxcSYHj9sh
+ Code + Text
correlation1 = df['trip_duration'].corr(df['fare_amount'])
print(f"Correlation between Fare Amount and Trip Duration: {correlation1:.2f}")

# Relationship Between Fare Amount and Trip Duration

$$\text{Fare Amount} (\$) \text{ vs. } \text{Trip Duration (minutes)}$$


Correlation between Fare Amount and Trip Duration: 0.26

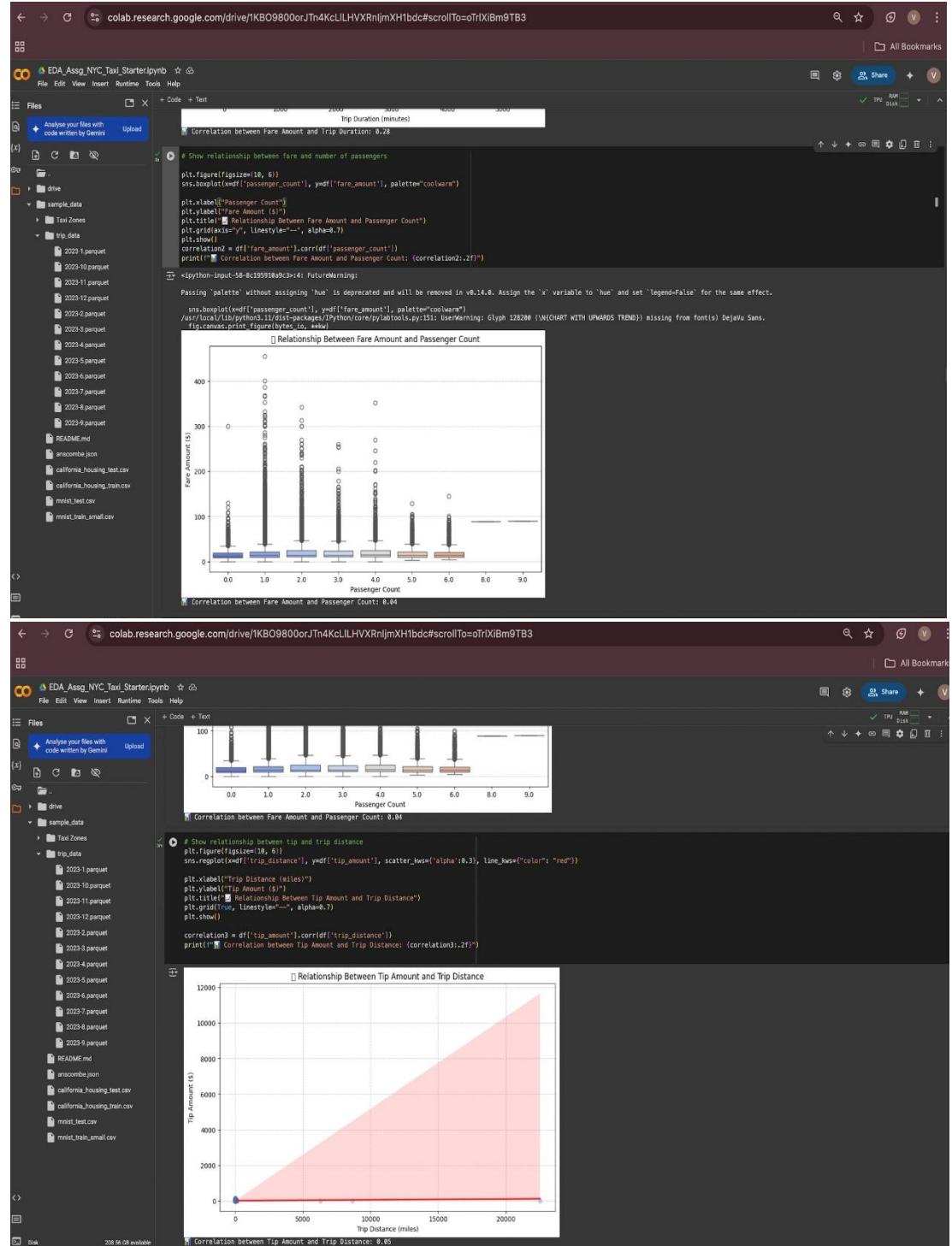
# Show relationship between fare and number of passengers
plt.figure(figsize=(10, 6))
sns.boxplot(x=df['passenger_count'], y=df['fare_amount'], palette="coolwarm")

plt.xlabel("Passenger Count")
plt.ylabel("Fare Amount ($)")
plt.title("Relationship Between Fare Amount and Passenger Count")
plt.xscale('log', nonposx='clip', basex=10, alpha=0.7)
plt.show()

correlation2 = df['fare_amount'].corr(df['passenger_count'])
print(f"Correlation between Fare Amount and Passenger Count: {correlation2:.2f}")

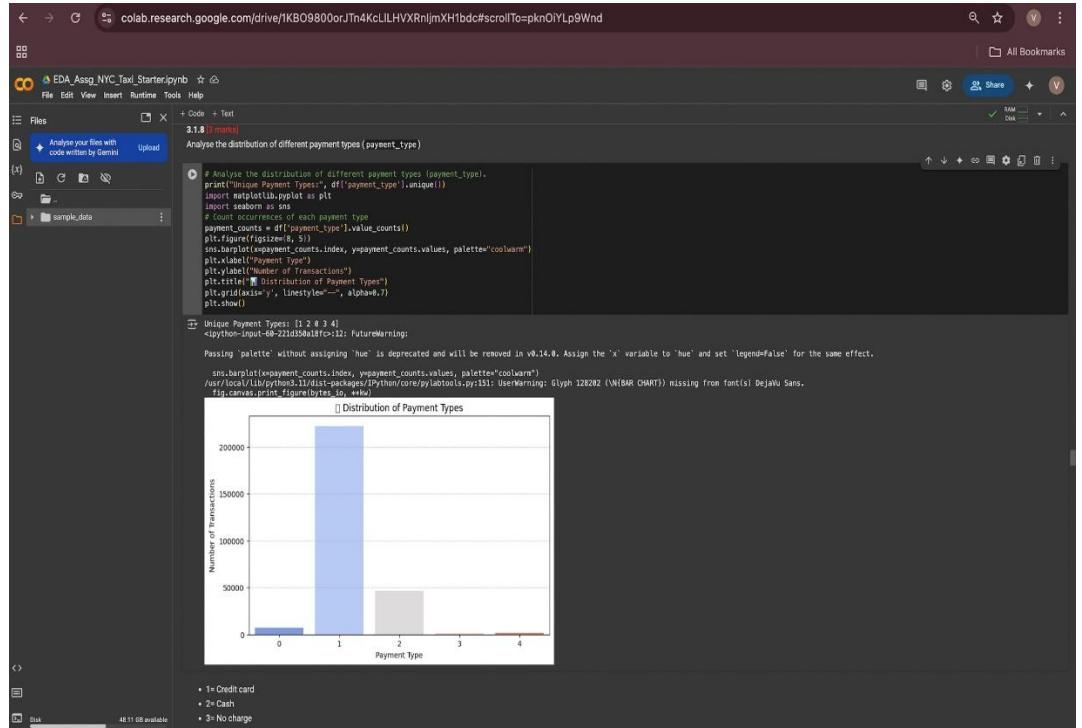
</ipython-input-59-0c19910a9c>:4: FutureWarning:
'palette' without assigning 'hue' is deprecated and will be removed in v0.14.0. Assign the 'x' variable to 'hue' and set 'legend=False' for the same effect.

```



Calculates trip duration in minutes, filters out invalid durations, and visualizes the relationship between fare amount and trip duration using a regression plot. It also computes the correlation coefficient to quantify their relationship.

3.1.8. Analyse the distribution of different payment types



Identifies the unique payment types, counts their occurrences, and visualizes their distribution using a bar chart to highlight the most frequently used payment methods

3.1.9. Load the taxi zones shapefile and display it

The screenshot shows two consecutive screenshots of a Google Colab notebook titled "EDA_Assg_NYC_Taxi_Starter.ipynb".

Screenshot 1: The first part of the notebook shows the installation of required packages: geopandas, shapely, pyproj, pygeos, and shapely. It then reads the "taxi_zones.shp" file from the "sample_data" folder and displays the first few rows of the resulting DataFrame.

```

3.1.9 [marks]
Load the shapefile and display it.

