Table of Contents

Sr. No.	Contents	Page No
1	Problem Statement	III
2	Background of the Project	IV
3	Dataset Description	V
4	Software Tool Used	VII
5	Business Scenario	VIII
6	Screenshots of Source code/Output	IX
7	Outcomes	XXVI
8	Conclusion	XXVI
9	References	XXVII

1.1 Problem Statement

This project mirrors tasks commonly encountered in real-world financial and cybersecurity data analysis: preprocessing transaction data, visualizing spending patterns, and applying classification algorithms to distinguish between legitimate and fraudulent transactions. Specifically, we aim to answer questions such as:

• Which features are the most strongly distinguishable between fraudulent and legitimate transactions? (For example, do transaction amount, time, or merchant category play key roles in identifying fraud?)

• How can we best visualize the relationships among these financial features? (We might use correlation heatmaps, transaction frequency plots, or dimensionality reduction techniques like PCA for better feature interpretation.)

• Can classification algorithms such as Logistic Regression, Random Forest, XGBoost, or Neural Networks effectively detect fraudulent transactions while minimizing false positives?

By addressing these questions, the project demonstrates how data scientists, financial analysts, and cybersecurity teams can leverage machine learning to detect fraudulent activities in payment systems - a critical task for preventing financial loss, improving security infrastructure, and maintaining customer trust.

The problem of fraud detection is especially important for banks, credit card companies, and online payment platforms, as it helps identify suspicious behavior in real-time. This approach can also be extended to various domains, such as cybersecurity (intrusion detection), insurance (claim fraud detection), and e-commerce (fake transaction identification).

1.2 Background of the Project:

The project focuses on the critical field of data extraction, transformation, and analysis using advanced data processing and analytical tools, particularly in the context of financial transaction and fraud detection datasets. Data analytics forms the foundation of decision-making in the modern financial sector, especially in areas like cybersecurity, banking, and digital payments. Extracting valuable insights from raw data involves multiple stages to transform it into meaningful and actionable knowledge.

• Data Extraction:
In this project, the dataset is derived from a Payment Card Fraud Detection dataset available on Kaggle, which contains details about legitimate and fraudulent transactions. The dataset includes various transaction-level attributes such as transaction amount, time, merchant ID, cardholder details (anonymized), and fraud labels. Data extraction was performed using Power Query and Python (pandas) to import, inspect, and organize the dataset for further analysis.

• Data Transformation:
Once the raw data was collected, it was prepared for analysis through a detailed data cleaning and preprocessing pipeline. Using Python (pandas and NumPy), several preprocessing steps were applied, including handling missing values, converting data types, normalizing numerical features, and encoding categorical variables. Outlier detection and removal techniques were also considered to ensure data consistency. Additionally, Power Query was used for basic transformation and combining data sources before importing into Power BI for visualization.

• Data Analysis:
In this phase, exploratory data analysis (EDA) was conducted using Python to compute summary statistics (mean, median, mode), examine data distributions, and identify key correlations between variables. Tools like Power BI and Matplotlib/Seaborn were used to create dashboards and

visualizations such as correlation heatmaps, trend charts, and category-wise fraud frequency plots.

Furthermore, machine learning algorithms such as Logistic Regression, Random Forest, and XGBoost can be applied to classify transactions as fraudulent or legitimate based on these features, demonstrating the practical use of data science in real-world fraud detection systems.

This fraud detection dataset is an excellent choice to demonstrate the entire data pipeline - from extraction to transformation and modelling. Although the dataset appears straightforward, it contains complex relationships among multiple variables, making the analysis both challenging and meaningful. By using tools like Power Query, Power BI, Weka tool and Python this project showcases how modern data platforms can work together to detect fraud, improve decision-making, and strengthen financial security systems.

1.3 Dataset Description:

The Payment Card Fraud Detection dataset is a multivariate dataset used for classification, anomaly detection, and visualization tasks in financial data analytics and cybersecurity. Each record represents an individual transaction and contains multiple attributes that describe its characteristics. This dataset is designed to help identify fraudulent activities in financial systems using data-driven and machine learning techniques.

Multiple Numerical and Categorical Features (Attributes):

- Transaction Amount: The monetary value of the transaction.
- Transaction Time: The time or timestamp of the transaction, which helps identify temporal spending patterns.
- Merchant Category: The category or type of business where the transaction occurred.
- Cardholder Information (Anonymized): Encoded attributes representing the customer or cardholder behaviour.
- Location Details: Information related to the geographical or device location (if available).
- Transaction Mode: Whether the transaction was made online, in-store, or through another medium.
- Derived Features: Additional calculated columns such as transaction frequency or average spending behavior for each user.
-

• Target Variable (Fraud Indicator):

- 0 → Legitimate Transaction
- 1 → Fraudulent Transaction

Each row in the dataset represents a single card transaction, with corresponding features and a binary target label indicating whether the transaction was fraudulent. The dataset is organized in a structured tabular format suitable for analysis and machine learning model training.
Despite variations in feature representation or anonymization methods, this dataset is ideal for applying data preprocessing, feature selection, and predictive modeling techniques in the financial domain.

They are commonly used to:

- Demonstrate the Application of Classification Algorithms
  - Such as Logistic Regression, Random Forest, Decision Tree, Support Vector Machines (SVM), and XGBoost, to classify transactions as fraudulent or legitimate based on transaction patterns.
- Provide a Platform for Data Visualization
  - Using tools like Power BI, Matplotlib, and Seaborn to create heatmaps, bar charts, and time-based visualizations, showing fraud frequency by amount, location, or time of day.
- Serve as a Benchmark for Anomaly Detection and Clustering
  - Algorithms like K-Means, DBSCAN, or Isolation Forest can be used to identify unusual transaction patterns and detect fraud without predefined labels (unsupervised learning).

This dataset serves as an excellent introduction to financial data analytics and fraud detection systems, offering insight into how machine learning and visualization tools such as Python, Power BI, and Power Query can be used to enhance fraud monitoring.

It is particularly valuable for understanding transaction behavior patterns, improving fraud prevention systems, and supporting secure digital payment ecosystems.

Product SKU: Unique Alphanumeric sequence assigned to a specific product variation.

Store ID: Unique Identifier For Each Store

Customer Loyalty Tier: Customer loyalty program tier

1.4 Software Tools Used:

The project makes use of a combination of Python, Power Query, Weka tool and Power BI - powerful tools designed for data extraction, transformation, visualization. These tools complement each other to provide a complete data analytics workflow, from raw data import to interactive dashboard visualization and predictive modeling.
Each tool was selected for its specific strengths in handling large datasets, performing advanced analytics, and presenting insights effectively.

• **Python (Google Colab Environment):**
Python is a versatile programming language widely used for data science, machine learning, and statistical analysis. In this project, Python was used for the entire data preprocessing and exploratory analysis pipeline. The key libraries employed include pandas, NumPy, Matplotlib, and Seaborn.
Python was used to:

- Load and Inspect Data: Using pandas, the Payment Card Fraud dataset was imported, and preliminary data checks (shape, data types, missing values) were performed.
- Data Cleaning and Transformation: Handling missing values, encoding categorical variables, normalizing numeric columns, and detecting outliers were performed using pandas and NumPy functions.
- Exploratory Data Analysis (EDA): Summary statistics such as mean, median, and mode were computed. Visualizations like histograms, boxplots, correlation heatmaps, and pairplots were generated using Matplotlib and Seaborn to understand feature relationships.
- Data Visualization: Visual exploration and trend identification were enhanced through Python-based visualizations before integrating results with Power BI.

• Power Query:

Power Query was used for the data extraction and initial transformation phase. It simplifies the process of importing datasets from multiple sources and performing basic transformations before deeper analysis.

- Data was imported, cleaned, and reshaped using Power Query's interface.
- Redundant or inconsistent fields were removed, and column names were standardized.
- The processed data was then exported for visualization and modeling stages.

• Power BI:

Power BI was employed for creating interactive dashboards and visual reports to represent insights derived from the dataset.
It was used to:

- Visualize fraud frequency by transaction amount, time, and category using bar charts, pie charts, and heatmaps.
- Display correlation relationships between features through matrix and trend visuals.
- Design real-time analytical dashboards summarizing fraud detection patterns and model predictions.

Power BI's integration with Power Query allowed for seamless data refresh and transformation, while Python's analytical outputs were imported for advanced visual representation.
The decision to use Python, Power Query, and Power BI reflects the project's need for a complete data analytics ecosystem-covering everything from preprocessing and transformation to modeling and visualization. This combination not only demonstrates modern data science workflows but also highlights practical applications of analytics in fraud detection, financial risk assessment, and data-driven decision-making.

• Weka:

Weka (Waikato Environment for Knowledge Analysis) is an open-source tool for machine learning and data mining, offering a graphical interface for model building and evaluation without extensive coding.

Applications of Weka in the project:

- Dataset Import and Preprocessing:
  The cleaned dataset from Power Query was imported into Weka for additional machine learning experimentation and validation.

- Model Building:
  Classification algorithms such as J48 Decision Tree, Naïve Bayes, and Random Forest were trained to identify fraudulent transactions based on key features.

- Model Evaluation:
  Weka's evaluation tools were used to assess accuracy, precision, recall, and F1-score, allowing comparison with Python-based models.

- Visualization:
  Weka's built-in visualization tools (such as attribute histograms, scatter plots, and decision tree diagrams) provided intuitive visual interpretations of classification results.

1.5 Business Scenario:

In the context of a business scenario, many organizations today rely on large-scale financial and transactional datasets to make strategic, data-driven decisions. Understanding the hidden patterns within this data can lead to better fraud prevention, enhanced financial security, and improved operational efficiency.

The problem of classifying transactions as fraudulent or legitimate is analogous to several real-world business applications that emphasize risk assessment, anomaly detection, and decision automation.

• Risk Assessment and Management:
Just as this project aims to classify transactions based on their characteristics, banks, e-commerce platforms, and payment processors assess transaction risk to prevent fraud and reduce financial losses. By analyzing variables such as transaction amount, frequency, and merchant category, businesses can use machine learning models to automatically flag suspicious activity and minimize exposure to fraudulent behavior.

• Customer Segmentation and Behavior Analysis:
Similar to classifying volcanoes based on geological features, organizations segment customers based on their spending patterns, frequency, and transaction habits. For example, businesses can use these insights to differentiate between high-value customers, frequent spenders, and potential fraudsters-helping tailor marketing strategies and improve service personalization.

• Financial Compliance and Monitoring:
Fraud detection techniques are directly applicable to regulatory compliance and anti-money laundering (AML) activities. Businesses can analyse transaction flows to detect unusual patterns that might indicate money laundering, unauthorized access, or data breaches-ensuring adherence to compliance standards and building customer trust.

• Predictive Analytics for Risk Mitigation:
Just as predictive models can estimate volcanic activity based on historical data, businesses use machine learning algorithms to forecast fraud probability for upcoming transactions. Predictive models help financial institutions prevent fraud before it occurs, leading to faster decision-making and reduced manual intervention in risk management.

• Operational Efficiency and Automation:
Machine learning-based fraud detection systems improve operational efficiency by automating the process of transaction verification. This reduces manual investigation efforts, lowers operational costs, and enhances the speed and accuracy of fraud identification-allowing businesses to focus more on innovation and customer experience.

By exploring the Payment Card Fraud Detection dataset and applying data analysis, visualization, and machine learning techniques, this project demonstrates how data-driven decision-making supports financial security, operational efficiency, and strategic business growth.
In today's data-centric environment, where millions of transactions occur every second, the ability to extract, analyse, and interpret data is essential for gaining a competitive edge and ensuring a secure and trustworthy digital economy.

1.6 Screenshots of Source code/Output:

Python:

```
from google.colab import files
uploaded = files.upload()
```

Choose Files | No file chosen — Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving luxury_cosmetics_fraud_analysis_2025.csv to luxury_cosmetics_fraud_analysis_2025.csv
```

```
import pandas as pd
```

```
import io
import pandas as pd
df = pd.read_csv(io.BytesIO(uploaded['luxury_cosmetics_fraud_analysis_2025.csv']))
```

LOADING DATSET

Read and Explore Dataset

```
print("Shape of dataset:", df.shape)  # Prints the number of rows and columns in the DataFrame 'df'
```

```
Shape of dataset: (2133, 16)
```

```
print("First 5 rows:", df.head())  # Displays the first 5 rows of the DataFrame 'df' to get a preview of the data
```

```
First 5 rows:                          Transaction_ID                       Customer_ID  \
0  702bdd9b-9c93-41e3-9dbb-a849b2422080  119dca0b-8554-4b2d-9bec-e964eaf6af97
1  2e64c346-36bc-4acf-bc2b-8b0fdf46abc5  299df086-26c4-4708-b6d7-fcaeceb14637
2  29ad1278-70ce-421f-8d81-23816b39f4ac  dfa3d24d-b935-49a5-aa1d-7d57a44d8773
3  07dc4894-e0eb-48f1-99a7-1942b1973d9b  7a67e184-9369-49ee-aeac-18f5b51b230f
4  ae407054-5543-429c-918a-cdcc42ea9782  cf14730a-8f5a-453d-b527-39a278852b27

  Transaction_Date Transaction_Time  Customer_Age Customer_Loyalty_Tier  \
0       2025-07-27         04:04:15          56.0                Silver
1       2025-03-14         20:23:23          46.0              Platinum
2       2025-02-20         12:36:02          32.0                Silver
3       2025-04-25         19:09:43          60.0                Bronze
4       2025-04-17         14:23:23           NaN              Platinum

       Location          Store_ID       Product_SKU Product_Category  \
0  San Francisco       FLAGSHIP-LA   NEBULA-SERUM-07        Concealer
1         Zurich  BOUTIQUE-SHANGHAI  STELLAR-FOUND-03         Lipstick
2          Milan        POPUP-TOKYO    SOLAR-BLUSH-04          Mascara
3         London       BOUTIQUE-NYC    GALAXIA-SET-08            Serum
4          Miami       BOUTIQUE-NYC     LUNAR-MASC-02            Serum

   Purchase_Amount  Payment_Method Device_Type       IP_Address  Fraud_Flag  \
0           158.24  Mobile Payment     Desktop   239.249.58.237           0
1            86.03     Credit Card      Tablet     84.49.227.90           0
2           255.69       Gift Card     Desktop     79.207.35.55           0
3           282.76       Gift Card      Mobile  176.194.167.253           0
4           205.86       Gift Card      Mobile    166.31.46.111           0

   Footfall_Count
0             333
1             406
2              96
3             186
4             179
```

Read and Explore Dataset

```python
# Data types
print("\n--- Data Types ---")
print(df.dtypes)  # Prints the data type of each column in the DataFrame

# Mean, Median, Mode
print("\n--- Mean ---\n", df.mean(numeric_only=True))    # Calculates and prints the mean of all numeric columns
print("\n--- Median ---\n", df.median(numeric_only=True))  # Calculates and prints the median of all numeric columns
print("\n--- Mode ---\n", df.mode().iloc[0])  # Calculates and prints the mode (most frequent value) for each column; selects the first row in case of multiple modes
```

```
--- Data Types ---
Transaction_ID            object
Customer_ID               object
Transaction_Date          object
Transaction_Time          object
Customer_Age             float64
Customer_Loyalty_Tier     object
Location                  object
Store_ID                  object
Product_SKU               object
Product_Category          object
Purchase_Amount          float64
Payment_Method            object
Device_Type               object
IP_Address                object
Fraud_Flag                 int64
Footfall_Count             int64
dtype: object

--- Mean ---
 Customer_Age        41.684262
Purchase_Amount     174.614074
Fraud_Flag            0.030942
Footfall_Count      272.461791
dtype: float64

--- Median ---
 Customer_Age        42.00
Purchase_Amount     174.18
Fraud_Flag            0.00
Footfall_Count      269.00
dtype: float64

--- Mode ---
 Transaction_ID           0004b5eb-d8e6-4d44-841d-6ba42d0aa54a
Customer_ID              001d3772-360f-455a-b16d-0faf018d9470
Transaction_Date                                   2025-03-09
Transaction_Time                                     00:15:02
Customer_Age                                             50.0
Customer_Loyalty_Tier                                 Bronze
Location                                              Sydney
Store_ID                                      FLAGSHIP-PARIS
Product_SKU                                   ECLIPSE-EYE-10
Product_Category                                       Serum
Purchase_Amount                                       186.63
Payment_Method                                    Debit Card
Device_Type                                           Tablet
IP_Address                                    1.118.110.46
Fraud_Flag                                             0.0
Footfall_Count                                       485.0
```

DATA Types, Mean, Median,Mode

info()

```
print("Info:")
df.info()  # This alone is enough to display the DataFrame summary
```

