# PROJECT REPORT
# CREDIT CARD FRAUD DETECTION

## Introduction

### Overview of the Credit Card Fraud

Credit card fraud is a deliberate act of unauthorized use of someone else's credit card information to obtain goods, services, or funds. With the rapid growth of digital transactions, fraudsters are employing increasingly sophisticated techniques to exploit vulnerabilities in payment systems. This poses significant risks to individuals and financial institutions, resulting in financial losses, reputational damage, and a lack of trust in electronic payment systems.

Detecting fraud is not just about finding fraudulent transactions—it is also about ensuring that genuine transactions are not mistakenly flagged as fraud and ensuring a seamless experience for genuine customers. Misclassifying legitimate transactions as fraudulent (false positives) can cause frustration and loss of trust, while failing to detect fraud (false negatives) can result in substantial financial harm.

### Key Factors for Evaluating Credit Card Fraud

Identifying credit card fraud requires the analysis of various transactional patterns and behavioural cues. Key factors for evaluation include:

1. **Transaction Amount:** Transactions with values that significantly deviate from a user's usual spending behaviour.
2. **Transaction Location:** Purchases made in geographic areas far from the user's normal locations.
3. **Time of Transaction:** Unusual transaction times, such as late at night, outside the user's typical behaviour.
4. **Frequency of Transactions:** A rapid increase in the number of transactions in a short timeframe may indicate fraud.
5. **Merchant Category:** Purchases from unfamiliar or high-risk merchants can signal fraudulent activity.

These are some of the factors that are analysed collectively to identify anomalies and patterns that distinguish legitimate transactions from fraudulent ones.

### Challenges in Detecting Credit Card Fraud

Fraud detection systems face several challenges that complicate the process of distinguishing genuine transactions from fraudulent ones:

1. **Imbalanced Data:** Fraudulent transactions represent a small fraction of total transactions, making it difficult to train models that can effectively detect these rare events.
2. **Dynamic Fraud Strategies:** Fraudsters continuously evolve their techniques, requiring systems to adapt in real-time.
3. **False Positives:** Transactions flagged as fraudulent but are actually legitimate can frustrate customers and damage trust in financial institutions.

4. **Data Privacy and Security:** Ensuring the privacy and security of sensitive user data while analyzing it poses ethical and regulatory challenges.
5. **Cost of Misclassification:** While undetected fraud can lead to substantial losses, falsely labelling genuine transactions as fraud can harm the customer experience and loyalty.

Effective fraud detection systems must navigate these challenges by leveraging advanced machine learning algorithms and continually updating their models to stay ahead of emerging threats. This project aims to address these challenges and develop a solution that achieves both security and user satisfaction.

## Objective

The objective of this project is to develop an effective system for detecting fraudulent credit card transactions using machine learning. By analyzing transactional data and identifying patterns that distinguish fraud from legitimate activities, the project aims to build and evaluate various machine learning models to achieve high accuracy while minimizing false positives.

This includes addressing challenges such as data imbalance, evolving fraud strategies, and the trade-off between precision and recall. The project further seeks to compare the performance of these models using key metrics to identify the best-performing algorithm for fraud detection. Ultimately, the goal is to enhance the reliability of fraud detection systems, improve customer trust, and contribute to safer digital payment ecosystems.

## About the Dataset

The **Credit Card Dataset** used in this project provides detailed information about 100,000 credit card transactions. It is specifically designed to identify fraudulent activity and includes various features capturing transactional and demographic data. Below are the key aspects of the dataset:

**Key Features:**

1. **Size and Structure**:
    - Total Rows: 100,000
    - Total Columns: 16

**Column Name Description**

| Column Name | Description |
|---|---|
| **Transaction ID** | Unique identifier for each transaction. |
| **Date and Time** | Details of when the transaction occurred. |
| **Type of Card** | Indicates the type of credit card (e.g., Visa, MasterCard). |
| **Entry Mode** | How the card was used (e.g., Tap, PIN, CVC). |
| **Amount** | The monetary value of the transaction. |
| **Type of Transaction** | Whether the transaction was POS (Point of Sale) or Online. |
| **Merchant Group** | Category of the merchant (e.g., Entertainment, Electronics). |
| **Country of Transaction** | The country where the transaction occurred. |
| **Demographics** | Includes fields like **gender, age, country of residence, and bank information.** |
| **Fraud** | The target variable indicating whether the transaction is legitimate (0) or fraudulent (1). |