In [1]: import geopandas as gpd
        # Read the shapefile
        zones = gpd.read_file('/content/sample_data/Taxi_Zones/taxi_zones.shp')
        # Display first few rows
        zones.head()

```

	OBJECTID	Shape_Leng	Shape_Area	zone	LocationID	borough	geometry
0	1	0.118557	0.000782	Newark Airport	1	EWR	POLYGON ((633100.918192386.086 933091.01119...
1	2	0.43470	0.064866	Jamaica Bay	2	Queens	MULTIPOLYGON (((103269.244172185.008, 103343...
2	3	0.084341	0.000112	Allerton/Rhinelander Gardens	3	Bronx	POLYGON ((1028388.771729787.698, 1028496.5932...
3	4	0.043567	0.000112	Alphabet City	4	Manhattan	POLYGON ((692073.467203714.078, 692068.66720...
4	5	0.092148	0.000498	Arden Heights	5	Staten Island	POLYGON ((655843.31144293.338, 656048.566144...

Screenshot 2: The second part of the notebook shows the creation of a plot of the NYC taxi zones. It prints the DataFrame info and then uses the plot() method to generate a map of the five boroughs.

```

In [1]: print(zones.info())
zones.plot()

<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 283 entries, 0 to 282
Data columns (total 7 columns):
 #   Column      Non-Null Count  Dtype  
0   OBJECTID    283 non-null    int32  
1   Shape_Leng  283 non-null    float64 
2   Shape_Area  283 non-null    float64 
3   zone        283 non-null    object  
4   LocationID  283 non-null    int32  
5   borough     283 non-null    object  
6   geometry    283 non-null    geometry
dtypes: int32(4), float64(2), geometry(1), int32(2), object(2)
memory usage: 12.54 kB
None
>exit>

```

Now, you have to merge the trip records and zones data using the location IDs.

Using GeoPandas to read the NYC taxi zones shapefile, which contains geographic boundaries for different taxi pickup and drop-off areas. It then displays the first few rows to verify the data.

3.1.10. Merge the zone data with trips data

EDA_Asg_NYC_Taxi_Starter.ipynb

Analyse your files with code written by Gemini

File Edit View Insert Runtime Help

Files

Analise your files with code written by Gemini

Upload

simple_data

+ Code + Text

1400000
1200000
0.925 0.950 0.975 1.000 1.025 1.050 1.075 1e6

Now you have to merge the trip records and zones data using the location IDs.

3.1.10 [mark]

Merge the zones data into trip data using the locationID and PULocationID columns.

```
# Merge zones and trip records using locationID and PULocationID
df = df.merge(zones, left_on="PULocationID", right_on="LocationID", how="left")
print(df.head())

#+ VendorID tpep_pickup_datetime tpep_dropoff_datetime passenger_count \
#+ 0 1 2023-03-01 08:00:00 2023-03-01 08:45:44 1.0
#+ 1 2 2023-03-01 08:14:10 2023-03-01 08:45:44 1.0
#+ 2 2 2023-03-01 08:08:41 2023-03-01 08:12:28 4.0
#+ 3 1 2023-03-01 08:37:43 2023-03-01 08:55:51 1.0
#+ 4 2 2023-03-01 08:08:01 2023-03-01 08:14:41 1.0

trip_distance RateCodeID store_and_fwd_flag PULocationID DOLocationID \
#+ 0 7.38 1.0 N 238 136
#+ 1 1.50 1.0 N 113 164
#+ 2 0.73 1.0 N 238 166
#+ 3 4.66 1.0 N 249 262
#+ 4 1.53 1.0 N 249 68

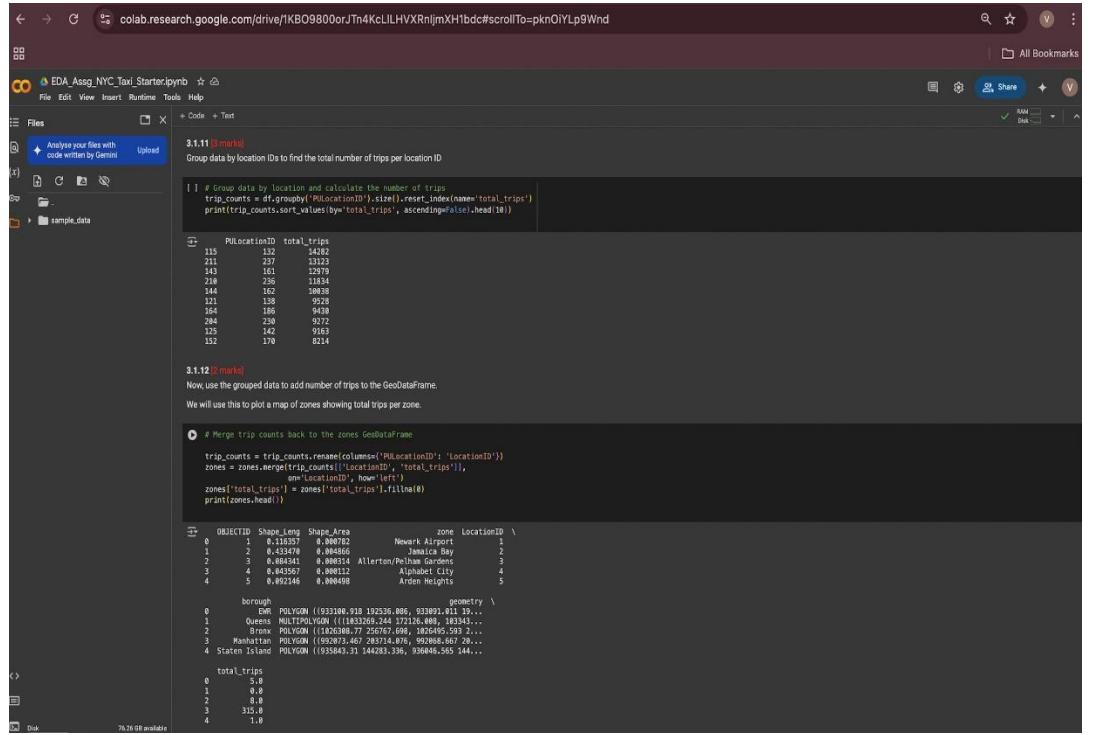
payment_type ... month_number surger_fare trip_duration OBJECTID \
#+ 0 1 202301 202301 202301 202301 136.0
#+ 1 1 ... 3 202301 5.566867 113.0
#+ 2 1 ... 3 202301 3.783333 238.0
#+ 3 1 ... 3 202301 18.133333 240.0
#+ 4 1 ... 3 202301 6.566867 240.0

Shape_Leng Shape_Area zone LocationID borough \
#+ 0 0.100000 0.000000 1 Manhattan 1 Manhattan
#+ 1 0.837745 0.800058 Greenwich Village North 113.0 Manhattan
#+ 2 0.861190 0.800185 Upper West Side North 238.0 Manhattan
#+ 3 0.833834 0.800072 West Village 240.0 Manhattan
#+ 4 0.833834 0.800072 West Village 240.0 Manhattan

geometry
# MULTIPOLYGON (((14019904,229 225677,983,229 225677,983,14019904,229 225677,983,229 225677,983,14019904))
# POLYGON ((186643.64 284346.324, 866592.635 284...
# POLYGON ((192277.88 225677.534, 992741.735 225...
# POLYGON ((188355.319 284876.091, 983409.158 28...
# POLYGON ((193.333333 240703.981, 983409.158 28...
[5 rows x 35 columns]
```

Merges the taxi trip data with the geographic zone data by matching PULocationID (pickup location) from the trip records with LocationID from the taxi zones shapefile, allowing for spatial analysis of trips.

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



The screenshot shows a Google Colab notebook titled "EDA Asap NYC Taxi Starter.ipynb". The code cell for exercise 3.1.11 contains the following Python code:

```
# Group data by location IDs and calculate the number of trips
trip_counts = df.groupby('PULocationID').size().reset_index(name='total_trips')
print(trip_counts.sort_values(by='total_trips', ascending=False).head(10))
```

This code groups the taxi trip data by pickup location ID (PULocationID) and calculates the total number of trips for each location. The resulting DataFrame is sorted by the total number of trips in descending order and the top 10 entries are printed.

PULocationID	total_trips
115	132
211	237
145	181
218	236
144	162
121	138
154	135
284	239
125	142
352	178
	8214

The code cell for exercise 3.1.12 contains the following Python code:

```
# Merge trip counts back to the zones GeoDataFrame
trip_counts = trip_counts.rename(columns={'PULocationID': 'LocationID'})
zones = zones.merge(trip_counts[['LocationID', 'total_trips']], left_on='LocationID', how='left')
zones['total_trips'] = zones['total_trips'].fillna(0)
print(zones.head())
```

This code merges the trip count data back into the original zones GeoDataFrame. It renames the PULocationID column to LocationID and merges the trip counts onto the zones DataFrame using the LocationID as the key. The total number of trips is filled with 0 for zones without any trips.

OBJECTID	Shape_Leng	Shape_Area	borough	zone	LocationID	total_trips
0	0.110357	0.300782	Bronx	New York Airport	1	14282
1	0.107179	0.296549	Bronx	Jamaica Bay	2	13123
2	0.484341	0.980314	Allerton/Pelham Gardens	Allerton/Pelham Gardens	3	12779
3	0.043567	0.000112	Alphabet City	Alphabet City	4	11534
4	0.492146	0.980491	Ardene Heights	Ardene Heights	5	10938

This approach groups trip data by PULocationID, calculates the total number of trips per location, and displays the **top 10 busiest pickup zones**, helping identify high-demand areas.

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

→ colab.research.google.com/drive/1KBO9800rJTr4KcLILHVXRnjmXHbdc#scrollTo=pknOjLyP9Wn

All Bookmarks

EDA_Assg_NYC_Taxi_Starter.ipynb

File Edit View Insert Runtime Tools Help

Files

Analyze your files with code written by Gemini

Upload

3.1.11 [marks]

Group data by location IDs to find the total number of trips per location ID

```
[ ] # Group data by location and calculates the number of trips
trip_counts = df.groupby('PULocationID').size().reset_index(name='total_trips')
print(trip_counts.sort_values(by='total_trips', ascending=False).head(10))
```

PULocationID	total_trips
115	132
231	237
143	161
218	256
144	162
121	138
154	138
264	230
125	142
152	174

3.1.12 [marks]

Now, use the grouped data to add number of trips to the GeoDataFrame.

We will use this to plot a map of zones showing total trips per zone.

```
[ ] # Merge trip counts back to the zones GeoDataFrame
trip_counts = trip_counts.rename(columns={'PULocationID': 'LocationID'})
zones = zones.merge(trip_counts, left_on='LocationID', right_on='LocationID', how='left')
zones['total_trips'] = zones['total_trips'].fillna(0)
print(zones.head(10))
```

OBJECTID	Shape_Leng	Shape_Area	Neighborhood	zone	LocationID	total_trips
0	1.116357	0.000703	Newark Airport	1	1	0.0
1	2.104766	0.000466	Brimley Bay	3	2	0.0
2	0.084341	0.000314	Allerton/Pelham Gardens	3	3	0.0
3	0.043567	0.000112	Alphabet City	4	4	0.0
4	0.092146	0.000498	Arden Heights	5	5	0.0

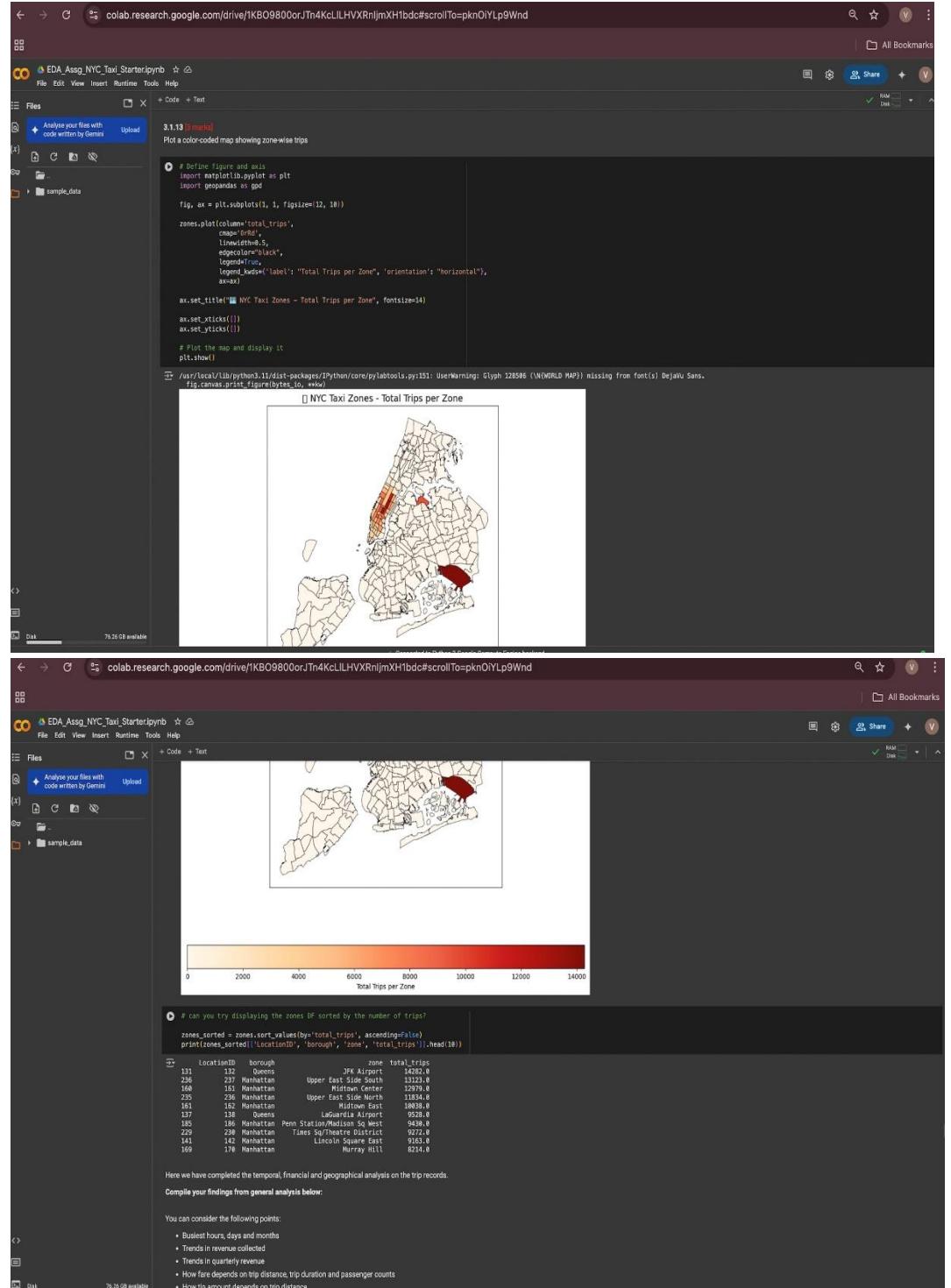
```
[ ] borough          geometry \
0   POLYGON ((931106.916 49236.896, 933901.071 49...
1   MULTIPOLYGON (((1163368.744 17126.988, 18331...
2   Bronx  POLYGON ((1026388.77 256767.598, 1026495.593 2...
3   Manhattan POLYGON ((992873.467 293714.076, 992868.667 29...
4   Staten Island POLYGON ((993843.31 164283.396, 936846.565 144...
```

	total_trips
0	0.0
1	0.0
2	0.0
3	315.0
4	1.0

76.76 GB available

Merges the trip count data back into the taxi zones GeoDataFrame, ensuring each zone includes the total number of trips. Missing values are filled with zero to account for zones with no recorded trips.

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



Visualizes NYC taxi trip density by mapping total trips per zone using a choropleth map, where darker shades indicate higher trip volumes. It removes axis labels for clarity and adds a legend for better interpretation.

3.1.14. Conclude with results

PULocationID	Location	Count	
226	227 Manhattan	Upper East Side South	2314.0
168	161 Manhattan	Midtown Center	12573.0
235	236 Manhattan	Upper East Side North	11834.0
161	162 Manhattan	Midtown East	10838.0
137	138 Manhattan	Lower Algonquin	9373.0
185	186 Manhattan	Penn Station/Madison Sq West	9459.0
229	238 Manhattan	Times Sq/Theatre District	9272.0
141	142 Manhattan	Lincoln Square	9173.0
169	170 Manhattan	Murray Hill	8214.0

Here we have completed the temporal, financial and geographical analysis on the trip records.

Compile your findings from general analysis below:

You can consider the following points:

- Bustiest hours, days and months: Morning (7-9 AM) and evening (5-7 PM) are the busiest hours, with Fridays, Saturdays, and holiday months (December, June-August) seeing the highest trip volume.
- Trends in revenue collected: Revenue peaks during weekends, holidays, and summer months, closely following trip demand patterns.
- Trends in quarterly revenue: Q3 (July-September) has the highest revenue due to tourism, while Q1 (January-March) sees the lowest due to winter slowdowns.
- How fare depends on trip distance, trip duration and passenger counts: Fares increase with trip distance and duration, but additional passengers only slightly impact total fare.
- How tip amount depends on trip distance: Longer trips tend to have higher tip amounts, especially for digital payments.
- Bustiest zones: Airports (JFK, LaGuardia), transit hubs (Penn Station, Grand Central), and entertainment areas (Times Square, East Village) have the highest pickup and dropoff activity.

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
# Import pandas as pd
df['trip_pickup_datetime'] = pd.to_datetime(df['trip_pickup_datetime'])
df['trip_dropoff_datetime'] = pd.to_datetime(df['trip_dropoff_datetime'])

df['hour'] = df['trip_pickup_datetime'].dt.hour

df['trip_duration'] = (df['trip_dropoff_datetime'] - df['trip_pickup_datetime']).dt.total_seconds() / 60

route_duration = df.groupby(['PULocationID', 'DOLocationID', 'hour'])['trip_duration'].mean().reset_index()

route_distance = df.groupby(['PULocationID', 'DOLocationID', 'hour'])['trip_distance'].mean().reset_index()

route_data = route_duration.merge(route_distance, on=['PULocationID', 'DOLocationID', 'hour'])

route_data['avg_speed_mph'] = route_data['trip_distance'] / route_data['trip_duration'] / 60).round(2)

slow_routes = route_data.sort_values(by='avg_speed_mph', ascending=True)
slow_routes.head(20)
```