```
Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2133 entries, 0 to 2132
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Transaction_ID        2133 non-null   object
 1   Customer_ID           2133 non-null   object
 2   Transaction_Date      2133 non-null   object
 3   Transaction_Time      2133 non-null   object
 4   Customer_Age          2027 non-null   float64
 5   Customer_Loyalty_Tier 2027 non-null   object
 6   Location              2133 non-null   object
 7   Store_ID              2133 non-null   object
 8   Product_SKU           2133 non-null   object
 9   Product_Category      2133 non-null   object
 10  Purchase_Amount       2133 non-null   float64
 11  Payment_Method        2027 non-null   object
 12  Device_Type           2133 non-null   object
 13  IP_Address            2133 non-null   object
 14  Fraud_Flag            2133 non-null   int64
 15  Footfall_Count        2133 non-null   int64
dtypes: float64(2), int64(2), object(12)
memory usage: 266.8+ KB
```

INFO

describe

```
print("Describe:")
print(df.describe(include="all"))  # Prints summary statistics for all columns (numeric and non-numeric), including coun
```

```
Describe:
                              Transaction_ID  \
count                                   2133
unique                                  2133
top     83108c8e-2ede-4e86-a0f6-ea4875c9e523
freq                                       1
mean                                     NaN
std                                      NaN
min                                      NaN
25%                                      NaN
50%                                      NaN
75%                                      NaN
max                                      NaN

                               Customer_ID Transaction_Date  \
count                                 2133             2133
unique                                2133              180
top     e94cecaf-9db8-49f4-97f6-a17bb5dd1187       2025-03-09
freq                                     1               20
mean                                   NaN              NaN
std                                    NaN              NaN
min                                    NaN              NaN
25%                                    NaN              NaN
50%                                    NaN              NaN
75%                                    NaN              NaN
max                                    NaN              NaN

       Transaction_Time  Customer_Age Customer_Loyalty_Tier Location  \
count              2133   2027.000000                  2027     2133
unique             2106           NaN                     5       20
top            21:38:38           NaN                Bronze   Sydney
freq                  2           NaN                   808      128
mean                NaN     41.684262                   NaN      NaN
std                 NaN     13.718110                   NaN      NaN
min                 NaN     18.000000                   NaN      NaN
25%                 NaN     30.000000                   NaN      NaN
```

Describe

dytypes

```
print("Data Types")
print(df.dtypes)  # Prints the data type (e.g., int64, float64, object) of each column in the DataFrame
```

```
Data Types
Transaction_ID           object
Customer_ID              object
Transaction_Date         object
Transaction_Time         object
Customer_Age            float64
Customer_Loyalty_Tier    object
Location                 object
Store_ID                 object
Product_SKU              object
Product_Category         object
Purchase_Amount         float64
Payment_Method           object
Device_Type              object
IP_Address               object
Fraud_Flag                int64
Footfall_Count            int64
dtype: object
```

Dytypes

mean

```
print("Mean:")
print(df.mean(numeric_only=True))  # Calculates and prints the mean of all numeric columns in the DataFrame
```

```
Mean:
Customer_Age        41.684262
Purchase_Amount    174.614074
Fraud_Flag           0.030942
Footfall_Count     272.461791
dtype: float64
```

mean

median

```
print("Median:")
print(df.median(numeric_only=True))  # Calculates and prints the median of all numeric columns in the DataFrame
```

```
Median:
Customer_Age        42.00
Purchase_Amount    174.18
Fraud_Flag           0.00
Footfall_Count     269.00
dtype: float64
```

Median

Sort any 5 attributes & store in Excel

```
cols_to_sort = df.columns[:5]                # Selects the first 5 columns from the DataFrame to use for sorting
sorted_df = df.sort_values(by=list(cols_to_sort))  # Sorts the DataFrame by these selected columns in ascending order
print(sorted_df)                             # Prints the sorted DataFrame
```

```
                           Transaction_ID  \
1410  0004b5eb-d8e6-4d44-841d-6ba42d0aa54a
1854  00196faa-626a-42aa-b1bd-a93fae0b475e
2118  001a6603-1da9-4231-b1db-94cc5abd1fcf
1441  003eb7b3-0fb8-4a7e-adb7-338394c3be75
926   006612c8-f1d7-4477-901f-5afd399c6b47
...                                    ...
1152  ffb0b7f9-9521-42ca-a763-a7e62c50b420
1102  ffd209b1-0174-4a52-9cc7-25ab2984b34c
1115  ffefb6c5-f008-4a88-8472-89e7346ed127
1787  fff03982-4c4e-460d-a16c-ed5c4a461d0b
685   fff3a3d6-f42f-4b48-b617-1f343eb573ff

                           Customer_ID Transaction_Date Transaction_Time  \
1410  ee4ab850-16d6-44f8-ad0c-2d05af2e0900       2025-07-29         14:38:30
1854  e9b78ce1-8c76-4e52-8e96-e1ce524409d4       2025-03-09         01:29:41
2118  f8eec5b5-e2eb-4044-8d10-25dcf74473e5       2025-02-28         19:36:08
1441  1271ec03-142c-4872-8cfd-04a92e571c99       2025-04-25         17:26:16
926   4f776aa7-7217-43a1-8195-bce0aa595268       2025-07-25         11:56:26
...                                    ...              ...              ...
1152  aa809630-6be3-44f2-a933-22699ba4e73a       2025-04-16         02:50:38
1102  fb32d05f-56d9-43f9-9cff-84a9c706f6fb       2025-05-29         01:00:54
1115  4ca2f3b4-b9c2-4992-9436-cc76814eaf2d       2025-06-17         18:30:51
1787  3e52d3b1-b1e7-486e-a976-448bd3fafca9       2025-07-18         18:58:13
685   e0c9ed0d-cba7-4b34-88c4-c7d57aba2a11       2025-04-24         11:03:31

      Customer_Age Customer_Loyalty_Tier   Location          Store_ID  \
1410          28.0                Bronze   New York      FLAGSHIP-ROME
1854          19.0                   NaN      Milan  CONCESSION-LONDON
2118          25.0              Platinum     Geneva  CONCESSION-LONDON
1441          57.0                  Gold      Milan        FLAGSHIP-LA
926           61.0                  Gold  Hong Kong      FLAGSHIP-ROME
...            ...                   ...        ...               ...
1152          63.0                Bronze  Las Vegas     FLAGSHIP-PARIS
1102          44.0              Platinum     Cannes        FLAGSHIP-LA
1115          27.0                Bronze     Sydney     FLAGSHIP-PARIS
1787          46.0                Silver  Singapore        POPUP-TOKYO
```

Sorting and storing

Identify the type of data (Quantitative or Qualitative)

```
for col in df.columns:                    # Loop through each column in the DataFrame
    if df[col].dtype in ["int64", "float64"]:  # Check if the column's data type is numeric (integer or float)
        print(f"{col} --> Quantitative")  # If numeric, label the column as Quantitative
    else:
        print(f"{col} --> Qualitative")   # Otherwise, label it as Qualitative (categorical or other non-numeric types)
```

```
Transaction_ID --> Qualitative
Customer_ID --> Qualitative
Transaction_Date --> Qualitative
Transaction_Time --> Qualitative
Customer_Age --> Quantitative
Customer_Loyalty_Tier --> Qualitative
Location --> Qualitative
Store_ID --> Qualitative
Product_SKU --> Qualitative
Product_Category --> Qualitative
Purchase_Amount --> Quantitative
Payment_Method --> Qualitative
Device_Type --> Qualitative
IP_Address --> Qualitative
Fraud_Flag --> Quantitative
Footfall_Count --> Quantitative
```

Quantitative or Qualitative

Store one column in in one object and display

```
col_data = df.iloc[:, 0]                # Selects all rows from the first column of the DataFrame and stores it in 'col_data'
print("Stored Column Data:", col_data.head())  # Prints the first 5 values of the stored column data
```

```
Stored Column Data: 0    702bdd9b-9c93-41e3-9dbb-a849b2422080
1    2e64c346-36bc-4acf-bc2b-8b0fdf46abc5
2    29ad1278-70ce-421f-8d81-23816b39f4ac
3    07dc4894-e0eb-48f1-99a7-1942b1973d9b
4    ae407054-5543-429c-918a-cdcc42ea9782
Name: Transaction_ID, dtype: object
```

Storing one column in one object and display

Replace the missing values with appropriate data

```
for col in df.columns:  # Loop through each column in the DataFrame
    if df[col].dtype in ["int64", "float64"]:  # Check if the column is numeric
        df[col].fillna(df[col].mean(), inplace=True)  # Replace missing values with the column mean
    else:
        df[col].fillna(df[col].mode()[0], inplace=True)  # Replace missing values with the most frequent value (mode) for non-numeric columns

print("Missing values handled")  # Confirmation message that missing values are filled
print(df.isnull().sum())  # Print the count of remaining missing values in each column (should be zero)
```

```
Missing values handled
Transaction_ID          0
Customer_ID             0
Transaction_Date        0
Transaction_Time        0
Customer_Age            0
Customer_Loyalty_Tier   0
Location                0
Store_ID                0
Product_SKU             0
Product_Category        0
Purchase_Amount         0
Payment_Method          0
Device_Type             0
IP_Address              0
Fraud_Flag              0
Footfall_Count          0
dtype: int64
```

Replace the missing values with appropriate data

Count text attributes and numeric attributes

```
num_count = len(df.select_dtypes(include=["int64", "float64"]).columns)  # Count numeric columns (integers and floats)
text_count = len(df.select_dtypes(include=["object"]).columns)          # Count text/categorical columns (object dtype)

print("\nNumber of Numeric Attributes:", num_count)  # Print number of numeric columns
print("Number of Text Attributes:", text_count)      # Print number of text/categorical columns
```

```
Number of Numeric Attributes: 4
Number of Text Attributes: 12
```

Count text attributes and numeric attributes

Filter only particular value from the column(Example, any tex or >190)

```
if df.select_dtypes(include=["int64", "float64"]).shape[1] > 0:  # Check if there is at least one numeric column
    num_col = df.select_dtypes(include=["int64", "float64"]).columns[0]  # Select the first numeric column
    filtered_num = df[df[num_col] > 190]  # Filter rows where the value in this column is greater than 190
    print(f"\nRows where {num_col} > 190:\n", filtered_num.head())  # Print first 5 filtered rows

if df.select_dtypes(include=["object"]).shape[1] > 0:  # Check if there is at least one text/categorical column
    text_col = df.select_dtypes(include=["object"]).columns[0]  # Select the first text column
    filtered_text = df[df[text_col] == df[text_col].iloc[0]]  # Filter rows where the value equals the first row's value in that column
    print(f"\nRows where {text_col} = {df[text_col].iloc[0]}:\n", filtered_text.head())  # Print first 5 filtered rows
```

```
Rows where Customer_Age > 190:
 Empty DataFrame
Columns: [Transaction_ID, Customer_ID, Transaction_Date, Transaction_Time, Customer_Age, Customer_Loyalty_Tier, Location, Store_ID, Product_SKU, Product_Category, Purchase_Amount, Payment_Method, D
Index: []

Rows where Transaction_ID = 702bdd9b-9c93-41e3-9dbb-a849b2422080:
                        Transaction_ID                       Customer_ID  \
0  702bdd9b-9c93-41e3-9dbb-a849b2422080  119dca0b-8554-4b2d-9bec-e964eaf6af97

  Transaction_Date Transaction_Time  Customer_Age Customer_Loyalty_Tier  \
0       2025-07-27         04:04:15          56.0               Silver

        Location     Store_ID      Product_SKU Product_Category  \
0  San Francisco  FLAGSHIP-LA  NEBULA-SERUM-07        Concealer

   Purchase_Amount  Payment_Method Device_Type      IP_Address  Fraud_Flag  \
0           158.24  Mobile Payment     Desktop  239.249.58.237           0

   Footfall_Count
0             333
```

Filter only particular value from the column(Example, any tex or >190)

```
df.loc[4, 'Purchase_Amount'] += 500  # Add 500 to the 'Purchase_Amount' value in the 5th row (index 4)
print("Updated 5th record (added 500):", df.loc[4, 'Purchase_Amount'])  # Print updated value

df.loc[5, 'Purchase_Amount'] -= 99.99  # Subtract 99.99 from the 'Purchase_Amount' in the 6th row (index 5)
print("Updated 6th record (subtracted 99.99):", df.loc[5, 'Purchase_Amount'])  # Print updated value

df.loc[0, 'Purchase_Amount'] *= 3  # Multiply the 'Purchase_Amount' in the 1st row (index 0) by 3
print("Updated 1st record (multiplied by 3):", df.loc[0, 'Purchase_Amount'])  # Print updated value

df.loc[36, 'Purchase_Amount'] /= 3  # Divide the 'Purchase_Amount' in the 37th row (index 36) by 3
print("Updated 37th record (divided by 3):", df.loc[36, 'Purchase_Amount'])  # Print updated value
```

```
Updated 5th record (added 500): 705.86
Updated 6th record (subtracted 99.99): 35.92
Updated 1st record (multiplied by 3): 474.72
Updated 37th record (divided by 3): 34.03
```

Increases the purchase amount by 500 for the 5th customer record.

```
result1 = df.loc[0, 'Customer_Age'] > df.loc[9, 'Customer_Age']  # Check if 'Customer_Age' in 1st row is greater than in 10th row
print("Is Customer_Age in first row > tenth row?", result1)  # Print the boolean result

result2 = df.loc[2, 'Customer_Age'] == df.loc[4, 'Customer_Age']  # Check if 'Customer_Age' in 3rd row equals that in 5th row
print("Is Customer_Age in third row equal to fifth row?", result2)  # Print the boolean result
```

```
Is Customer_Age in first row > tenth row? True
Is Customer_Age in third row equal to fifth row? False
```

Compares the age of the 1st and 10th customer to check which is older.

```python
fraud_giftcard_count = df[(df['Fraud_Flag'] == True) & (df['Payment_Method'] == 'Gift Card')].shape[0]
# Count rows where 'Fraud_Flag' is True and 'Payment_Method' is 'Gift Card'
print("Count of fraud purchases using Gift Card:", fraud_giftcard_count)  # Print the count

premium_payment_methods = df[df['Payment_Method'].isin(['Credit Card', 'PayPal'])]
# Filter rows where 'Payment_Method' is either 'Credit Card' or 'PayPal'
print("Extracted premium_payment_methods:")
print(premium_payment_methods.head())  # Display first 5 rows of this filtered DataFrame

high_value_female_fraud = df[
    (df['Location'].str.contains("France", case=False, na=False)) &  # Location contains 'France', case-insensitive, ignore NaNs
    (df['Purchase_Amount'] > 1000) &                                 # Purchase amount greater than 1000
    (df['Fraud_Flag'] == True)                                       # Fraud flag is True
]
print("High-value fraud cases from France (Purchase > $1000):")
print(high_value_female_fraud.head())  # Show first 5 matching rows
```

```
Count of fraud purchases using Gift Card: 16
Extracted premium_payment_methods:
                         Transaction_ID  \
1   2e64c346-36bc-4acf-bc2b-8b0fdf46abc5
8   d1563d4b-de86-47c5-9df6-bc78259834a8
9   2ee80052-067d-4ff2-87a6-ec8919d13546
11  7a7be10d-b2ca-4ae7-a3f1-430d1eaf61fe
12  152955a2-b19a-4670-a9dc-71d7f87930c7

                          Customer_ID Transaction_Date Transaction_Time  \
1   299df086-26c4-4708-b6d7-fcaeceb14637       2025-03-14         20:23:23
8   8d7bd046-4a87-4c8c-850e-4d55f4767119       2025-08-06         20:17:15
9   3ae3b36f-25bb-48ca-afee-7c524b58e1a0       2025-07-03         03:01:47
11  2dba69e9-e8cc-4cf1-a809-fab0ea8568f3       2025-07-15         12:58:35
12  2b019bf7-67ca-4a5e-b09a-5b899e4e20bf       2025-06-02         03:16:49

    Customer_Age Customer_Loyalty_Tier   Location          Store_ID  \
1           46.0              Platinum     Zurich  BOUTIQUE-SHANGHAI
8           40.0                  Gold   Shanghai        POPUP-TOKYO
9           28.0                Bronze      Tokyo  BOUTIQUE-SHANGHAI
11          41.0                Bronze  Las Vegas        POPUP-TOKYO
12          53.0                  Gold     Sydney  BOUTIQUE-SHANGHAI

        Product_SKU Product_Category  Purchase_Amount Payment_Method  \
1   STELLAR-FOUND-03         Lipstick           86.03    Credit Card
8     ECLIPSE-EYE-10            Serum          113.85    Credit Card
9   ORION-CONCEAL-09          Mascara          203.38    Credit Card
11    ECLIPSE-EYE-10        Concealer           62.07    Credit Card
12    CELESTE-EYE-05    Setting Spray           50.26    Credit Card

   Device_Type       IP_Address  Fraud_Flag  Footfall_Count
1       Tablet     84.49.227.90           0             406
8       Tablet  175.162.153.155           0             481
9      Desktop   236.222.176.45           0             118
11      Tablet   219.150.31.189           1             384
12      Tablet     231.27.0.183           0              63
High-value fraud cases from France (Purchase > $1000):
Empty DataFrame
Columns: [Transaction_ID, Customer_ID, Transaction_Date, Transaction_Time, Customer_Age, Customer_Loyalty_Tier, Location, Store_ID, Product_SKU, Product_Category, Purchase_Amount, Payment_Method, D
Index: []
```

Counts how many fraudulent transactions used a gift card as payment.