**Dataset Characteristics**

- The dataset combines transactional, geographic, and demographic features, making it suitable for analysing fraud patterns.
- It provides opportunities to explore correlations between fraud and various features, such as transaction type, amount, and demographics.
- The imbalance in the target variable reflects a common challenge in fraud detection, necessitating techniques like resampling or class-weight adjustments.

This dataset forms the foundation for building and evaluating machine learning models to detect fraudulent transactions effectively.

## Tools and Libraries Used

**Programming Language:** Python

To analyze and train the predictive model, we used Python and its powerful libraries. These libraries made it easier to manipulate, visualize, and train the dataset. Some of the key libraries used include:

- **Pandas:** For data manipulation and transformation.
- **NumPy:** For numerical computations.
- **Matplotlib and Seaborn:** For visualizing data trends and relationships.
- **Scikit-learn:** For training and evaluating the machine learning model.

These tools allowed us to uncover meaningful insights and build an accurate prediction model efficiently.

## MACHINE LEARNING MODELS USED

A Machine Learning (ML) model is a computational algorithm designed to identify patterns and make decisions or predictions based on data. These models learn from historical data during the training phase and use that knowledge to analyze new, unseen data.

In the context of supervised learning, where the goal is to predict a specific outcome, classifiers are a type of ML model used for categorizing data into predefined classes. For example, in this project, classifiers are used to distinguish between fraudulent and legitimate transactions.

Classifiers work by identifying decision boundaries within the data, leveraging mathematical and statistical techniques to separate different categories effectively. By evaluating their performance using metrics such as accuracy, precision, and recall, classifiers enable the development of robust and reliable predictive systems.

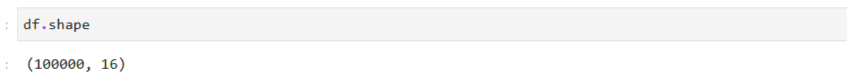
The following machine learning models were implemented to detect fraudulent transactions and evaluate their performance:

- **Random Forest**: An ensemble technique combining multiple decision trees to improve accuracy and reduce overfitting.
- **Gradient Boosting**: A boosting method that sequentially builds models to minimize errors and enhance predictive performance.
- **Support Vector Machine (SVM)**: A powerful algorithm that finds the optimal decision boundary for classifying data.
- **Naive Bayes:** A probabilistic classifier based on Bayes' Theorem, assuming independence among features, known for its simplicity and effectiveness, especially in text classification and other domains.
- **K-Nearest Neighbors (KNN)**: A distance-based algorithm that classifies transactions based on their similarity to nearby points.

These models were evaluated to determine the most effective one for detecting credit card fraud while addressing challenges like data imbalance and minimizing false positives

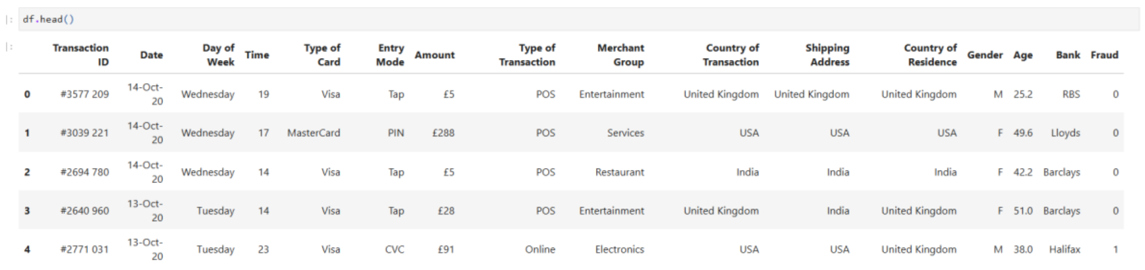
## EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is the process of analyzing and visualizing datasets to summarize their main characteristics and uncover patterns, relationships, or anomalies. It helps in understanding the structure of the data and preparing it for further modeling and decision-making.



```
df.shape
```