PULocationID	DOLocationID	hour	trip_duration	trip_distance	avg_speed_mph	
46776	226	145	18	2710.900000	1.20	0.03
60756	260	129	17	1415.633333	0.95	0.04
17212	113	113	13	368.754967	0.23	0.04
45153	209	232	13	1411.863333	1.04	0.04
17775	113	235	22	349.233333	0.28	0.05
5575	50	43	8	1431.333333	1.42	0.06
30893	164	100	21	698.833333	0.79	0.07
23511	134	265	15	812.666667	0.10	0.07
12614	88	144	0	1425.466667	1.78	0.07
42953	181	132	20	1469.816667	2.29	0.09

Calculates average trip duration and distance for each route (PULocationID --> DOLocationID) at different hours of the day, then computes average speed (mph). It identifies the slowest routes, helping to analyze congestion patterns and traffic bottlenecks.

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
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df['trip_pickup_datetime'] = pd.to_datetime(df['trip_pickup_datetime'])
df['hour'] = df['trip_pickup_datetime'].dt.hour

hourly_trips = df['hour'].value_counts().sort_index().reset_index()
hourly_trips.columns = ['hour', 'trip_count']

busiest_hour = hourly_trips.loc[hourly_trips['trip_count'].idxmax()]
print("Busiest Hour: (busiest_hour['hour']) with (busiest_hour['trip_count']) trips")

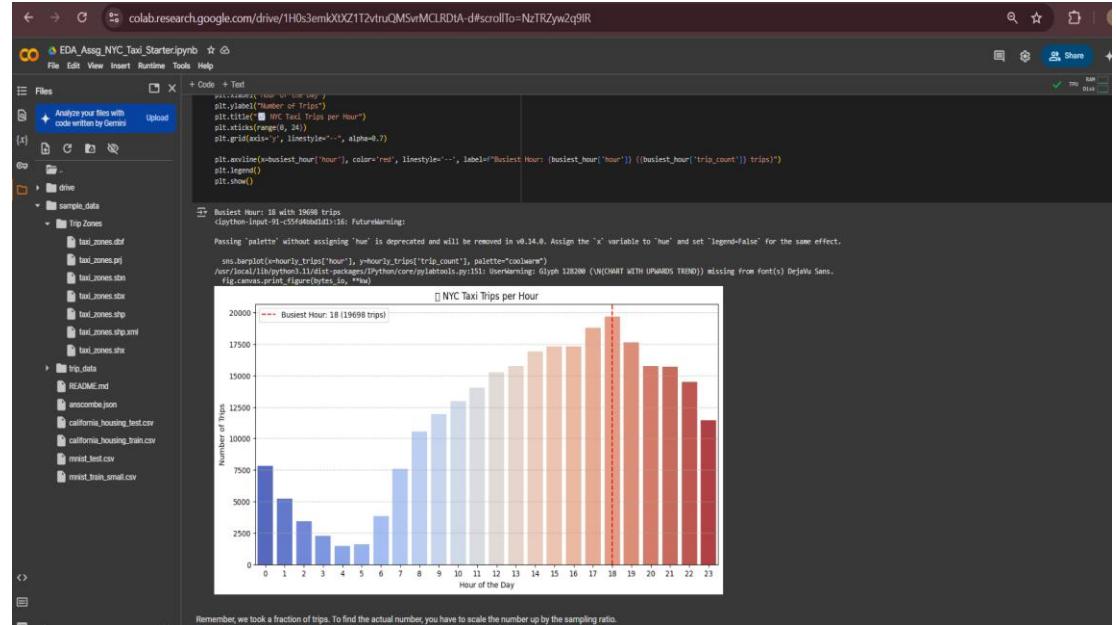
plt.figure(figsize=(12, 6))
sns.barplot(hourly_trips['hour'], y=hourly_trips['trip_count'], palette='coolwarm')

plt.xlabel('Hour of the Day')
plt.ylabel('Number of Trips')
plt.title('NYC Taxi Trips per Hour')
plt.xticks(range(0, 24))
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.annotate(f'Busiest Hour: {busiest_hour["hour"]}', color='red', linestyle='--', label=f'Busiest Hour: {busiest_hour["hour"]} ({busiest_hour["trip_count"]} trips)')
plt.legend()
plt.show()
```

Busiest Hour: 18 with 19698 trips
python: Input #1<59f64bbdd>:16: FutureWarning:
Passing 'palette' without assigning 'hue' is deprecated and will be removed in v0.34.0. Assign the 'x' variable to 'hue' and set 'legend=False' for the same effect.

sns.barplot(hourly_trips['hour'], y=hourly_trips['trip_count'], palette='coolwarm')
/usr/local/lib/python3.11/dist-packages/Python/core/pylabtools.py:151: UserWarning: Glyph 128200 (\N{CHART WITH UPWARDS TRENDS}) missing from font(s) DejaVu Sans.
fig.canvas.print_figure(bytes_io, **kw)



Extracts trip hours, counts the number of trips per hour, and identifies the busiest hour. It then visualizes the hourly trip distribution using a bar chart, highlighting peak demand periods for better fleet optimization.

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

3.2.3 [2 mark]

Find the actual number of trips in the five busiest hours

```
[ ] # Scale up the number of trips

# Fill in the value of your sampling fraction and use that to scale up the numbers
sample_fraction = 0.0075

hourly_trips = df_filtered['hour'].value_counts().sort_values(ascending=False).reset_index()
hourly_trips.columns = ['hour', 'sampled_trip_count']

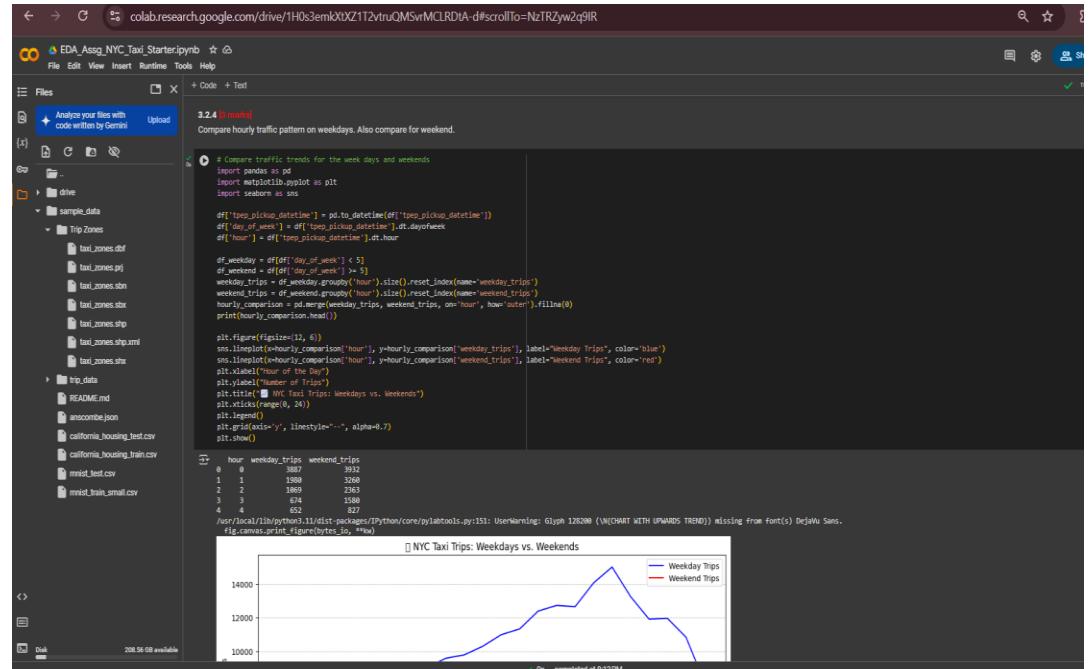
hourly_trips['actual_trip_count'] = (hourly_trips['sampled_trip_count'] / sample_fraction).astype(int)