```
nan_count = df.isna().sum().sum()
# Count total number of missing (NaN) values in the entire DataFrame by summing null counts column-wise and then overall
print("Total missing (NaN) values in the DataFrame:", nan_count)  # Print total missing values

credit_card_seniors = df[(df['Payment_Method'] == 'Credit Card') & (df['Customer_Age'] > 45)]
# Filter rows where 'Payment_Method' is 'Credit Card' and 'Customer_Age' is greater than 45
print("Customers using Credit Card with Customer_Age > 45:")
print(credit_card_seniors.head())  # Show first 5 such customers

max_age = df['Customer_Age'].max()  # Find the maximum value in the 'Customer_Age' column
print("Maximum Customer_Age:", max_age)  # Print the maximum customer age
```

```
Total missing (NaN) values in the DataFrame: 0
Customers using Credit Card with Customer_Age > 45:
                         Transaction_ID  \
1   2e64c346-36bc-4acf-bc2b-8b0fdf46abc5
12  152955a2-b19a-4670-a9dc-71d7f87930c7
19  52b37161-d601-4635-ae3c-ba3a11d1d23e
21  8b13ed0c-e1e4-4cb0-9717-babd8b58c8a5
72  187b0df8-0e1f-4c11-aa01-0172dcd7e492

                          Customer_ID Transaction_Date Transaction_Time
1   299df086-26c4-4708-b6d7-fcaeceb14637       2025-03-14         20:23:23
12  2b019bf7-67ca-4a5e-b09a-5b899e4e20bf       2025-06-02         03:16:49
19  76d93b6d-c64d-482f-9011-24e6951ccf4c       2025-07-18         21:54:20
21  dc6dc635-47f7-404d-aa33-eba56816c6e7       2025-07-07         02:35:34
72  c2a51b1a-1104-4e9e-ad11-68d2f9eb80ca       2025-07-26         22:20:46

    Customer_Age Customer_Loyalty_Tier Location              Store_ID  \
1           46.0              Platinum   Zurich     BOUTIQUE-SHANGHAI
12          53.0                  Gold   Sydney     BOUTIQUE-SHANGHAI
19          61.0                Bronze    Dubai  CONCESSION-SINGAPORE
21          55.0              Platinum    Paris           POPUP-MILAN
72          53.0                Bronze   Sydney           POPUP-TOKYO

         Product_SKU   Product_Category  Purchase_Amount Payment_Method  \
1   STELLAR-FOUND-03           Lipstick            86.03    Credit Card
12    CELESTE-EYE-05      Setting Spray            50.26    Credit Card
19     AURORA-LIP-01  Eyeshadow Palette            86.21    Credit Card
21     LUNAR-MASC-02          Concealer            92.05    Credit Card
72  STELLAR-FOUND-03              Blush           113.29    Credit Card

   Device_Type     IP_Address  Fraud_Flag  Footfall_Count
1       Tablet   84.49.227.90           0             406
12      Tablet   231.27.0.183           0              63
19      Laptop      3.36.8.73           0             386
21      Tablet  22.70.212.233           0             498
72      Mobile  46.225.35.251           0             146
Maximum Customer_Age: 65.0
```

Calculates the total number of missing (NaN) values in the entire dataset.

```
def grade_customer_age(age):
    if 1 < age <= 25:
        return 'Low'          # Age between 2 and 25 is graded as 'Low'
    elif 25 < age <= 50:
        return 'Medium'       # Age between 26 and 50 is graded as 'Medium'
    elif 50 < age <= 100:
        return 'High'         # Age between 51 and 100 is graded as 'High'
    else:
        return 'Unknown'      # Any other age value (e.g., <=1 or >100) is 'Unknown'

df['Customer_Age_Grade'] = df['Customer_Age'].apply(grade_customer_age)  # Apply grading function to 'Customer_Age' column, create new colum
print("Added Customer_Age_Grade column. Sample:")
print(df[['Customer_Age', 'Customer_Age_Grade']].head())  # Show first 5 rows of the age and corresponding grade
```

```
Added Customer_Age_Grade column. Sample:
   Customer_Age Customer_Age_Grade
0     56.000000              High
1     46.000000            Medium
2     32.000000            Medium
3     60.000000              High
4     41.684262            Medium
```

Show first 5 rows of the age and corresponding grade

```
m_lux = luxx['Customer_Age'].mean()          # Calculate the mean (average) of the 'Customer_Age' column in the DataFrame 'luxx'
print("mean of customer age is: ", m_lux)   # Print the calculated mean value
```

```
mean of customer age is:  41.68426245683276
```

Mean

```
s_age = luxx['Customer_Age'].std()          # Calculate the standard deviation of 'Customer_Age'
s_amt = luxx['Purchase_Amount'].std()       # Calculate the standard deviation of 'Purchase_Amount'

M_age = luxx['Customer_Age'].median()       # Calculate the median of 'Customer_Age'
M_amt = luxx['Purchase_Amount'].median()    # Calculate the median of 'Purchase_Amount' (missing parentheses fixed)

print("standard deviation of customer age is: ", s_age)       # Print std dev of customer age
print("standard deviation of purchase amount is: ", s_amt)    # Print std dev of purchase amount
print("median of customer age is: ", M_age)                    # Print median of customer age
print("median of purchase amount is: ", M_amt)                 # Print median of purchase amount
```

```
standard deviation of customer age is:  13.372739906052226
standard deviation of purchase amount is:  73.54582997459002
median of customer age is:  41.684262456832755
median of purchase amount is:  174.18
```

Standard deviation, median

```
na_count = luxx.isna().sum()          # Count the number of missing (NaN) values in each column
na = na_count[na_count > 0]           # Filter columns that have at least one missing value
print("variable with na")
print(na)                            # Print the counts of missing values per column that have NaNs
print("columns with missing values:", list(na.index))  # Print a list of column names that contain missing values
```

```
variable with na
Series([], dtype: int64)
columns with missing values: []
```

Count, filter

```
import numpy as np
bins = [0, 100, 200, 300, 400]                    # Define bin edges for categorizing 'Purchase_Amount'
labels = [100, 200, 300, 400]                     # Define labels corresponding to each bin

luxx['Purchase_Amount_bin'] = pd.cut(
    luxx['Purchase_Amount'], bins=bins, labels=labels
)  # Categorize 'Purchase_Amount' into bins with specified labels

print(luxx[['Purchase_Amount', 'Purchase_Amount_bin']].head())  # Display first 5 rows of 'Purchase_Amount' and its binned category
```

```
   Purchase_Amount Purchase_Amount_bin
0           474.72                 NaN
1            86.03                 100
2           255.69                 300
3           282.76                 300
4           705.86                 NaN
```

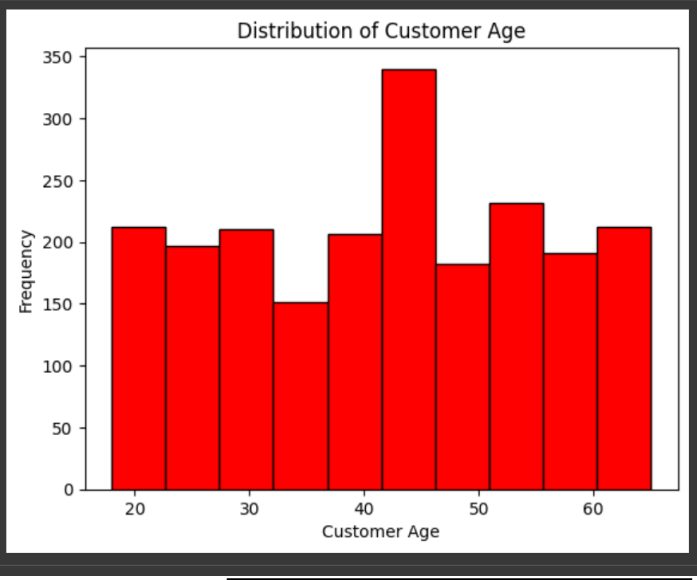
Display first 5 rows purchse_amount

```
import pandas as pd

luxx['Transaction_Date'] = pd.to_datetime(luxx['Transaction_Date'])
# Convert 'Transaction_Date' to datetime format

month_dict = {
    1: 'January', 2: 'February', 3: 'March', 4: 'April',
    5: 'May', 6: 'June', 7: 'July', 8: 'August',
    9: 'September', 10: 'October', 11: 'November', 12: 'December'
}
# Dictionary mapping month numbers to month names

luxx['Purchase_Month'] = luxx['Transaction_Date'].dt.month
# Extract month number from 'Transaction_Date'

luxx['Purchase_Month_Name'] = luxx['Purchase_Month'].map(month_dict)
# Map month number to month name (new column)

print(luxx[['Transaction_Date', 'Purchase_Month_Name']].head())
# Display first 5 rows of transaction date and month name

for i in luxx.index:
    month_num = luxx.loc[i, 'Purchase_Month']  # Get numeric month for the current row
    luxx.loc[i, 'Purchase_Month'] = month_dict.get(month_num, month_num)
    # Replace numeric month with month name using dictionary (fallback to number if missing)

print(luxx[['Purchase_Month']].head())
# Display first 5 rows of the updated 'Purchase_Month' column (now with month names)
```

```
  Transaction_Date Purchase_Month_Name
0       2025-07-27                July
1       2025-03-14               March
2       2025-02-20            February
3       2025-04-25               April
4       2025-04-17               April
/tmp/ipython-input-3381567591.py:24: FutureWarning: Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value 'July' has dtype incompatible wi
  luxx.loc[i, 'Purchase_Month'] = month_dict.get(month_num, month_num)
  Purchase_Month
0           July
1          March
2       February
3          April
4          April
```

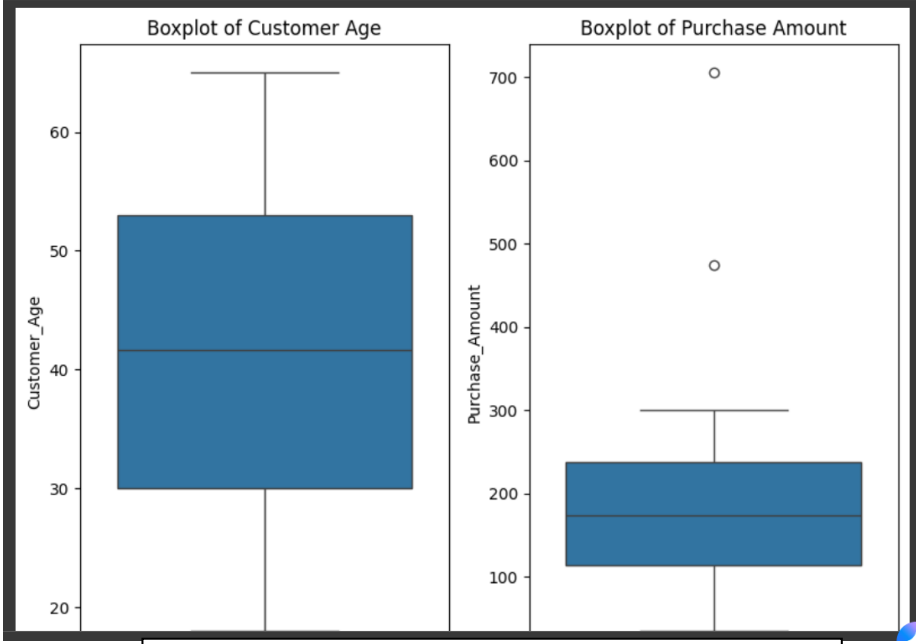
Cleaning data

```python
import matplotlib.pyplot as plt

plt.hist(luxx['Customer_Age'].dropna(), bins=10, color='red', edgecolor='black')
# Create a histogram of 'Customer_Age' values, ignoring NaNs, with 10 bins, red bars, and black edges

plt.xlabel('Customer Age')      # Label for x-axis
plt.ylabel('Frequency')         # Label for y-axis
plt.title('Distribution of Customer Age')  # Title of the plot

plt.show()  # Display the histogram
```



Histogram of customer age

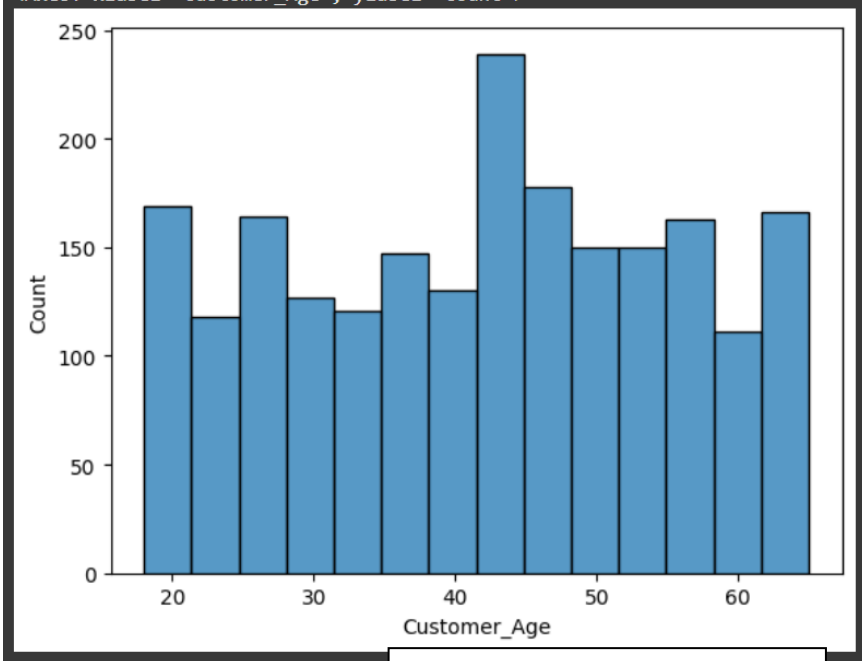
```python
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8,6))          # Set the figure size

plt.subplot(1, 2, 1)               # Create the first subplot in a 1x2 grid (left plot)
sns.boxplot(y=luxx['Customer_Age'].dropna())  # Boxplot for 'Customer_Age' ignoring NaNs
plt.title('Boxplot of Customer Age')           # Title for the first subplot

plt.subplot(1, 2, 2)               # Create the second subplot in a 1x2 grid (right plot)
sns.boxplot(y=luxx['Purchase_Amount'].dropna())  # Boxplot for 'Purchase_Amount' ignoring NaNs
plt.title('Boxplot of Purchase Amount')          # Title for the second subplot

plt.tight_layout()                 # Adjust subplots to prevent overlap
plt.show()                        # Display the plots
```



Boxplot for customer age and purchase amount

```
import seaborn as sns

sns.histplot(lux['Customer_Age'].dropna())  # Plot a histogram of 'Customer_Age', ignoring missing values
```

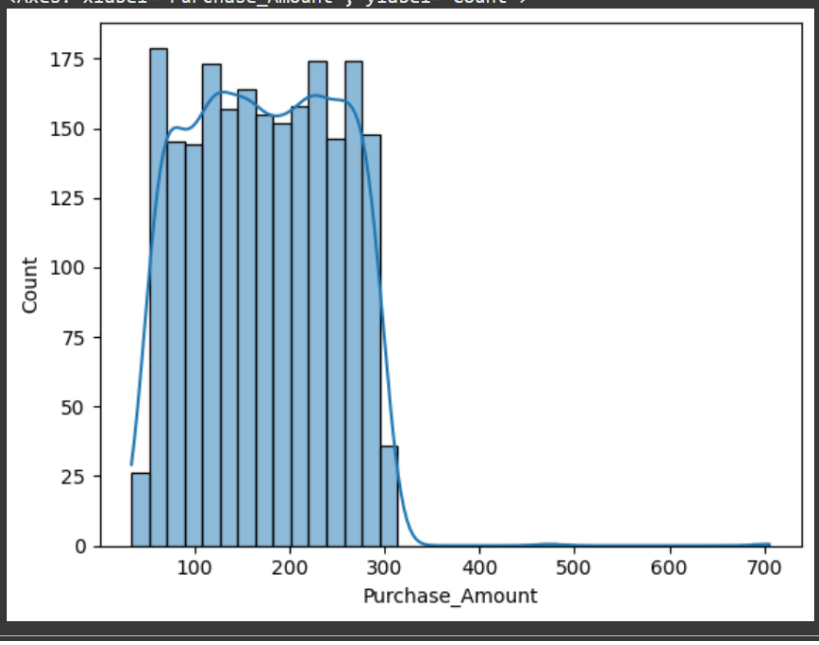
```
<Axes: xlabel='Customer_Age', ylabel='Count'>
```



Histplot for customer age

```
import seaborn as sns

sns.histplot(lux['Purchase_Amount'].dropna(), kde=True)
# Plot a histogram of 'Purchase_Amount' with a KDE (kernel density estimate) curve overlay,
# dropping any missing values to avoid errors.
```

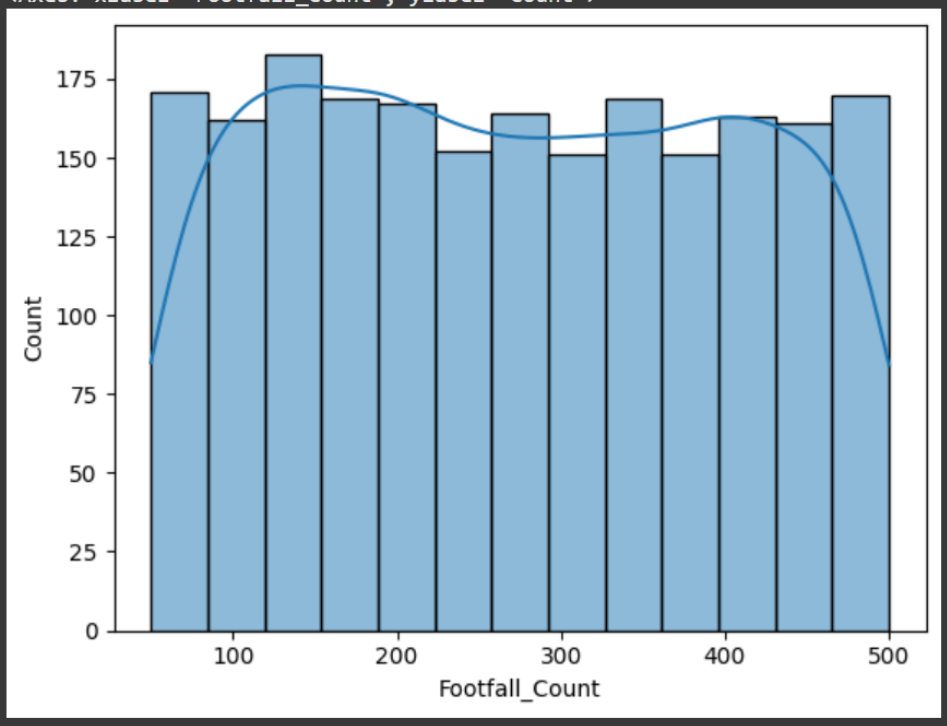
```
<Axes: xlabel='Purchase_Amount', ylabel='Count'>
```



Histplot for purchase amount

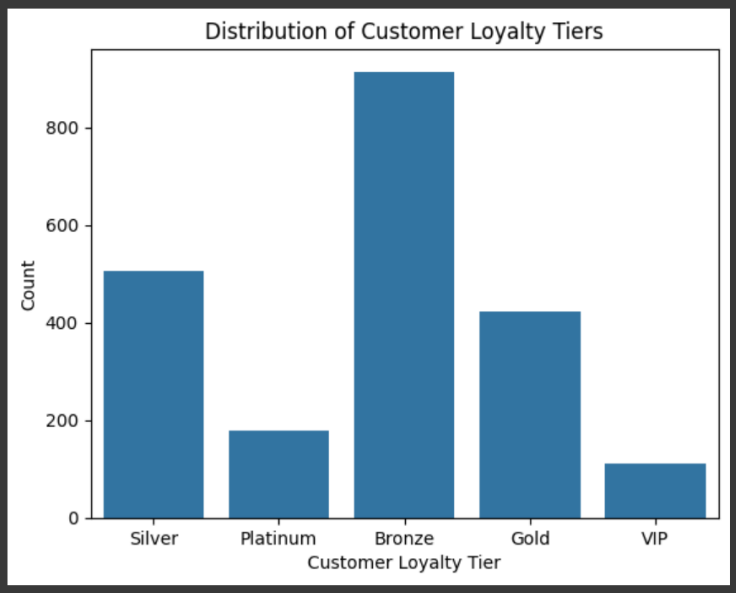
```
import seaborn as sns

sns.histplot(lux['Footfall_Count'].dropna(), kde=True)
# Plot histogram of 'Footfall_Count' with KDE curve, ignoring missing values
```

```
<Axes: xlabel='Footfall_Count', ylabel='Count'>
```



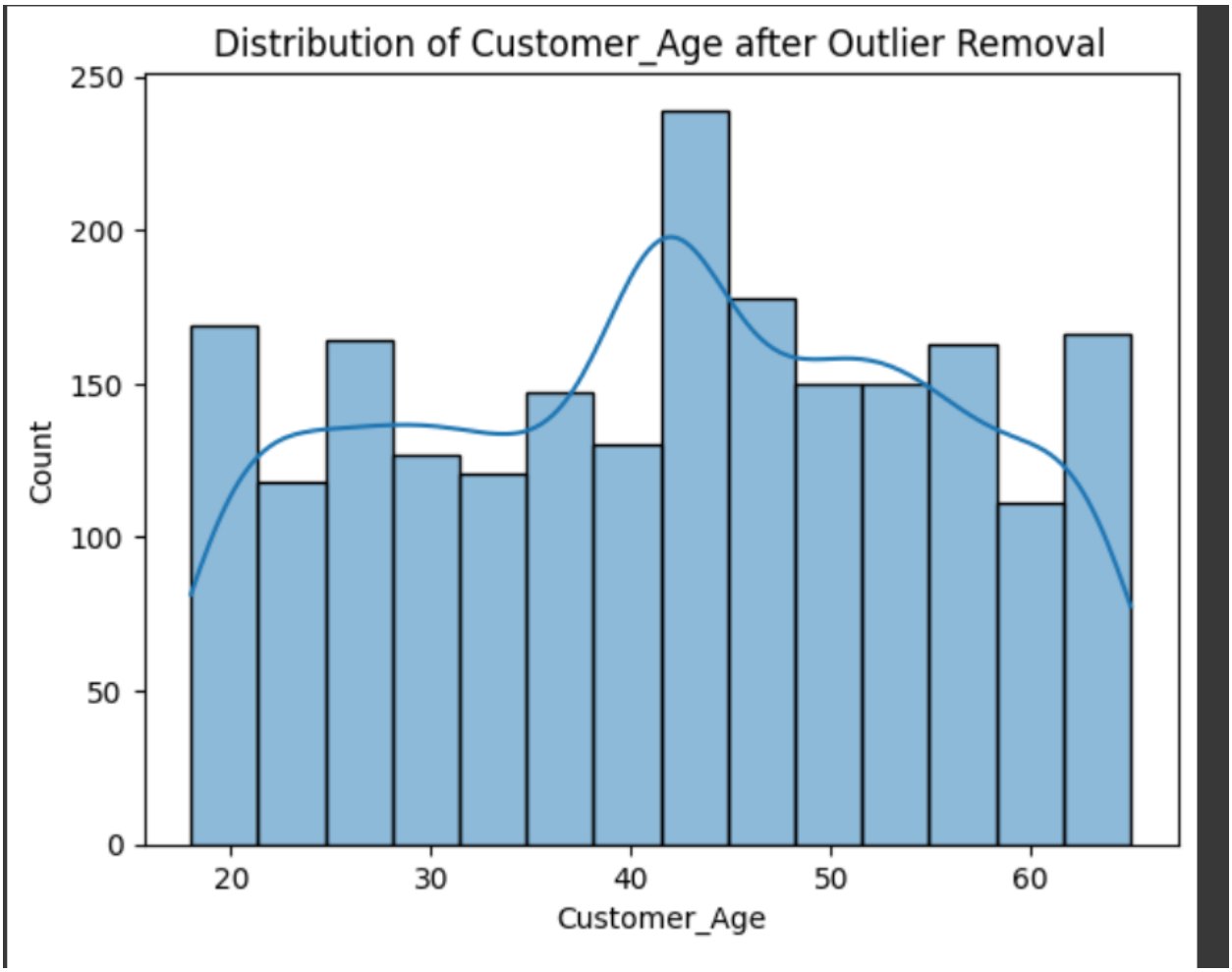
Histplot for Footfall count

```
import seaborn as sns

sns.countplot(data=lux, x='Customer_Loyalty_Tier')
# Plot count of each category in 'Customer_Loyalty_Tier'

plt.xlabel('Customer Loyalty Tier')
plt.ylabel('Count')
plt.title('Distribution of Customer Loyalty Tiers')
plt.show()
```

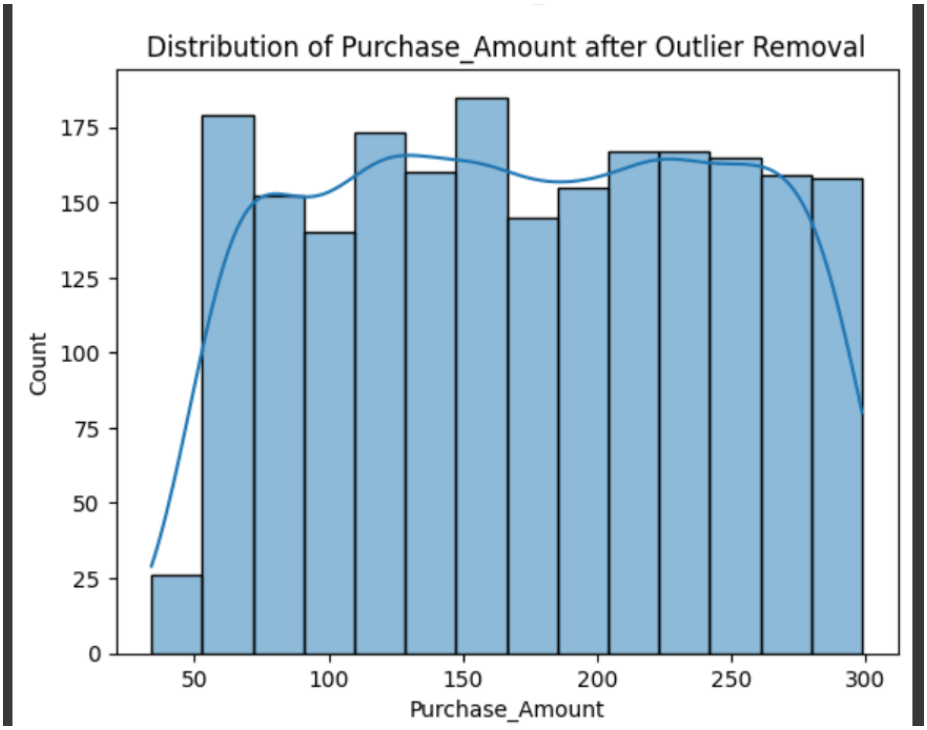


Customer loyalty tier graph

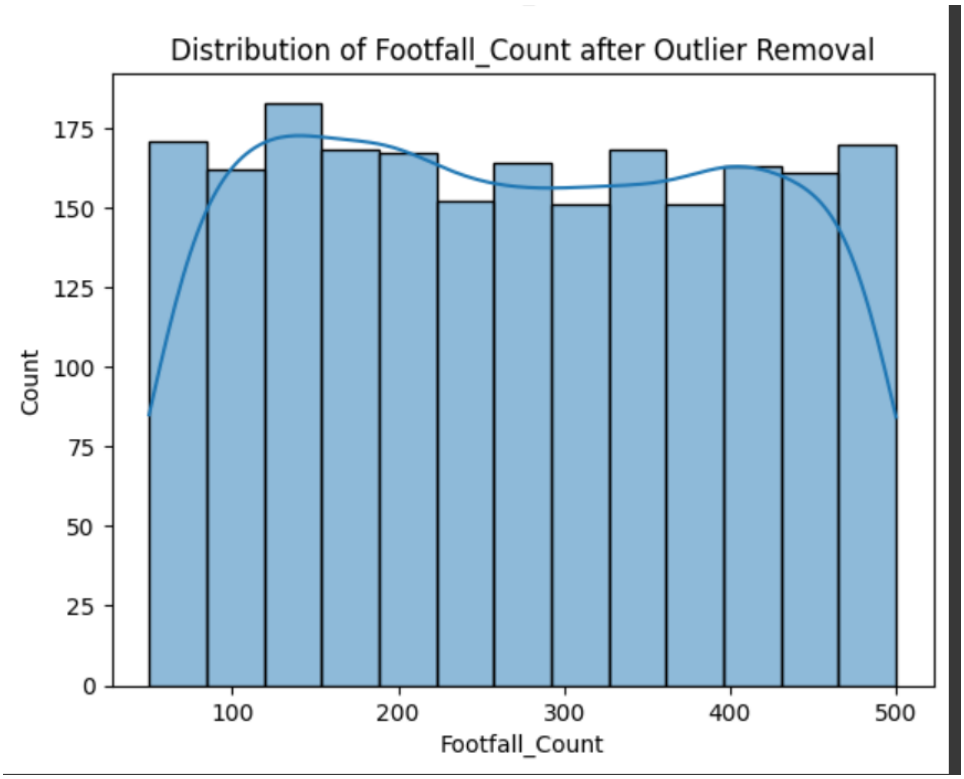
```python
import seaborn as sns
import matplotlib.pyplot as plt

def remove_outliers(lux, column):
    Q1 = lux[column].quantile(0.25)           # Calculate 1st quartile
    Q3 = lux[column].quantile(0.75)           # Calculate 3rd quartile
    IQR = Q3 - Q1                             # Interquartile Range
    lower_bound = Q1 - 1.5 * IQR              # Lower fence
    upper_bound = Q3 + 1.5 * IQR              # Upper fence
    lux = lux[(lux[column] >= lower_bound) & (lux[column] <= upper_bound)]  # Keep rows within fences
    return lux

col = ["Customer_Age", "Purchase_Amount", "Footfall_Count"]
for i in col:
    lux = remove_outliers(lux, i)             # Remove outliers in each column iteratively
    sns.histplot(lux[i], kde=True)             # Plot histogram with KDE after outlier removal
    plt.title(f'Distribution of {i} after Outlier Removal')
    plt.show()