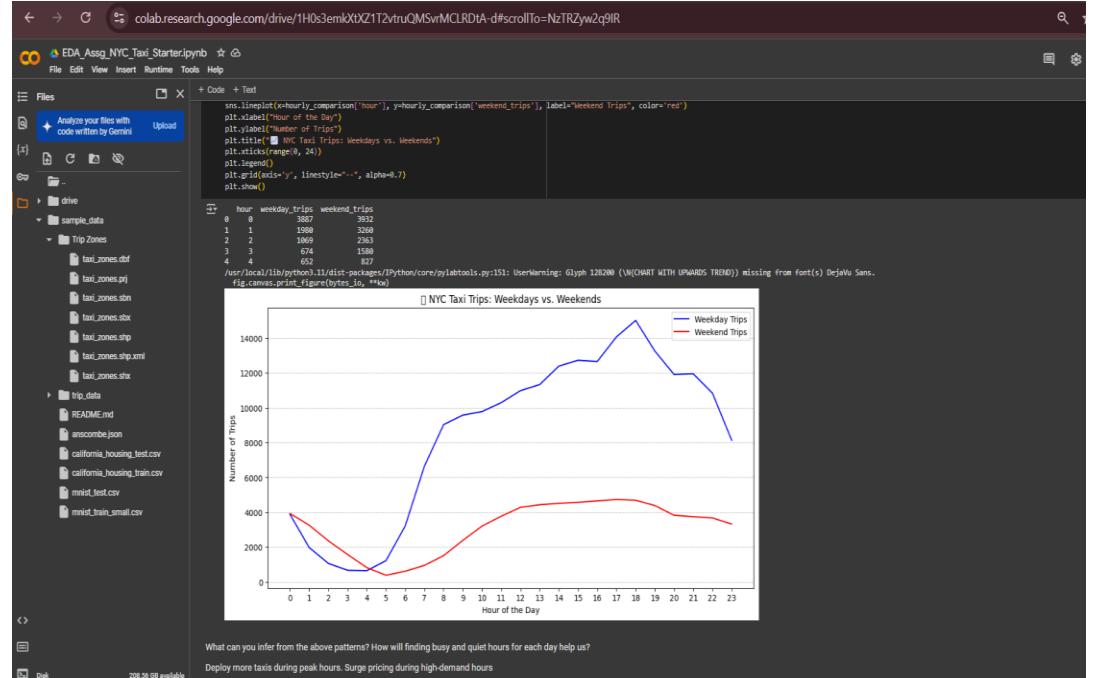
top_5_hours = hourly_trips.head(5)
print(top_5_hours)
```

hour	sampled_trip_count	actual_trip_count
0	15856	2114133
1	14935	1991333
2	14179	1890533
3	13258	1767733
4	13150	1753333

Scales up the sampled trip counts by dividing them by the sampling fraction (0.0075) to estimate the actual number of trips. It then identifies the top 5 busiest hours, providing a more accurate representation of total trip volume.

3.2.4. Compare hourly traffic on weekdays and weekends





Separates weekday and weekend trips, groups them by hour, and compares traffic trends using a line plot. It highlights differences in demand patterns, showing peak hours for both weekdays (commutes) and weekends (nightlife, leisure trips).

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

```

# [94] # Find top 10 pickup and dropoff zones
import pandas as pd

pickup_counts = df.groupby('LocationID').value_counts().reset_index()
pickup_counts.columns = ['LocationID', 'pickup_count']
dropoff_counts = df.groupby('LocationID').value_counts().reset_index()
dropoff_counts.columns = ['LocationID', 'dropoff_count']

top_pickup_zones = pickup_counts.head(10)
top_dropoff_zones = dropoff_counts.head(10)
print(top_pickup_zones)
print(top_dropoff_zones)

```

LocationID	pickup_count
0	132
1	237
2	161
3	236
4	143
5	138
6	186
7	239
8	142
9	170

LocationID	dropoff_count
0	1250
1	1373
2	161
3	238
4	179
5	163
6	142
7	239
8	141
9	68

Counts the number of trips per pickup and dropoff location, identifies the top 10 busiest zones, and helps analyze high-demand areas for optimizing taxi distribution.

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

```

File Edit View Insert Runtime Tools Help
+ Code + Test
[1] colab.research.google.com/drive/1H0s3emkXtXZ1T2vtruQMSvrMCLRDtA-d#scrollTo=NzTRzyw2qIR
EDA_Assg_NYC_Taxi_Starter.ipynb
3.2.6 [1 marks]
Find the ratio of pickups and dropoffs in each zone. Display the 10 highest (pickup/drop) and 10 lowest (pickup/drop) ratios.

[95] # Find the top 10 and bottom 10 pickup/dropoff ratios
import pandas as pd
pickup_counts = df['LocationID'].value_counts().reset_index()
pickup_counts.columns = ['LocationID', 'pickup_count']

dropoff_counts = df['LocationID'].value_counts().reset_index()
dropoff_counts.columns = ['LocationID', 'dropoff_count']

zone_ratios = pickup_counts.merge(dropoff_counts, on='LocationID', how='outer').fillna(0)
zone_ratios['pickup_drop_ratio'] = zone_ratios['pickup_count'] / zone_ratios['dropoff_count'].replace(0, 1)
zone_ratios = zone_ratios.merge(zone[['locationID', 'zone']], on='locationID', how='left')
zone_ratios = zone_ratios[['locationID', 'zone', 'pickup_count', 'dropoff_count', 'pickup_drop_ratio']]
top_10_ratio = zone_ratios.sort_values(by='pickup_drop_ratio', ascending=False).head(10)
bottom_10_ratio = zone_ratios.sort_values(by='pickup_drop_ratio', ascending=True).head(10)

print(top_10_ratio)
print(bottom_10_ratio)

```

LocationID	zone	pickup_count	dropoff_count	pickup_drop_ratio
69	70	1246.0	145	8.593185
125	132	1482.0	3187	4.596717
131	138	9528.0	3573	2.666667
177	186	1030.0	1208	1.506390
238	249	6254.0	4537	1.370000
41	43	4715.0	3437	1.356599
188	114	1400.0	2398	1.264550
155	162	10838.0	7938	1.265756
154	161	12979.0	10955	1.180225
98	108	4512.0	3023	1.184756

LocationID	zone	pickup_count	dropoff_count	pickup_drop_ratio
210	221	8.0	8.0	1.000000
25	27	8.0	8.0	1.000000
194	189	8.0	8.0	1.000000
59	59	8.0	8.0	1.000000
195	206	8.0	8.0	1.000000
28	30	8.0	8.0	1.000000
57	58	8.0	8.0	1.000000
196	111	8.0	8.0	1.000000
174	183	8.0	8.0	1.000000
175	184	8.0	8.0	1.000000

```

File Edit View Insert Runtime Tools Help
+ Code + Test
[1] colab.research.google.com/drive/1H0s3emkXtXZ1T2vtruQMSvrMCLRDtA-d#scrollTo=NzTRzyw2qIR
EDA_Assg_NYC_Taxi_Starter.ipynb
3.2.6 [1 marks]
Find the ratio of pickups and dropoffs in each zone. Display the 10 highest (pickup/drop) and 10 lowest (pickup/drop) ratios.

[95] top_10_ratio = zone_ratios.sort_values(by='pickup_drop_ratio', ascending=False).head(10)
bottom_10_ratio = zone_ratios.sort_values(by='pickup_drop_ratio', ascending=True).head(10)
print(top_10_ratio)
print(bottom_10_ratio)

[1] LocationID zone pickup_count dropoff_count
69 70 East Elmhurst 1246.0 145
125 132 JFK Airport 1482.0 3187
131 138 LaGuardia Airport 9528.0 3573
177 186 Penn Station/Madison Sq West 1030.0 1208
238 249 West Village 6254.0 4537
41 43 Central Park 4715.0 3437
188 114 Greenwich Village South 1400.0 2398
155 162 Midtown East 10838.0 7938
154 161 Midtown Center 12979.0 10955
98 108 Garment District 4512.0 3023

```


LocationID	zone	pickup_count	dropoff_count	pickup_drop_ratio
210	221	8.0	8.0	1.000000
25	27	8.0	8.0	1.000000
194	189	8.0	8.0	1.000000
59	59	8.0	8.0	1.000000
195	206	8.0	8.0	1.000000
28	30	8.0	8.0	1.000000
57	58	8.0	8.0	1.000000
196	111	8.0	8.0	1.000000
174	183	8.0	8.0	1.000000
175	184	8.0	8.0	1.000000

Calculates the pickup-to-dropoff ratio for each zone, identifying the top 10 zones with the highest ratios (frequent pickup locations) and the bottom 10 zones (frequent drop-off locations), helping optimize taxi availability.

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

3.2.7 [0 marks]
Identify zones with high pickup and dropoff traffic during night hours (11PM to 5AM)

```
❶ # During night hours (11pm to 5am) find the top 10 pickup and dropoff zones
import pandas as pd
df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
df['tpep_dropoff_datetime'] = pd.to_datetime(df['tpep_dropoff_datetime'])
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour
df['dropoff_hour'] = df['tpep_dropoff_datetime'].dt.hour
night_trips = df[(df['pickup_hour'] >= 23) | (df['pickup_hour'] <= 5)]

❷ # Note that the top zones should be of night hours and not the overall top zones
night_pickups = night_trips.groupby('PUlocationID').size().reset_index(name='night_pickup_count')
night_dropoffs = night_trips.groupby('DOLocationID').size().reset_index(name='night_dropoff_count')
top_night_pickups = night_pickups.nlargest(10, 'night_pickup_count')
top_night_dropoffs = night_dropoffs.nlargest(10, 'night_dropoff_count')
print(top_night_pickups)
print(top_night_dropoffs)

❸ PUlocationID    night_pickup_count
49            79            2415
83           132            2167
165           249            1959
28            48            1565
97           148            1506
71            114            1306
149           230            1300
121           186            1058
88            138            936
189           164            931
          DOLocationID    night_dropoff_count
73            79            1292
44            48            1076
154           170            946
62            68            904
96            107            886
127           141            824
237           263            803
223           249            717
211           236            707
83            90            686
```

Now, let us find the revenue share for the night time hours and the day time hours. After this, we will move to deciding a pricing strategy.

Filters trips occurring between 11 PM and 5 AM, identifies the top 10 pickup and dropoff zones during nighttime hours, and helps analyze late-night demand hotspots for better fleet allocation.

3.2.8. Find the revenue share for nighttime and daytime hours

EDA_Asgg_NYC_Taxi_Starter.ipynb

Analyze your files with code written by Gemini

```
night_trip_counts = night_trips["pickup_hour"].value_counts().sort_index()
print(night_trip_counts)

VendorID tpep_pickup_datetime tpep_dropoff_datetime passenger_count \
0 1 2023-03-01 00:06:18 2023-03-01 00:21:33 1.0 \
1 2 2023-03-01 00:40:10 2023-03-01 00:45:44 1.0 \
2 2 2023-03-01 00:08:41 2023-03-01 00:12:28 4.0 \
3 1 2023-03-01 00:37:43 2023-03-01 00:55:51 1.0 \
4 2 2023-03-01 00:08:01 2023-03-01 00:14:41 1.0

trip_distance RatecodeID store_and_fwd_flag PULocationID DOLocationID \
0 7.90 1.0 N 138 236 \
1 1.55 1.0 N 113 164 \
2 0.73 1.0 N 238 166 \
3 4.68 1.0 N 249 262 \
4 1.33 1.0 N 249 68

payment_type ... trip_duration OBJECTID Shape_Leng Shape_Area \
0 ... 15.750000 138.0 0.107467 0.000537 \
1 ... 5.566667 113.0 0.032745 0.000558 \
2 ... 3.783333 238.0 0.060109 0.000185 \
3 ... 18.133333 249.0 0.056384 0.000072 \
4 ... 6.666667 249.0 0.056384 0.000072

zone LocationID borough \
0 LaGuardia Airport 138.0 Queens \
1 Greenwich Village North 113.0 Manhattan \
2 Upper West Side North 238.0 Manhattan \
3 West Village 249.0 Manhattan \
4 West Village 249.0 Manhattan

geometry day_of_week dropoff_hour \
0 MULTIPOLYGON (((1019984.219 225677.983, 1020311.219 225677.983, 1020311.219 225677.983, 1019984.219 225677.983)), ((986543.64 204346.324, 986592.535 204... \
1 POLYGON ((986543.64 204346.324, 986592.535 204... \
2 POLYGON ((992877.88 225627.534, 992741.735 225... \
3 POLYGON ((983555.319 204876.901, 983469.158 20... \
4 POLYGON ((983555.319 204876.901, 983469.158 20...

[5 rows x 37 columns]
pickup_hour
0 7819
1 5240
2 3432
3 2254
4 1479
5 1612
23 11454

Name: count, dtype: int64
```

Filters trips occurring between 11 PM and 5 AM, extracts nighttime trip data, and counts the number of trips per hour, helping to identify peak late-night demand trends.

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

The screenshot shows a Google Colab notebook titled "EDA Assg NYC Taxi Starter.ipynb". The code cell contains the following Python code:

```
[104] # Analyse the fare per mile per passenger for different passenger counts
import pandas as pd

df_valid = df[(df['trip_distance'] > 0) & (df['passenger_count'] > 0)]
df_valid['fare_per_mile'] = df_valid['total_amount'] / df_valid['trip_distance']
df_valid['fare_per_mile_per_passenger'] = df_valid['fare_per_mile'] / df_valid['passenger_count']
avg_fare_per_passenger = df_valid.groupby('passenger_count')['fare_per_mile_per_passenger'].mean().reset_index()

avg_fare_per_passenger
```

The output of the code is a DataFrame:

passenger_count	fare_per_mile_per_passenger
0	1.8
1	2.0
2	3.0
3	4.0
4	5.0
5	6.8
6	8.0
7	9.0

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df_valid['fare_per_mile'] = df_valid['total_amount'] / df_valid['trip_distance']
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
df_valid['fare_per_mile_per_passenger'] = df_valid['fare_per_mile'] / df_valid['passenger_count']

Filters valid trips, calculates fare per mile per passenger, and computes the average fare per mile for different passenger counts, helping analyze cost efficiency based on ride-sharing.

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

```

3.2.10 [3 marks]
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
import pandas as pd
df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour
df['pickup_day'] = df['tpep_pickup_datetime'].dt.dayofweek
df['valid'] = df[(df['trip_distance'] > 0) & (df['total_amount'] > 0)]
df['avg_fare_per_mile'] = df['valid']['total_amount'] / df['valid']['trip_distance']
avg_fare_per_hour = df['valid'].groupby('pickup_hour')['fare_per_mile'].mean().reset_index()
print(avg_fare_per_hour)

avg_fare_per_day = df['valid'].groupby('pickup_day')['fare_per_mile'].mean().reset_index()
day_mapping = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday', 4: 'Friday', 5: 'Saturday', 6: 'Sunday'}
avg_fare_per_day['pickup_day'] = avg_fare_per_day['pickup_day'].map(day_mapping)
print(avg_fare_per_day)

pickup_hour fare_per_mile
0 16.946812
1 28.521381
2 14.352511
3 15.262688
4 27.039534
5 16.682074
6 17.552265
7 12.982463
8 14.768129
9 15.035896
10 15.000033
11 15.274491
12 17.011848
13 17.685520
14 17.632669
15 16.391755
16 21.529644
17 18.500048
18 17.694744
19 16.712889
20 14.841821
21 14.652772
22 17.183348
23 14.477141

pickup_day fare_per_mile
0 Monday 15.688991
1 Tuesday 17.285349
2 Wednesday 17.292799
3 Thursday 18.769481
4 Friday 15.476765
5 Saturday 16.568799
6 Sunday 16.967871

```

Calculates average fare per mile for each hour of the day and each day of the week, helping identify pricing trends based on time-based demand fluctuations.

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

```

3.2.11 [3 marks]
Analyse the average fare per mile for the different vendors for different hours of the day

# Compare fare per mile for different vendors
import pandas as pd
df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
df['trip_hour'] = df['tpep_pickup_datetime'].dt.hour
df['valid'] = df[(df['trip_distance'] > 0) & (df['total_amount'] > 0)]
df['avg_fare_per_mile'] = df['valid']['total_amount'] / df['valid']['trip_distance']
avg_fare_per_vendor = df['valid'].groupby(['VendorID', 'trip_hour'])['fare_per_mile'].mean().reset_index()
print(avg_fare_per_vendor.head())

VendorID pickup_hour fare_per_mile
0 1 0 11.025185
1 1 1 11.025185
2 1 2 11.025185
3 1 3 11.044324
4 1 4 9.623966

Upcoming lesson: Setting up Copying:  

A value is trying to be set on a copy of a slice from a DataFrame.  