updated = lux.describe()                      # Summary statistics of the cleaned DataFrame
print(updated)
```



Distribution for customer age



Distribution for purchase amount

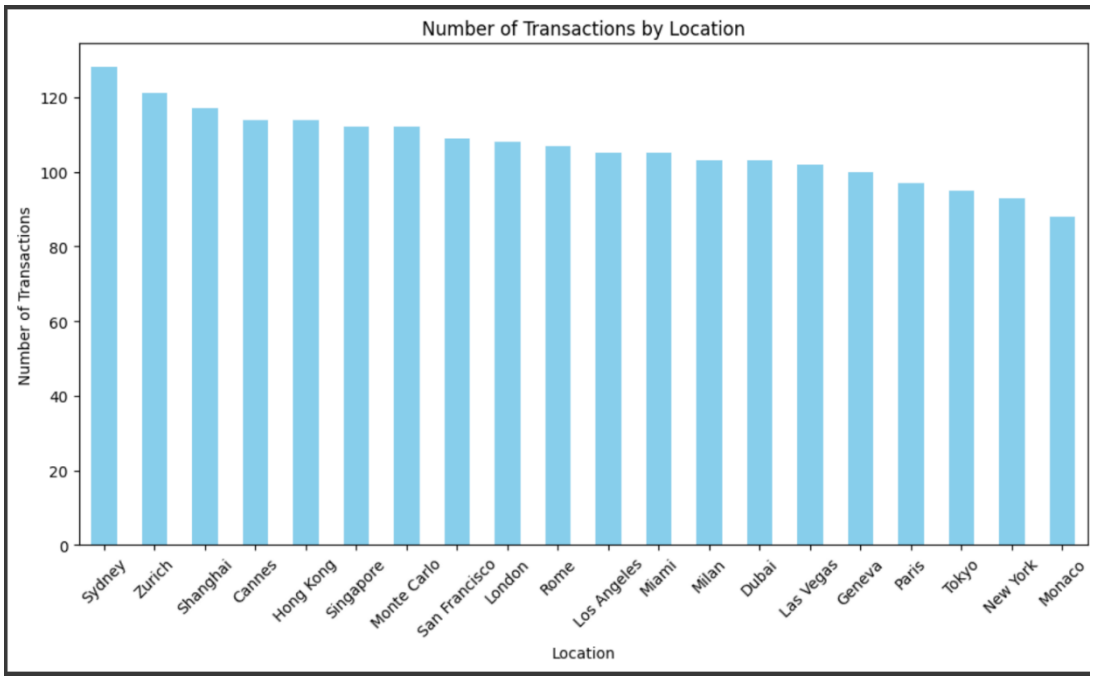


Distribution forFootfall count

	Customer_Age	Purchase_Amount	Fraud_Flag	Footfall_Count
count	2131.000000	2131.000000	2131.000000	2131.000000
mean	41.677545	174.036133	0.030971	272.477241
std	13.375419	72.375502	0.173281	131.152375
min	18.000000	34.000000	0.000000	50.000000
25%	30.000000	113.000000	0.000000	157.000000
50%	41.684262	174.000000	0.000000	269.000000
75%	53.000000	236.000000	0.000000	388.000000
max	65.000000	299.000000	1.000000	500.000000

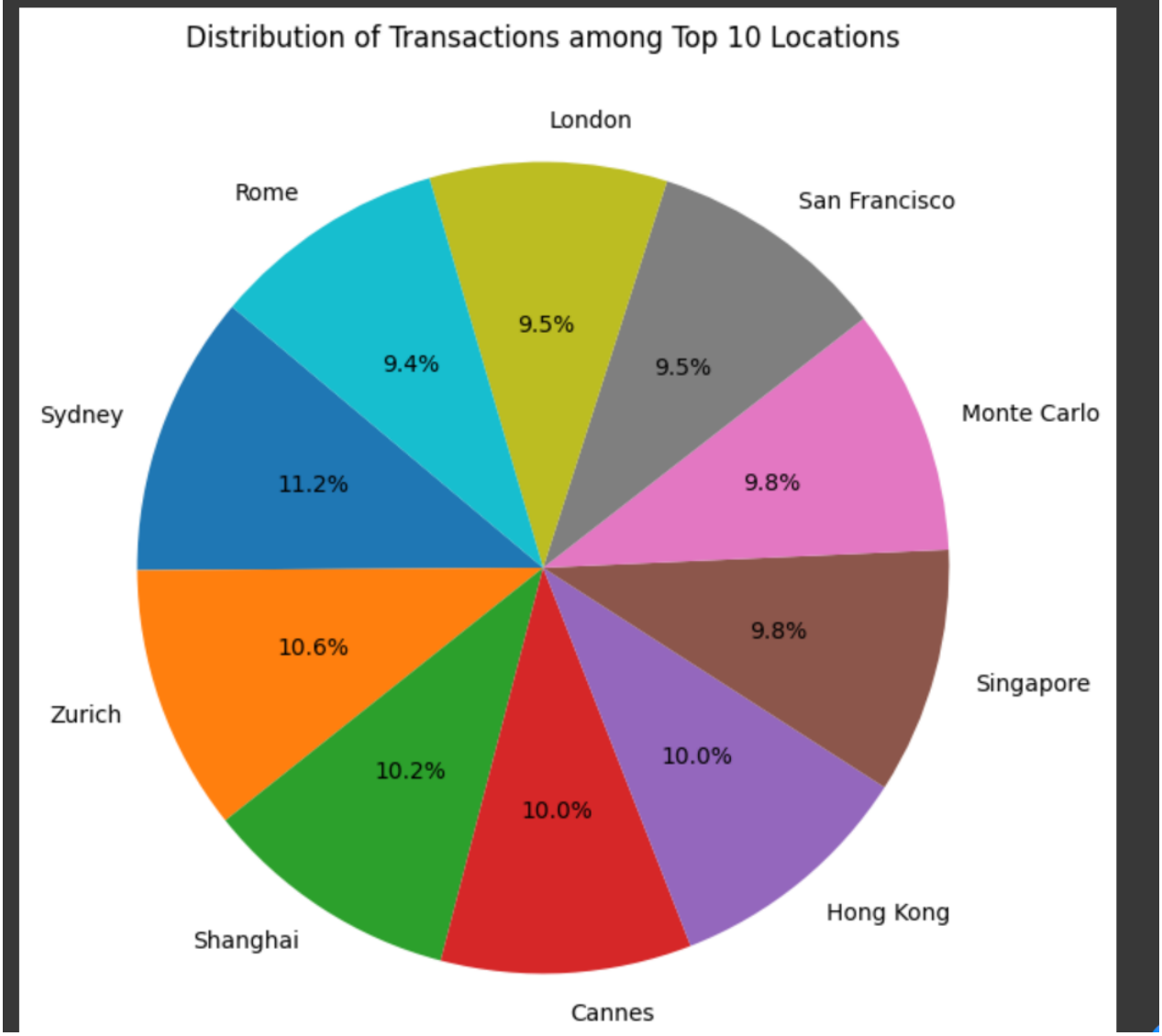
```python
import pandas as pd
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("luxury_cosmetics_fraud_analysis_2025.csv")

# Convert Transaction_Date to datetime
df['Transaction_Date'] = pd.to_datetime(df['Transaction_Date'], errors='coerce')
```

```python
# 1. Bar chart: Number of Transactions by Location
location_counts = df['Location'].value_counts()
plt.figure(figsize=(12,6))
location_counts.plot(kind='bar', color='skyblue')
plt.title("Number of Transactions by Location")
plt.xlabel("Location")
plt.ylabel("Number of Transactions")
plt.xticks(rotation=45)
plt.show()
```



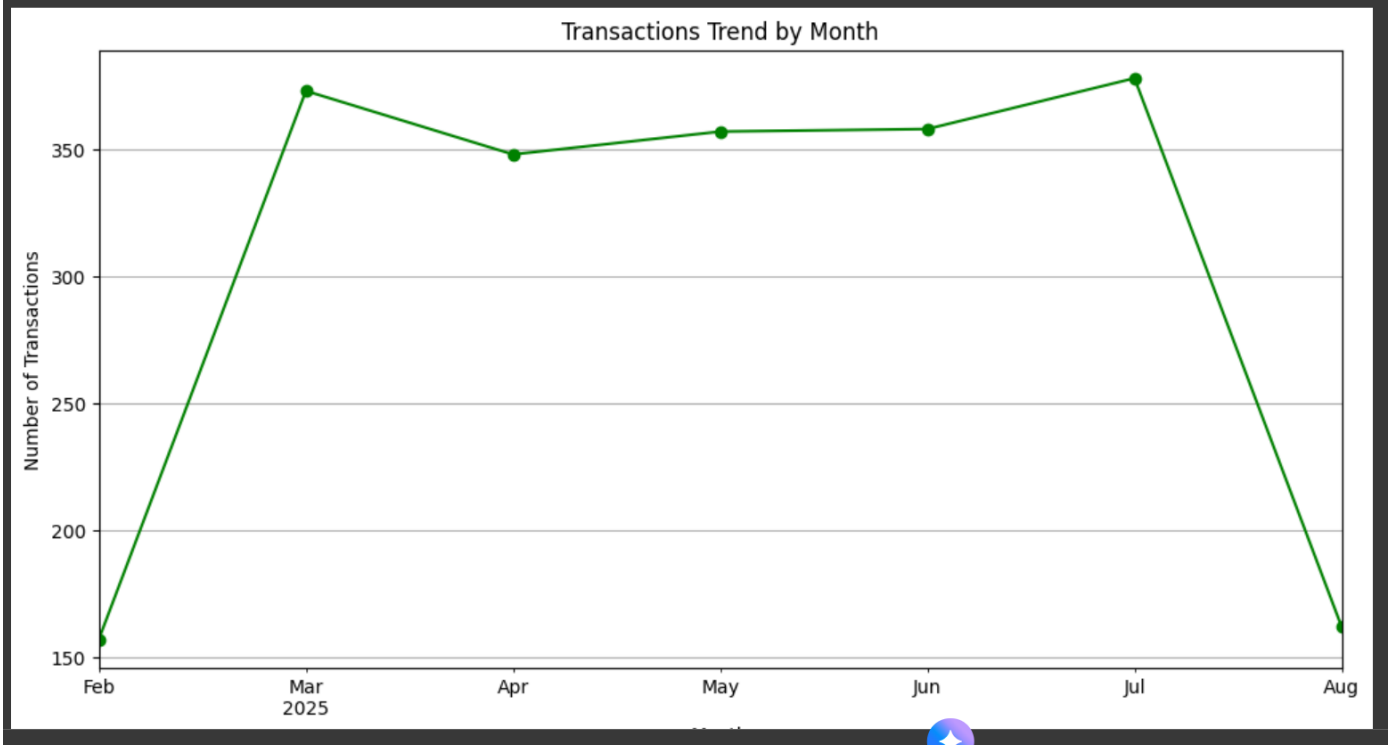
Graph for no of transactions by location

```
# 2. Pie chart: Distribution among Top 10 Locations
top10_locations = df['Location'].value_counts().head(10)
plt.figure(figsize=(8,8))
plt.pie(top10_locations, labels=top10_locations.index, autopct='%1.1f%%', startangle=140)
plt.title("Distribution of Transactions among Top 10 Locations")
plt.show()
```



Pie chart for Distribution among 10 locations

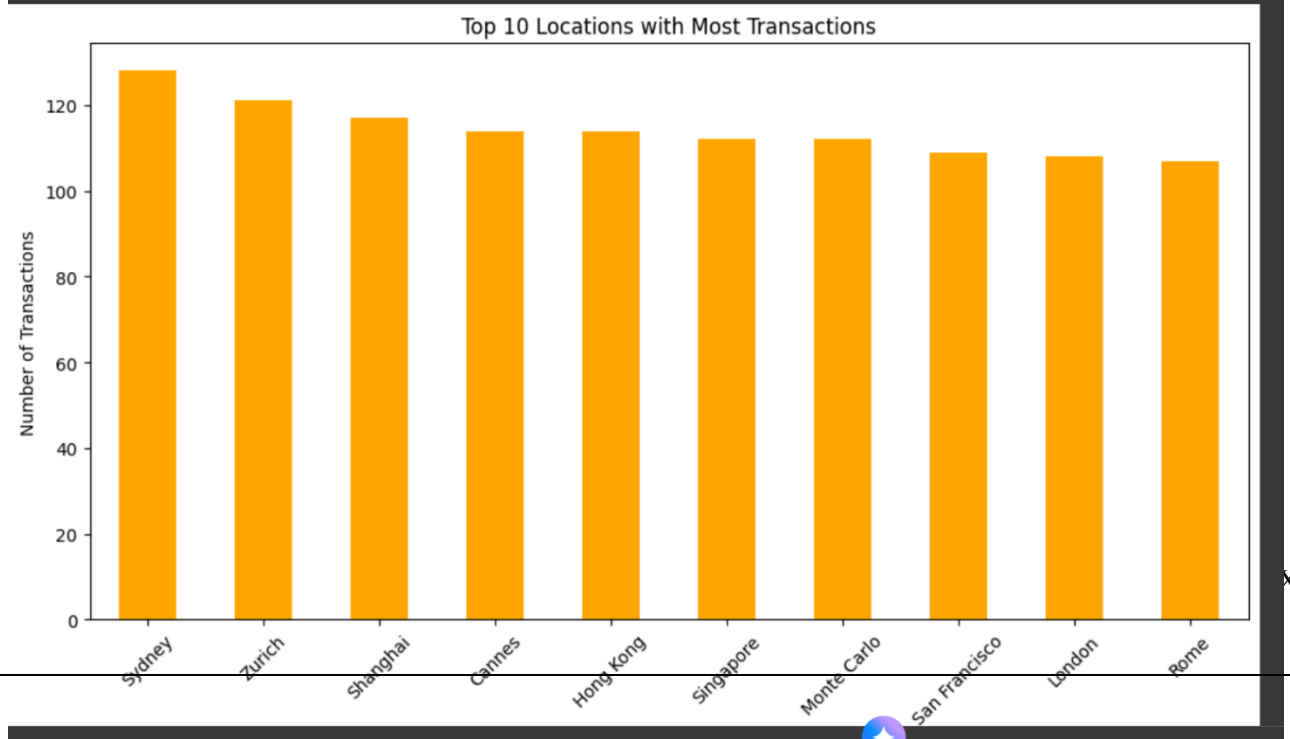
```python
# 3. Line chart: Transactions Trend by Month
df['Month'] = df['Transaction_Date'].dt.to_period('M')
month_counts = df['Month'].value_counts().sort_index()
plt.figure(figsize=(12,6))
month_counts.plot(kind='line', marker='o', color='green')
plt.title("Transactions Trend by Month")
plt.xlabel("Month")
plt.ylabel("Number of Transactions")
plt.grid(True)
plt.show()
```



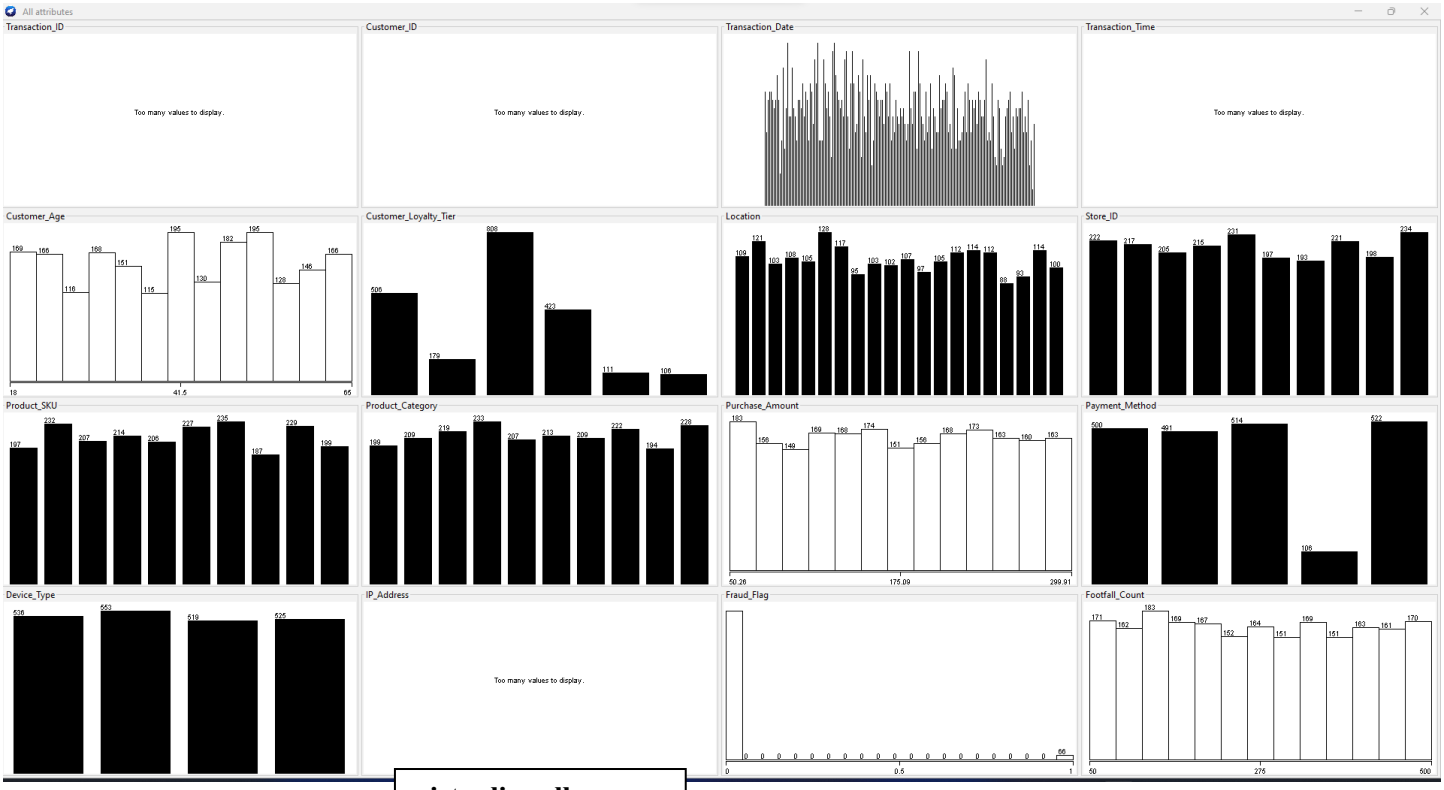
Line chart: Transactions Trend by Month