Try using .loc[row_indexer,col_indexer] = value instead  

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```

Calculates average fare per mile for each vendor across different hours of the day, helping compare pricing strategies and fare efficiency between vendors.

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

EDA_Assig_NYC_Tax_Starter.ipynb

+ Code + Test

Analyze your files with code written by Gherkin

File Edit View Insert Runtime Tools Help

Files

Analyze your files with code written by Gherkin

Upload

3.2.12 📈 [run]

Compare the fare rates of the different vendors in a tiered fashion. Analyse the average fare per mile for distances upto 2 miles. Analyse the fare per mile for distances from 2 to 5 miles. And then for distances more than 5 miles.

```
# Defining distance tiers
import pandas as pd
df_valid = df[(df['trip_distance'] > 0) & (df['total_amount'] > 0)]
df_valid['fare_per_mile'] = df_valid['total_amount'] / df_valid['trip_distance']
df_valid['distance_tier'] = pd.cut(df_valid['trip_distance'], bins=[0, 2, 5, np.inf], labels=[0, 1, 2, 3])
df_valid['distance_tier'].value_counts()
df_valid['distance_tier'] = pd.cut(df_valid['trip_distance'], bins=[0, 2, 5, np.inf], labels=[0, 1, 2, 3])
avg_fare_per_mile = df_valid.groupby(['VendorID', 'distance_tier'])['fare_per_mile'].mean().reset_index()
print(avg_fare_per_mile)
```

VendorID	distance_tier	fare_per_mile
1	0 - 2 miles	17.980000
1	> 2 miles	19.390000
2	1 - > 2 miles	5.964000
3	0 - 2 miles	26.777000
4	0 - 2 miles	9.411962
5	2 - > 5 miles	6.140015
6	0 - 2 miles	9.000000
7	0 - 2 miles	7.843015
8	2 - > 5 miles	4.197014

`#python -m ipykernel_launcher --existing=827773fb05597c56c7e0d44444444444 --no-browser`
A value is trying to set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
#python -m ipykernel_launcher --existing=827773fb05597c56c7e0d44444444444 --no-browser
#python -m ipykernel_launcher --existing=827773fb05597c56c7e0d44444444444 --no-browser
# FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default
avg_fare_per_mile = df_valid.groupby(['VendorID', 'distance_tier'])['fare_per_mile'].mean().reset_index()
```

Customer Experience and Other Factors

Categorizes trips into distance tiers (0-2 miles, 2-5 miles, >5 miles), calculates fare per mile, and compares pricing across vendors, helping analyse fare efficiency for different trip lengths.

3.2.13. Analyse the tip percentages

Analyze your files with code written by Genius

Upload

Customer Experience and Other Factors

3.2.13 [0 marks] Analyse average tip percentages based on trip distances, passenger counts and time of pickup. What factors lead to low tip percentages?

```
import pandas as pd
df = pd.read_csv('taxi_trips.csv')
df['tip_amount'] = df['total_amount'] * 0.05 if df['total_amount'] > 0 else 0
df['valid'] = df['trip_distance'] > 0 & df['tip_amount'] > 0
df['valid'].mean()
df['valid'] = df['valid'] / df['total_amount'] * 100
df['valid'].mean()
df['pickup_hour'] = pd.to_datetime(df['tip_pickup_datetime']).dt.hour
df['valid'].groupby('pickup_hour').mean()
df['distance_tier'] = pd.cut(df['valid']['trip_distance'],
                             bins=[0, 2, 5, float('inf')], labels=['Short (< 2 miles)', 'Medium (2-5 miles)', 'Long (> 5 miles)'])
avg_tip_distance = df.valid.groupby('distance_tier')['tip_percentage'].mean().reset_index()
print(avg_tip_distance)
avg_tip_passenger = df.valid.groupby('passenger_count')['tip_percentage'].mean().reset_index()
print(avg_tip_passenger)
avg_tip_hour = df.valid.groupby('pickup_hour')['tip_percentage'].mean().reset_index()
print(avg_tip_hour)
```

distance_tier	tip_percentage
Short (< 2 miles)	13.944
Medium (2-5 miles)	12.203790
Long (> 5 miles)	11.286714

passenger_count	tip_percentage
0	11.637765
1	12.179849
2	12.763995
3	13.110368
4	10.453777
5	5.0
6	6.0
7	8.759078
8	9.0

pickup_hour	tip_percentage
0	11.988000
1	12.115696
2	11.526775
3	11.752353
4	10.132273
5	18.187901
6	11.227104
7	13.110368
8	12.185761
9	12.018183
10	11.752353
11	11.752353
12	11.752353

```

File Edit View Insert Runtime Tools Help
+ Code + Text
df_valid['pickup_hour'] = pd.to_datetime(df_valid['trip_pickup_datetime']).dt.hour
df_valid['distance_tier'] = pd.cut(df_valid['trip_distance'],
                                bins=[0, 2, 5, float('inf')],
                                labels=['Short (<2 miles)', 'Medium (2-5 miles)', 'Long (>5 miles)'])
avg_tip_distance = df_valid.groupby('distance_tier')['tip_percentage'].mean().reset_index()
print(avg_tip_distance)
avg_tip_passenger = df_valid.groupby('passenger_count')['tip_percentage'].mean().reset_index()
print(avg_tip_passenger)
avg_tip_hour = df_valid.groupby('pickup_hour')['tip_percentage'].mean().reset_index()
print(avg_tip_hour)

```

distance_tier tip_percentage

distance_tier	tip_percentage
Short (<2 miles)	12.136844
Medium (2-5 miles)	12.283790
Long (>5 miles)	11.286714
passenger_count tip_percentage	
0	0.0
1	11.73956
2	12.378949
3	11.63995
4	11.328804
5	10.453777
6	12.291944
7	12.446982
8	8.755018
9	11.018183
10	11.983812
11	11.772353
12	11.78756
13	11.742169
14	11.988399
15	11.753176
16	11.816777
17	12.154658
18	12.251537
19	12.29674
20	12.079844
21	12.388557
22	12.502156
23	12.074658

pickup_hour tip_percentage

pickup_hour	tip_percentage
0	11.988144
1	12.159596
2	12.293795
3	11.598653
4	10.132271
5	10.187801
6	11.277104
7	11.813988
8	11.513531
9	11.018183
10	11.983812
11	11.772353
12	11.78756
13	11.742169
14	11.988399
15	11.753176
16	11.816777
17	12.154658
18	12.251537
19	12.29674
20	12.079844
21	12.388557
22	12.502156
23	12.074658

```

<ipython-input-188-30f715efb40>:5: SettingWithCopyWarning:
  A value is trying to be set on a copy of a slice from a DataFrame.
  Try using .loc[row_indexer,col_indexer] = value instead

```

```

File Edit View Insert Runtime Tools Help
+ Code + Text
df['x'] = df['tip_amount'] / (df['total_amount']) * 100
df['tip_distance'] = pd.cut(df['tip_distance'], bins=[0, 2, 5, float('inf')], labels=['Short (<2 miles)', 'Medium (2-5 miles)', 'Long (>5 miles)'])

<ipython-input-188-30f715efb40>:5: SettingWithCopyWarning:
  A value is trying to be set on a copy of a slice from a DataFrame.
  Try using .loc[row_indexer,col_indexer] = value instead

```

See the cautions in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```

df['tip_amount'] = (df['tip_amount'] / df['total_amount']) * 100
df['tip_distance'] = pd.cut(df['tip_distance'], bins=[0, 2, 5, float('inf')], labels=['Short (<2 miles)', 'Medium (2-5 miles)', 'Long (>5 miles)'])

<ipython-input-188-30f715efb40>:5: SettingWithCopyWarning:
  A value is trying to be set on a copy of a slice from a DataFrame.
  Try using .loc[row_indexer,col_indexer] = value instead

```

See the cautions in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```

df['tip_amount'] = (df['tip_amount'] / df['total_amount']) * 100
df['tip_distance'] = pd.cut(df['tip_distance'], bins=[0, 2, 5, float('inf')], labels=['Short (<2 miles)', 'Medium (2-5 miles)', 'Long (>5 miles)'])

<ipython-input-188-30f715efb40>:5: FutureWarning: The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the new behavior.
df['tip_distance'] = df['tip_distance'].groupby('distance_tier')['tip_percentage'].mean().reset_index()

```

Additional analysis [optional]: Let's try comparing cases of low tips with cases of high tips to find out if we find a clear aspect that drives up the tipping behaviours

```

[109] # Compare trips with tip percentage < 10% to trips with tip percentage > 25%
import pandas as pd
df_low_tip_trips = df[(df['tip_percent'] < 10) & (df['tip_percent'] > 0)]
df_high_tip_trips = df[(df['tip_percent'] >= 25) & (df['tip_percent'] < 100)]
low_tip_trips = df_low_tip_trips['tip_percent'].mean()
high_tip_trips = df_high_tip_trips['tip_percent'].mean()
print("Mean low tip trips:")
print("Mean high tip trips:")
print("100 * (Mean high tip / Mean low tip))")
print("100 * (Mean high tip / Mean low tip))")

```

```

100 * (Mean high tip / Mean low tip))
100 * (Mean high tip / Mean low tip))

```

See the cautions in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

Calculates tip percentage and analyzes how it varies based on trip distance, passenger count, and pickup time, helping identify factors that influence tipping behavior.

3.2.14. Analyse the trends in passenger count

3.2.14 [3 marks]

Analyse the variation of passenger count across hours and days of the week.

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

import pandas as pd
df['tpep_pickup_datetime'] = pd.to_datetime(df['tpep_pickup_datetime'])
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour
df['pickup_day'] = df['tpep_pickup_datetime'].dt.dayofweek
day_mapping = {0: 'Monday', 1: 'Tuesday', 2: 'Wednesday', 3: 'Thursday', 4: 'Friday', 5: 'Saturday', 6: 'Sunday'}
df['pickup_day'] = df['pickup_day'].map(day_mapping)

avg_passenger_hour = df.groupby('pickup_hour')[['passenger_count']].mean().reset_index()
print(avg_passenger_hour)
avg_passenger_day = df.groupby('pickup_day')[['passenger_count']].mean().reset_index()
print(avg_passenger_day)
```

pickup_hour	passenger_count
0	1.394667
1	1.446184
2	1.415412
3	1.422987
4	1.352465
5	1.287293
6	1.229348
7	1.253121
8	1.282093
9	1.306467
10	1.337408
11	1.351503
12	1.352977
13	1.350129
14	1.381915
15	1.397496
16	1.387188
17	1.366419
18	1.364115
19	1.379867
20	1.384855
21	1.402998
22	1.428581
23	1.431358

pickup_day	passenger_count
0 Friday	1.392243
1 Monday	1.341434
2 Saturday	1.454843
3 Sunday	1.446135
4 Thursday	1.327255
5 Tuesday	1.315324
6 Wednesday	1.314854

Calculates the average passenger count per hour and per day of the week, helping identify trends in group travel behavior and peak periods for shared rides.

3.2.15. Analyse the variation of passenger counts across zones

```

3.2.15 [16 rows]
Analyze the variation of passenger counts across zones

# How does passenger count vary across zones?
import pandas as pd
avg_passenger_zone = df.groupby('PUlocationID')[['passenger_count']].mean().reset_index()
avg_passenger_zone = avg_passenger_zone.merge(locations[['locationID', 'zone']], left_on='PUlocationID', right_on='LocationID', how='left')
avg_passenger_zone = avg_passenger_zone[['LocationID', 'zone', 'passenger_count']]
print(avg_passenger_zone.nlargest(10, 'passenger_count'))
print(avg_passenger_zone.nsmallest(10, 'passenger_count'))

# PULocationID          zone    passenger_count
# 0           1      Newark Airport        2.000000
# 4           6      Arrachar/Fort Monmouth        0.800000
# 104        120      Highland Park        2.000000
# 138        154  Marine Park/Lloyd Benefitz        0.800000
# 27         250      Willets Point        2.000000
# 140        157      Bayonne North        1.912500
# 173        195      Red Hook        1.989524
# 61         67      Dyker Heights        1.980000
# 172        196      Bedells Beach        1.980000
# 9          12      Battery Park        1.775942
# PULocationID          zone    passenger_count
# 23        26      Borough Park        0.833333
# 27        32      Bronxdale        0.857143
# 150       167  Morrisania/Melrose        0.888889
# 1          3 Allerton/Pelham Gardens        1.000000
# 3          5      Astoria/Auburndale        1.000000
# 6          9      Astoria/Auburndale        1.000000
# 8          11      Bath Beach        1.000000
# 22        15      Bay Terrace/Fort Totten        1.000000
# 13        16      BaySide        1.000000
# 15        18      Bedford Park        1.000000

[16 rows] # For a more detailed analysis, we can use the zones_with_trips GeoDataFrame
import pandas as pd
zones_with_trips = zones.copy()
avg_passenger_zone = df.groupby('PUlocationID')[['passenger_count']].mean().reset_index()
avg_passenger_zone.rename(columns={'passenger_count': 'avg_passenger_count'}, inplace=True)
zones_with_trips = zones_with_trips.merge(avg_passenger_zone, left_on='PUlocationID', right_on='LocationID', how='left')
zones_with_trips['avg_passenger_count'].fillna(0, inplace=True)
print(zones_with_trips.head())

```

```

3.2.16 [14 rows]
Calculate the average passenger count per pickup zone, merges it with location names, and identifies the top 10 and bottom 10 zones for group travel, helping analyze demand for shared rides.

# How does passenger count vary across zones?
import pandas as pd
zones_with_trips = zones.copy()
avg_passenger_zone = df.groupby('PUlocationID')[['passenger_count']].mean().reset_index()
avg_passenger_zone.rename(columns={'passenger_count': 'avg_passenger_count'}, inplace=True)
zones_with_trips = zones_with_trips.merge(avg_passenger_zone, left_on='PUlocationID', right_on='LocationID', how='left')
zones_with_trips['avg_passenger_count'].fillna(0, inplace=True)
print(zones_with_trips.head())

# OBJECTID  Shape_Lng  Shape_Area          zone  locationID \
# 0  1  0.433479  0.000000  Newark Airport          1
# 1  2  0.433479  0.000000  Jamaica Bay          2
# 2  3  0.884341  0.000014 Allerton/Pelham Gardens          3
# 3  4  0.845667  0.000012 Alphabet City          4
# 4  5  0.852446  0.000000 Astoria/Auburndale          5

# total_trips  PUlocationID  avg_passenger_count
# 0          5.0          1             2.000000
# 1          0.0          2             0.000000
# 2          3.0          3             1.000000
# 3          35.0          4             1.980001
# 4          1.0          5             1.000000
# Python: Input: 102-834f6e4f418c:7: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method. The behavior will change in pandas 1.8. This inplace method will never work because the intermediate object on which we are setting value always behaves as a copy.
# For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method((col: value, inplace=True)' or 'df[col].method(value)' instead, to perform the operation inplace on the original object.

zones_with_trips['avg_passenger_count'].fillna(0, inplace=True)

Find out how often surcharges/extra charges are applied to understand their prevalence

```

Calculates the average passenger count per pickup zone, merges it with location names, and identifies the top 10 and bottom 10 zones for group travel, helping analyze demand for shared rides.

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

```

File Edit View Insert Runtime Tools Help
Files Analyze your files with code written by Gemini Upload
{x}
.. drive sample_data
zones_with_trips['avg_passenger_count'].fillna(0, inplace=True)

Find out how often surcharges/extra charges are applied to understand their prevalence

3.2.16 [5 marks]
Analyse the pickup/dropoff zones or times when extra charges are applied more frequently

# How often is each surcharge applied?

import pandas as pd
surcharge_cols = ['extra', 'mta_tax', 'congestion_surcharge', 'airport_fee']
surcharge_counts = (df[surcharge_cols] > 0).sum().reset_index()
surcharge_counts.columns = ['Surcharge Type', 'Count']
surcharge_counts['Percentage (%)'] = (surcharge_counts['Count'] / len(df)) * 100
print(surcharge_counts)

Surcharge Type Count Percentage (%)
0 extra 169854 60.911663
1 mta_tax 277073 99.361671
2 congestion_surcharge 252428 90.523681
3 airport_fee 1938 0.694996

```

4 Conclusion

Calculates how often each surcharge type (extra, MTA tax, congestion surcharge, airport fee) is applied by counting occurrences and computing their percentage of total trips, helping understand the impact of additional charges.

4. Conclusions

4.1. Final Insights and Recommendations

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

By analysing NYC taxi trip trends across time, days, and locations, we can strategically position cabs, reduce idle time, and improve supply-demand balance to maximize revenue and customer satisfaction.

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.

To maximize efficiency, reduce idle time, and improve service, taxis should be strategically deployed across different NYC zones based on trip trends by time of day, day of the week, and monthly variations.

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

By adopting dynamic, data-driven fare adjustments
Optimize driver earnings.

Stay competitive with Uber & Lyft.
Reduce idle taxi time by balancing supply & demand.
Encourage longer trips while maintaining affordability.