```python
# 4. Bar chart: Top 10 Locations with Most Transactions
plt.figure(figsize=(12,6))
top10_locations.plot(kind='bar', color='orange')
plt.title("Top 10 Locations with Most Transactions")
plt.xlabel("Location")
plt.ylabel("Number of Transactions")
plt.xticks(rotation=45)
plt.show()
```

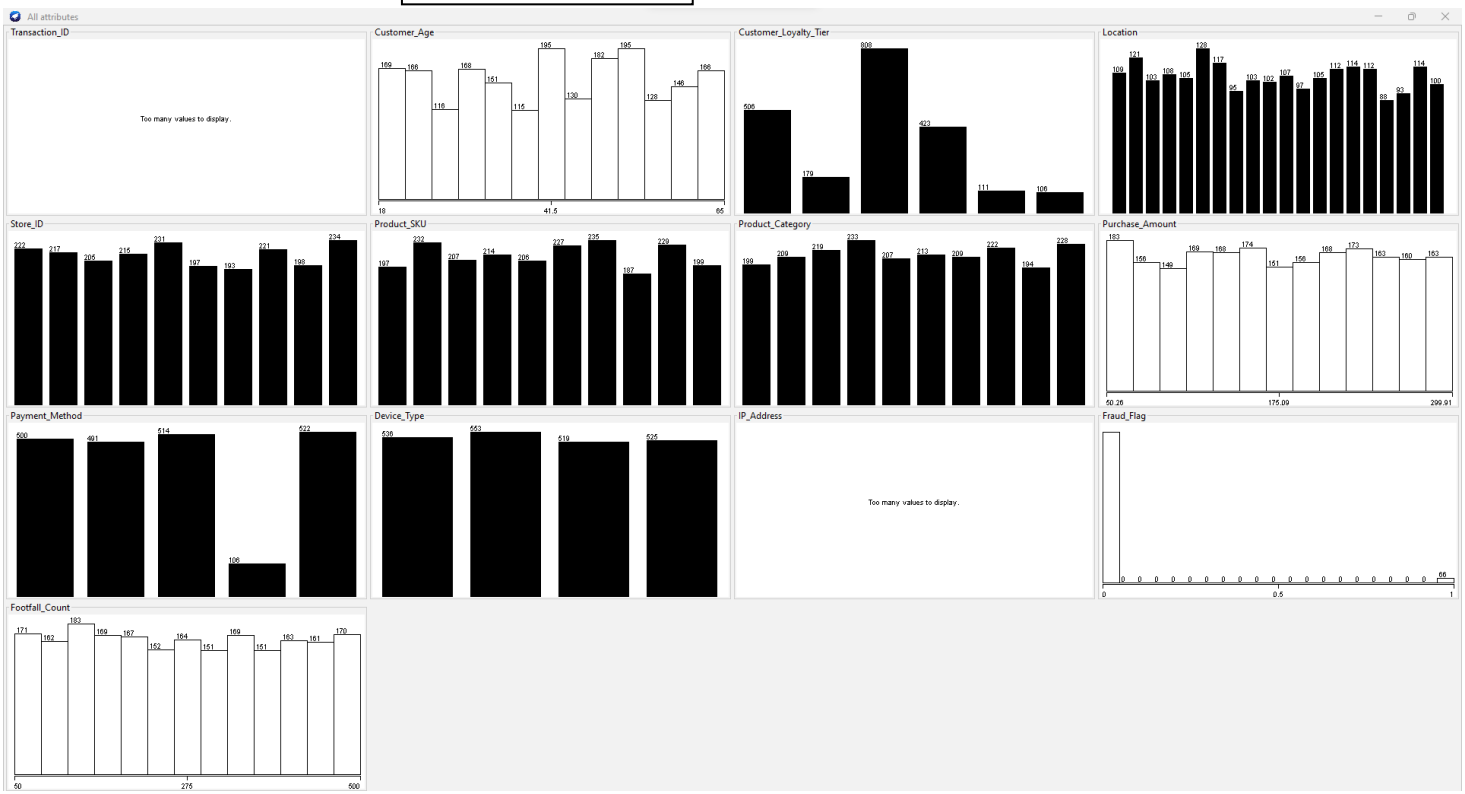
Bar chart: Top 10 Locations with Most



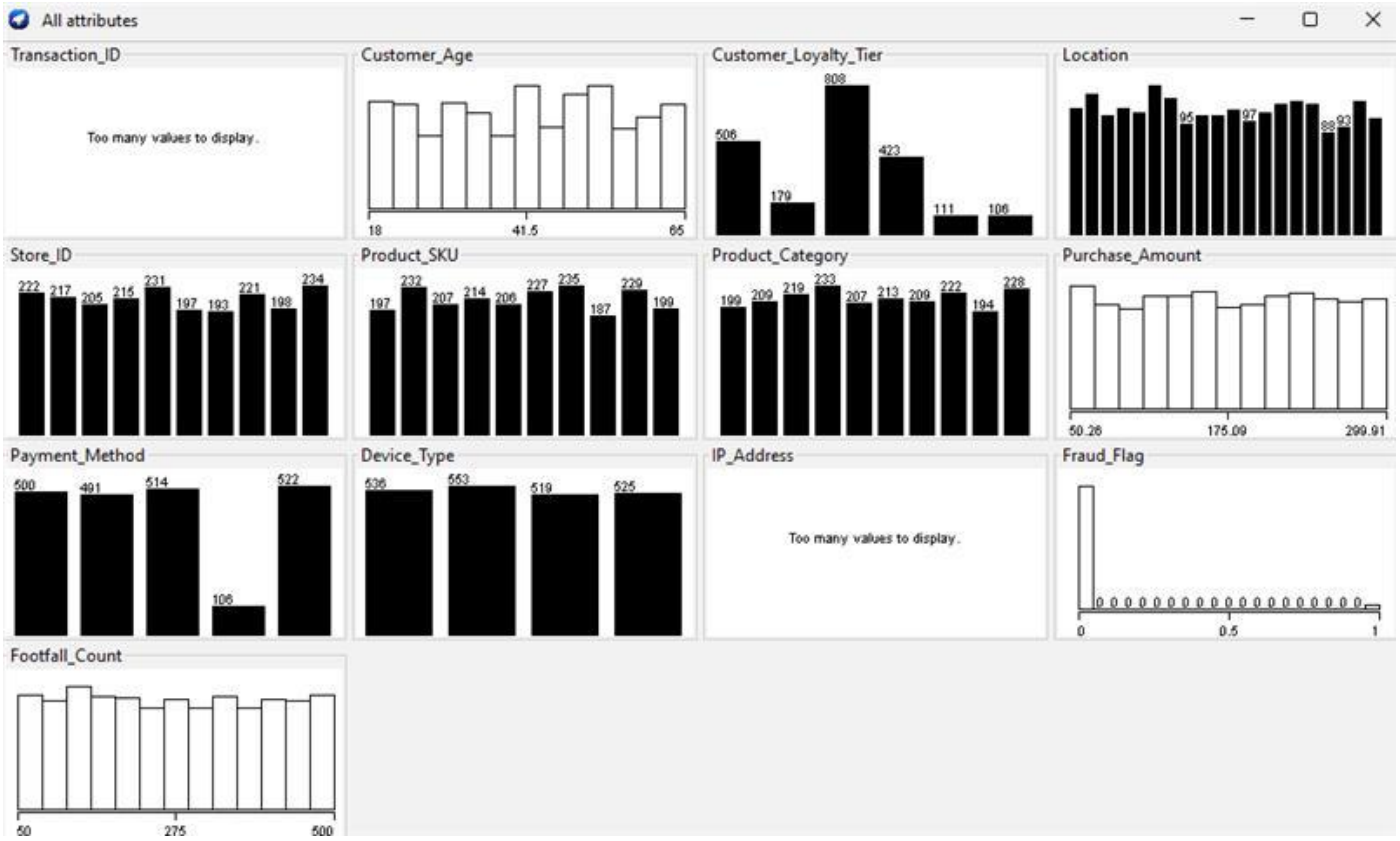
WEKA TOOL:



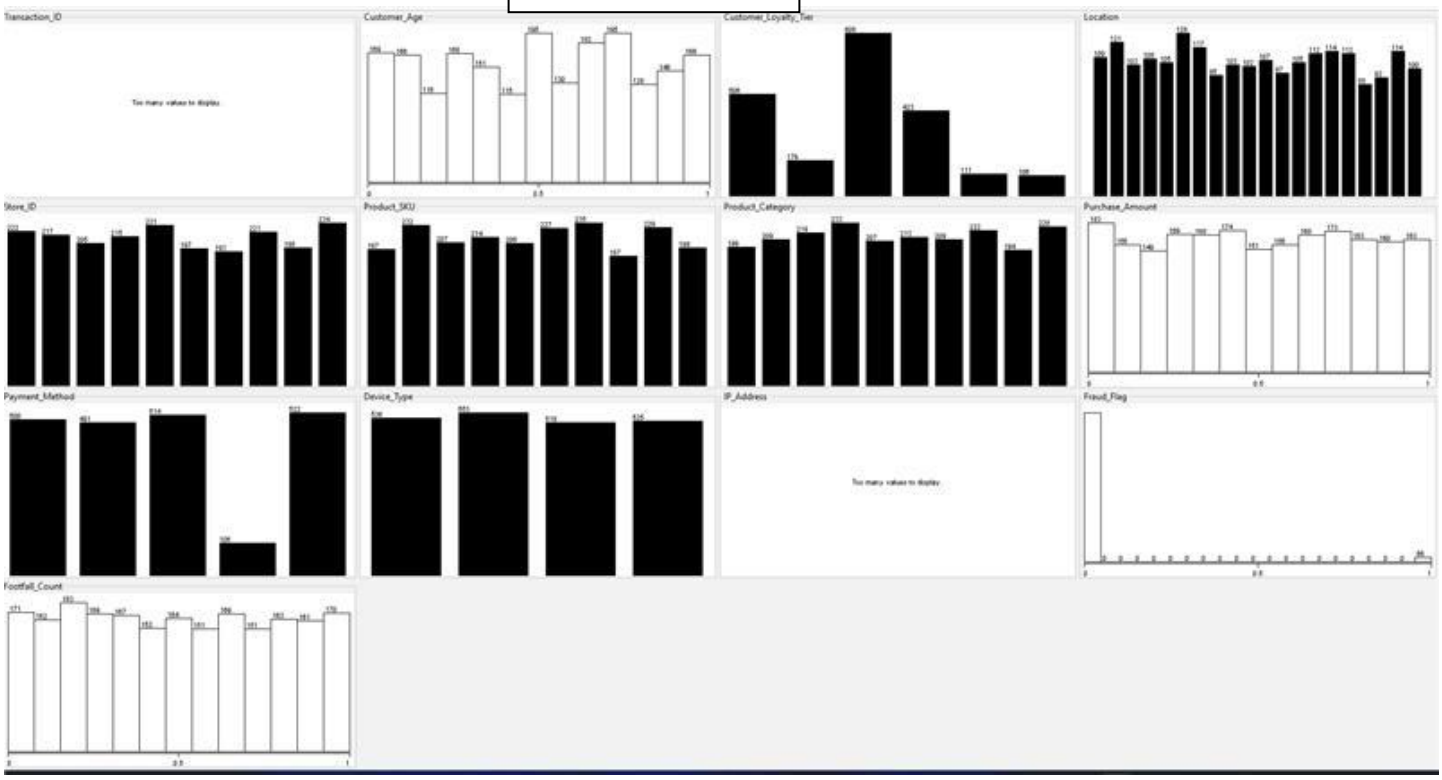
vistualize all



remove filter

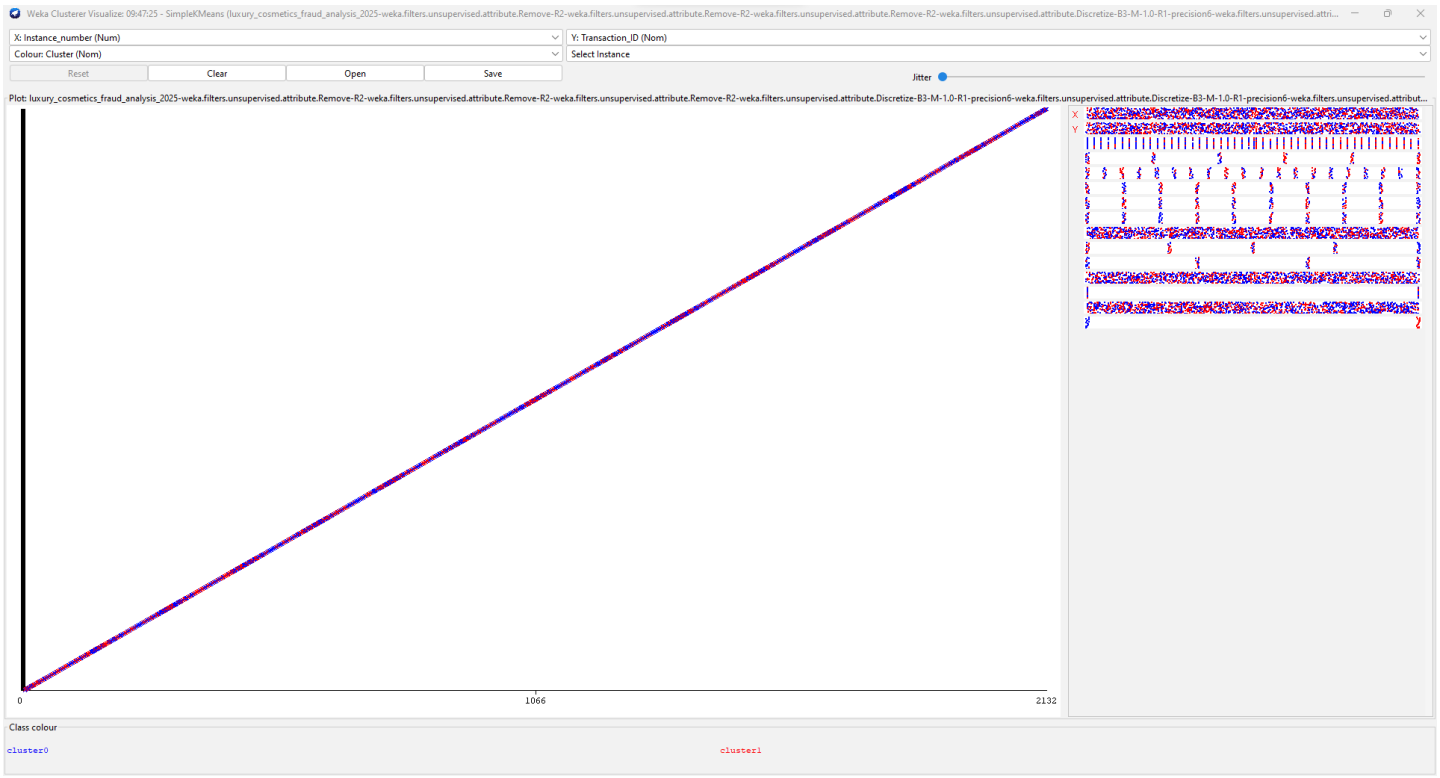


Discretize

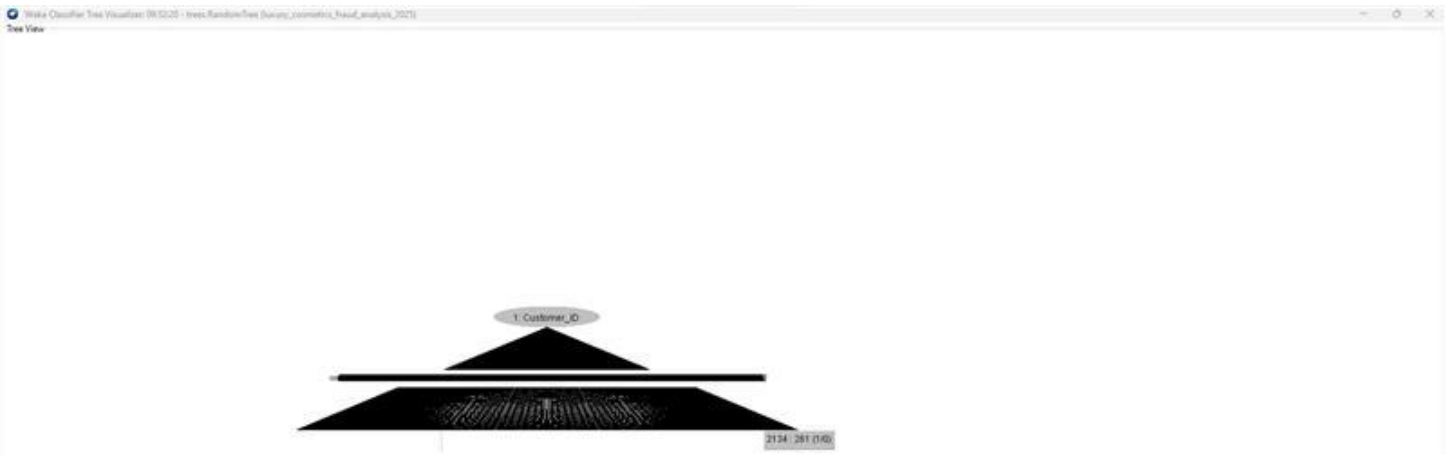


normalize

replace missing value

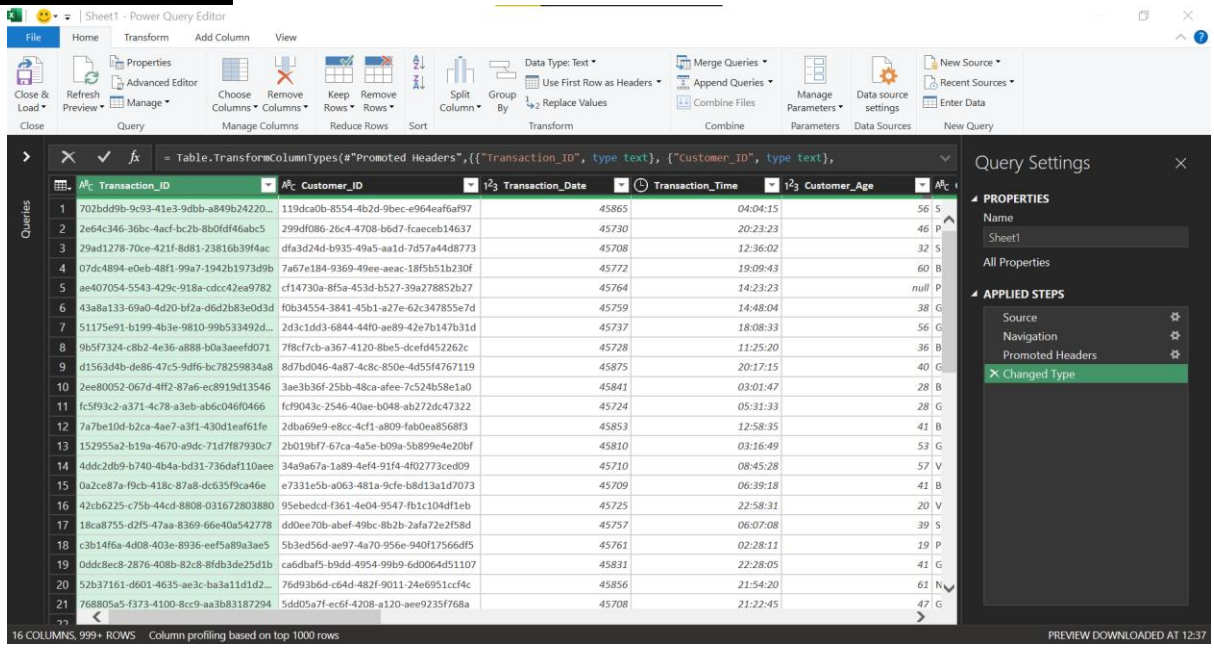


cluster

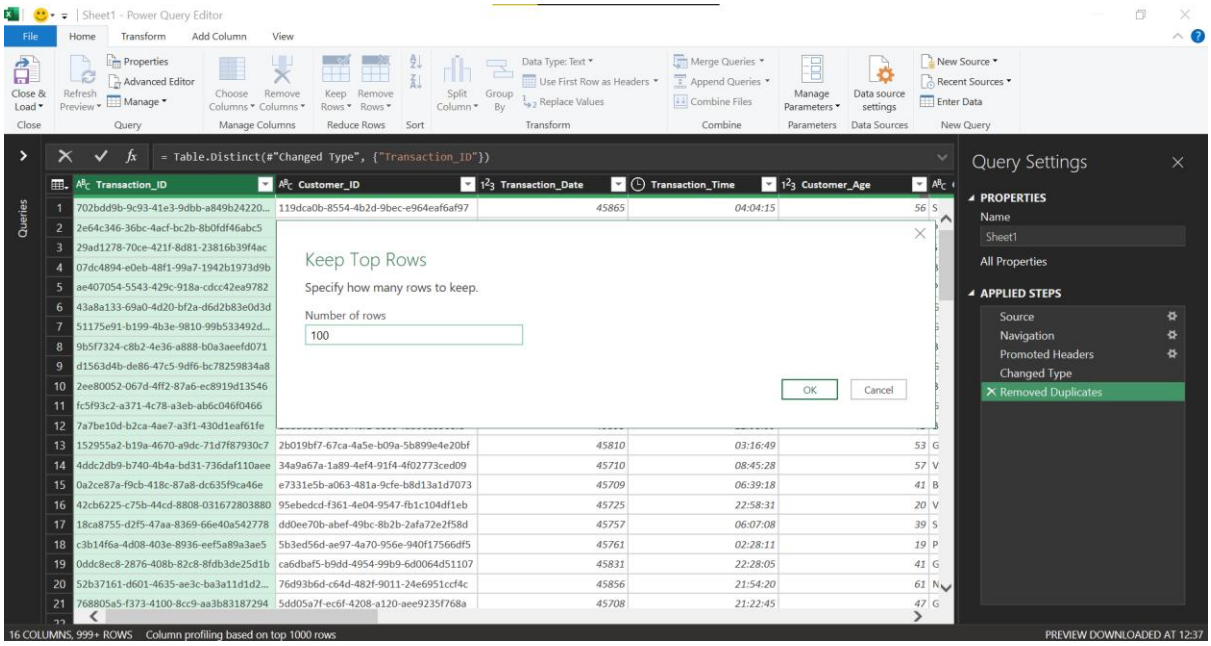


random tree

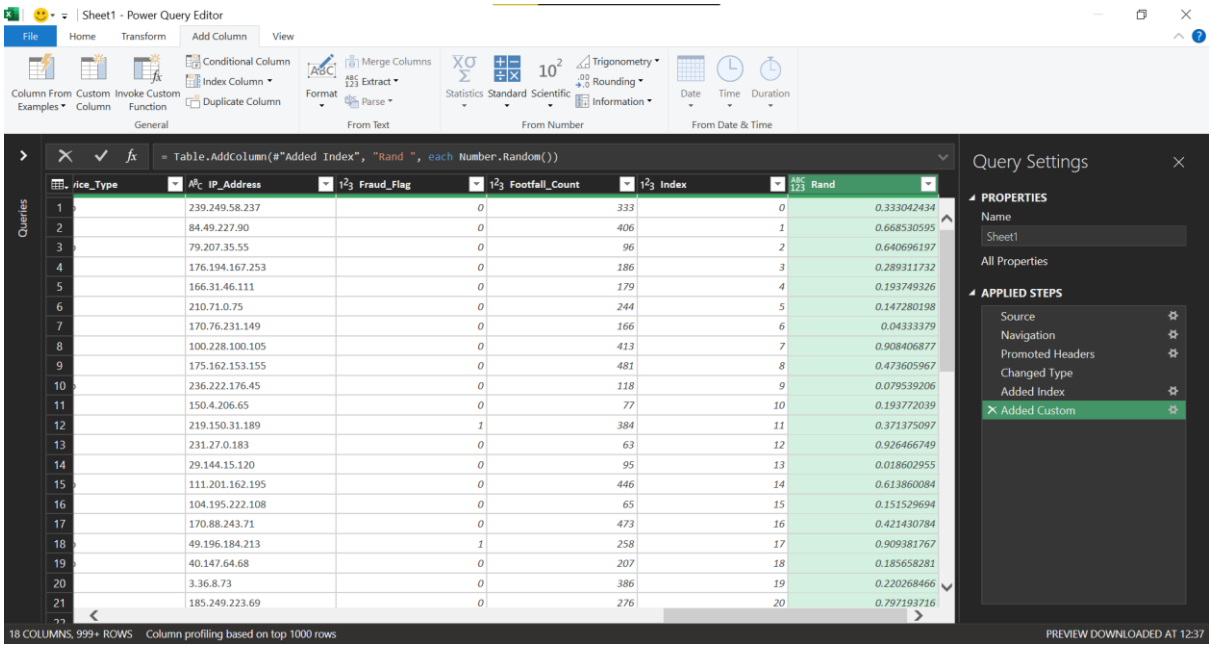
Power query:



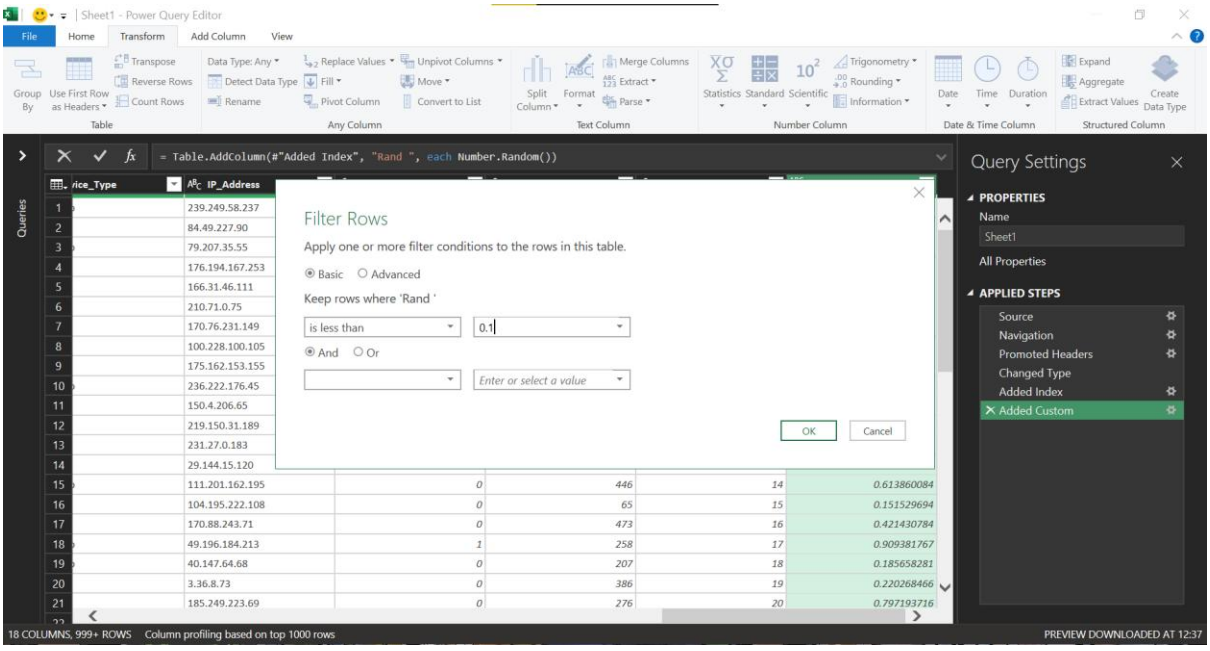
LOADING OF Data



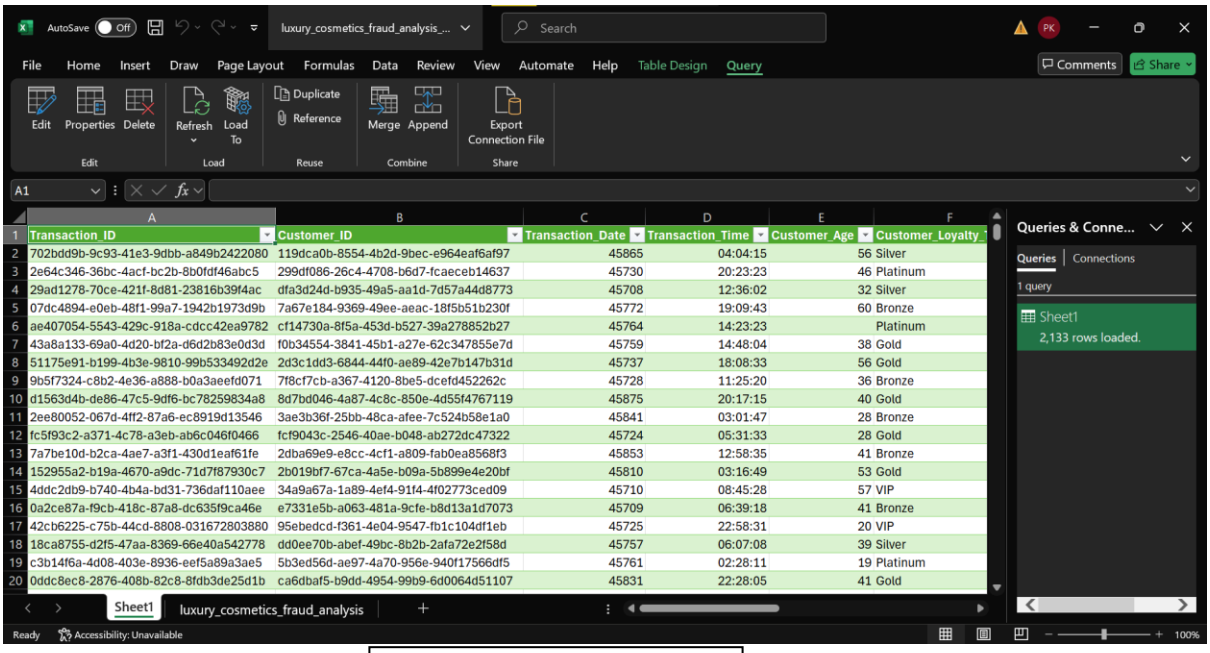
Sorting Data



TRANSFORMING DATA

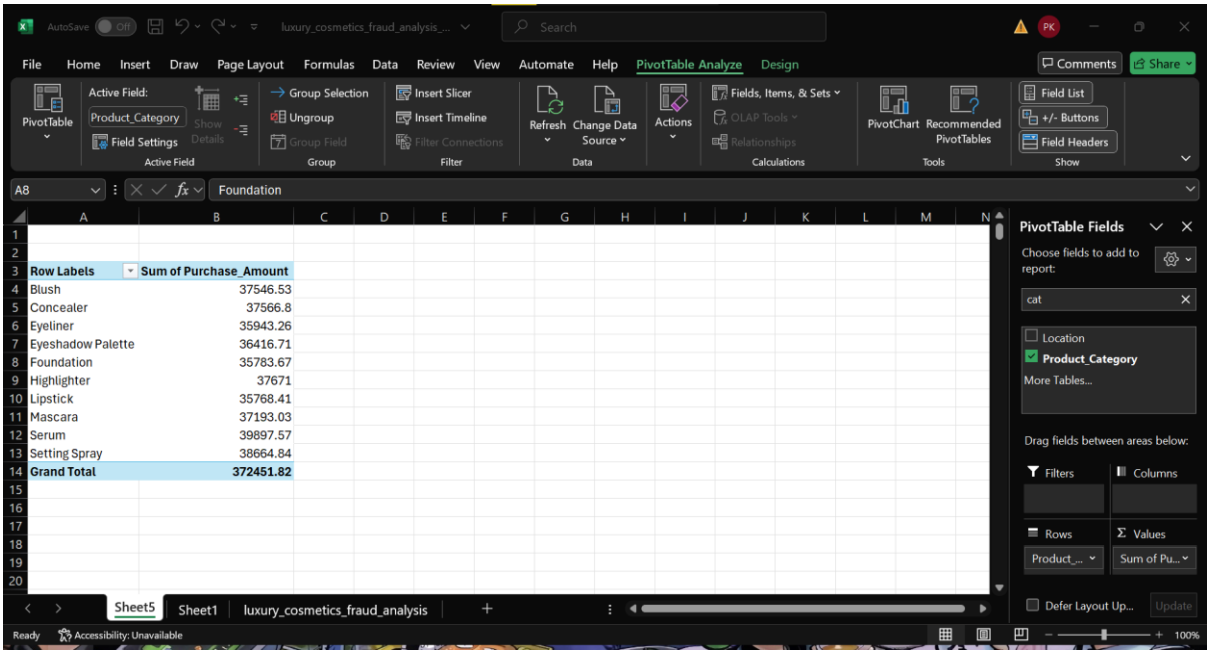


Cleaning Data



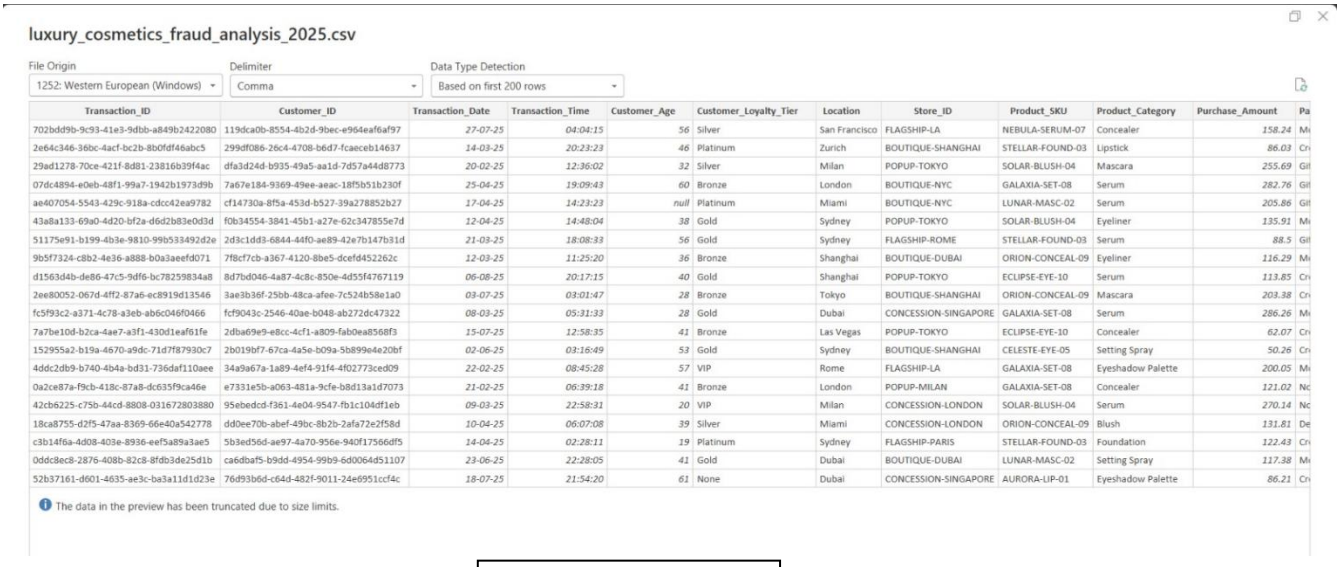
Transaction_ID	Customer_ID	Transaction_Date	Transaction_Time	Customer_Age	Customer_Loyalty
702bdd9b-9c93-41e3-9dbb-a849b2422080	119dca0b-8554-4b2d-9bec-e964eaf6af97	45865	04:04:15	56	Silver
2e64c346-36bc-4acf-bc2b-8b0fdf46abc5	299df086-26c4-4708-b6d7-fcaeceb14637	45730	20:23:23	46	Platinum
29ad1278-70ce-421f-8d81-23816b39f4ac	dfa3d24d-b935-49a5-aa1d-7d57a44d8773	45708	12:36:02	32	Silver
07dc4894-e0eb-48f1-99a7-1942b1973d9b	7a67e184-9369-49ee-aeac-18f5b51b230f	45772	19:09:43	60	Bronze
ae407054-5543-429c-918a-cdcc42ea9782	cf14730a-8f5a-453d-b527-39a278852b27	45764	14:23:23		Platinum
43a8a133-69a0-4d20-bf2a-d6d2b83e0d3d	f0b34554-3841-45b1-a27e-62c347855e7d	45759	14:48:04	38	Gold
51175e91-b199-4b3e-9810-99b533492d2e	2d3c1dd3-6844-44f0-ae89-42e7b147b31d	45737	18:08:33	56	Gold
9b5f7324-c8b2-4e36-a888-b0a3aeefd071	7f8cf7cb-a367-4120-8be5-dcefd452262c	45728	11:25:20	36	Bronze
d1563d4b-de86-47c5-9df6-bc78259834a8	8d7bd046-4a87-4c8c-850e-4d55f4767119	45875	20:17:15	40	Gold
2ee80052-067d-4ff2-87a6-ec8919d13546	3ae3b36f-25bb-48ca-afee-7c524b58e1a0	45841	03:01:47	28	Bronze
fc5f93c2-a371-4c78-a3eb-ab6c046f0466	fcf9043c-2546-40ae-b048-ab272dc47322	45724	05:31:33	28	Gold
7a7be10d-b2ca-4ae7-a3f1-430d1eaf61fe	2dba69e9-e8cc-4cf1-a809-fab0ea8568f3	45853	12:58:35	41	Bronze
152955a2-b19a-4670-a9dc-71d7f87930c7	2b019bf7-67ca-4a5e-b09a-5b899e4e20bf	45810	03:16:49	53	Gold
4ddc2db9-b740-4b4a-bd31-736daf110aee	34a9a67a-1a89-4ef4-91f4-4f02773ced09	45710	08:45:28	57	VIP
0a2ce87a-f9cb-418c-87a8-dc635f9ca46e	e7331e5b-a063-481a-9cfe-b8d13a1d7073	45709	06:39:18	41	Bronze
42cb6225-c75b-44cd-8808-031672803880	95ebedcd-f361-4e04-9547-fb1c104df1eb	45725	22:58:31	20	VIP
18ca8755-d2f5-47aa-8369-66e40a542778	dd0ee70b-abef-49bc-8b2b-2afa72e2f58d	45757	06:07:08	39	Silver
c3b14f6a-4d08-403e-8936-eef5a89a3ae5	5b3ed56d-ae97-4a70-956e-940f17566df5	45761	02:28:11	19	Platinum
0ddc8ec8-2876-408b-82c8-8fdb3de25d1b	ca6dbaf5-b9dd-4954-99b9-6d0064d51107	45831	22:28:05	41	Gold

Creating pivotchart

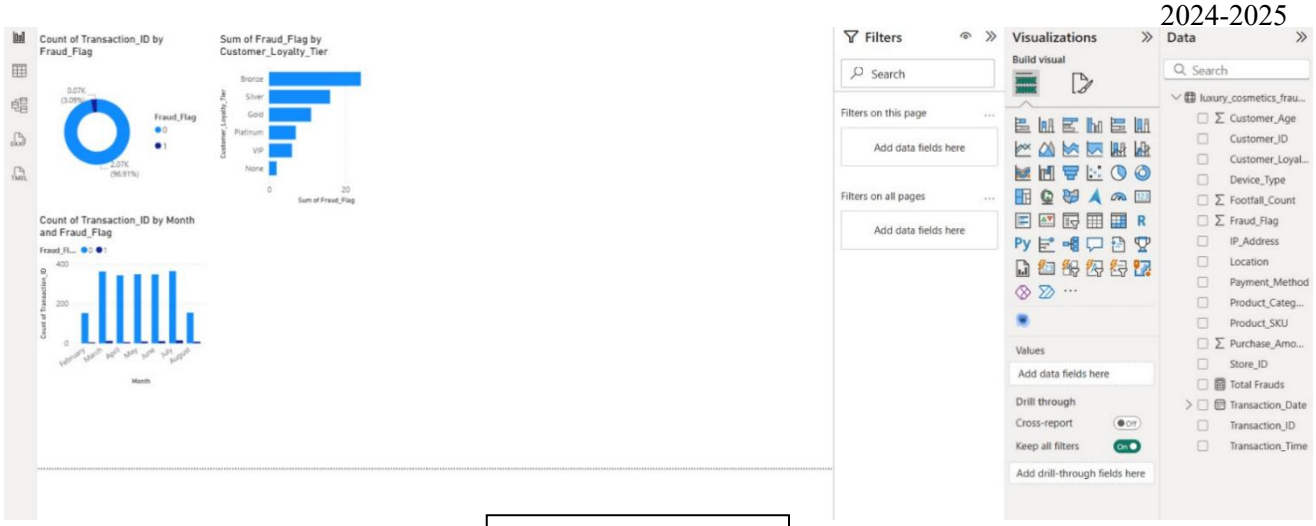


Sales analysis

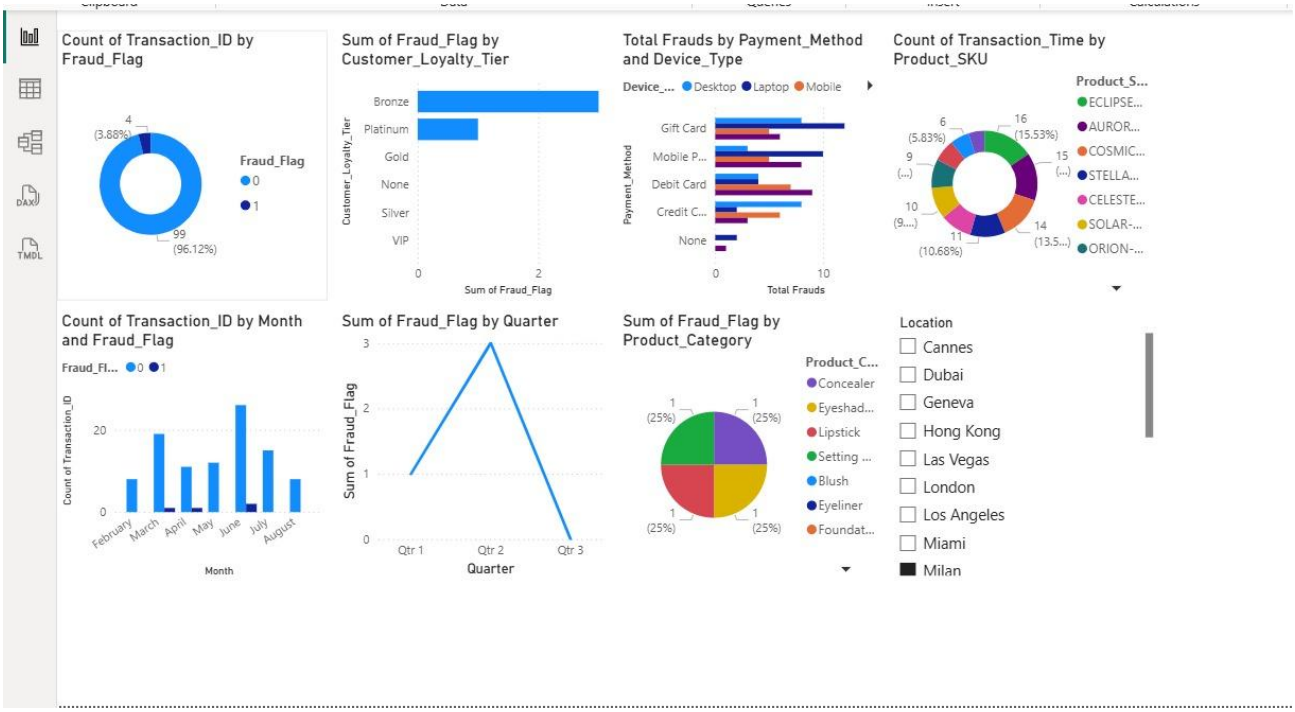
PowerBI



Loading of Data



Using slicer



Using bar plots and donut chart to identify major contributors

Overall Final Analysis:

The analysis of the Payment Card Fraud Detection dataset using descriptive statistics, data visualization, correlation analysis, and machine learning techniques such as classification and clustering provides comprehensive insights into transaction behavior and potential fraudulent patterns. Below is the breakdown of the analytical methods and their outcomes:

- Summary of Transaction Data

The initial step summarizes key numerical features such as transaction amount, customer age, and transaction frequency. Measures like mean, median, variance, and standard deviation highlight the spread of values and typical transaction behaviors.
This helps in identifying anomalies such as unusually high transaction amounts or outlier customers that may indicate fraud.

- Visualization of Transaction Distributions

Histograms and bar charts of numerical variables (like *Purchase_Amount* or *Customer_Age*) and categorical features (like *Payment_Method* or *Location*) reveal how data points are distributed.
Fraudulent transactions often appear in the tails of the distribution — e.g., high-value purchases or rare payment types (such as Gift Cards).

- Comparison of Fraud vs. Non-Fraud Transactions

Boxplots comparing fraudulent and non-fraudulent transactions provide a clear visual distinction between the two groups.
Fraudulent transactions often have a wider spread in purchase amount or occur disproportionately in certain locations or payment methods.

- Correlation Analysis

Correlation matrices reveal relationships among numerical variables (such as *Purchase_Amount*, *Customer_Age*, and *Transaction_Time*).
A low correlation among most features indicates independent factors, while moderate correlations can point toward influential predictors for fraud (e.g., high amounts and unusual payment methods).

- Feature Engineering and Transformation

By creating derived features — such as *Purchase_Amount_offer* or grading customer age into categories — the dataset becomes more interpretable for both statistical analysis and machine learning models.
These engineered variables often enhance fraud detection performance by representing hidden relationships in simpler terms.

- Dimensionality Reduction (PCA)

Principal Component Analysis (PCA) helps summarize the variability across all features, reducing noise and highlighting the most significant dimensions of the dataset.
Result: Most variance in transaction data tends to be captured in the first few components, indicating that a limited number of combined features explain the bulk of behavioral differences between fraudulent and legitimate transactions.

- Clustering Analysis

Using K-Means clustering, transactions are grouped into segments based on similarity in attributes such as *Purchase_Amount*, *Customer_Age*, and *Payment_Method*.
Result: Clusters often correspond to different transaction patterns — such as small legitimate payments, medium regular purchases, and high-value outliers that may correspond to fraud. This segmentation aids in identifying hidden fraud patterns not captured by explicit labels.

- Fraud Detection Insights

The combination of descriptive analytics and unsupervised learning reveals that fraudulent transactions typically deviate from the normal spending behavior in multiple dimensions: higher amounts, certain payment types, and unusual time or location distributions.
These findings support the development of predictive fraud detection models that can automatically flag suspicious transactions.

1.7 Outcomes

- Gained **hands-on experience** in implementing ETL (Extract, Transform, Load) processes to prepare the fraud detection dataset for analysis, ensuring clean and structured transactional data.
- Enhanced **data transformation and analysis skills** by performing data cleaning, visualization, and dimensionality reduction (PCA) to uncover key transaction trends and relationships.
- Applied **machine learning techniques**, including **K-Means clustering**, to identify hidden patterns in financial transactions and group similar behaviors for better fraud detection insights.
- Developed **understanding of structured ETL workflows**, demonstrating how organized data pipelines improve model accuracy and simplify fraud detection analysis.
- Recognized **the importance of ETL and analytics** in building efficient, data-driven systems that support secure decision-making in financial and e-commerce domain

1.8 Conclusion

This project demonstrates the effectiveness of the ETL (Extract, Transform, Load) process and data analytics techniques in analyzing the Payment Card Fraud Detection dataset using multiple tools — Jupyter Notebook, Power BI, Power Query, and WEKA. The workflow ensured systematic data extraction, cleaning, transformation, and visualization, resulting in a well-structured and reliable dataset for analysis.

In Python (Jupyter Notebook), data preprocessing, correlation analysis, PCA, and K-Means clustering helped uncover patterns and group transactions based on similarity. Power BI and Power Query enhanced data exploration and dashboard visualization, enabling clear insight into transaction trends and fraud distribution.

Finally, WEKA was used for machine learning model experimentation, validating the predictive capability of the dataset.

Overall, this integrated analytical approach proved how combining ETL workflows with visualization and machine learning tools supports efficient fraud detection, data-driven insights, and better decision-making in financial analytics.

1.9 References

1. **Kaggle (Payment Card Fraud Detection Dataset)**
   • Website: https://www.kaggle.com/datasets/pratyushpuri/payment-card-fraud-detection-with-ml-models-2025
   • Kaggle is a leading platform for machine learning datasets and projects. The Payment Card Fraud Detection dataset was sourced from here for analysis and modeling.

2. **Python Official Documentation**
   • Website: https://docs.python.org
   • The official Python documentation provides references for data handling, programming syntax, and integration with analytical libraries such as pandas, NumPy, and matplotlib.

3. **pandas Documentation**
   • Website: https://pandas.pydata.org/docs/
   • pandas is the core library used in the Jupyter Notebook for data cleaning, transformation, and preprocessing during the ETL process.

4. **Power BI Documentation**
   • Website: https://learn.microsoft.com/en-us/power-bi/
   • Power BI was used to visualize transaction data and fraud patterns through interactive dashboards for clearer interpretation and reporting.

5. **Microsoft Power Query**
   • Website: https://learn.microsoft.com/en-us/power-query/
   • Power Query enabled efficient data extraction, transformation, and loading (ETL) before analysis and visualization in Power BI.

6. **WEKA (Waikato Environment for Knowledge Analysis)**
   • Website: https://www.cs.waikato.ac.nz/ml/weka/
   • WEKA was used for applying and validating machine learning models, offering a GUI-based environment for classification and clustering experiments.