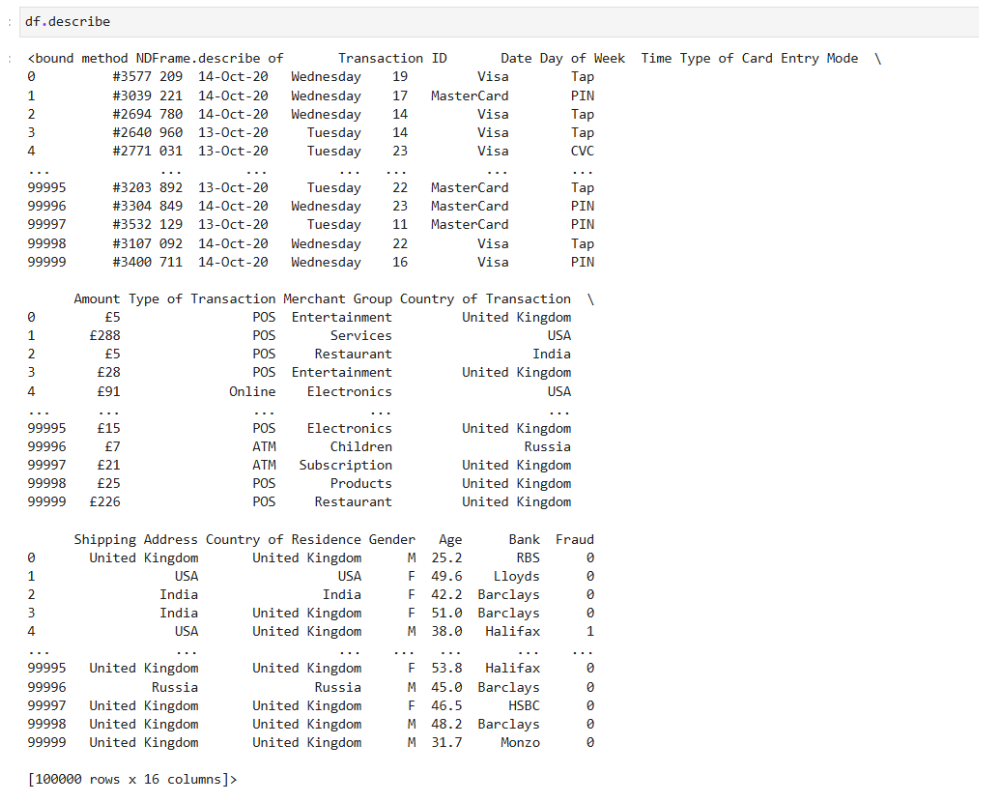
```
(100000, 16)
```



```
df.head()
```

| | Transaction ID | Date | Day of Week | Time | Type of Card | Entry Mode | Amount | Type of Transaction | Merchant Group | Country of Transaction | Shipping Address | Country of Residence | Gender | Age | Bank | Fraud |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | #3577 209 | 14-Oct-20 | Wednesday | 19 | Visa | Tap | £5 | POS | Entertainment | United Kingdom | United Kingdom | United Kingdom | M | 25.2 | RBS | 0 |
| **1** | #3039 221 | 14-Oct-20 | Wednesday | 17 | MasterCard | PIN | £288 | POS | Services | USA | USA | USA | F | 49.6 | Lloyds | 0 |
| **2** | #2694 780 | 14-Oct-20 | Wednesday | 14 | Visa | Tap | £5 | POS | Restaurant | India | India | India | F | 42.2 | Barclays | 0 |
| **3** | #2640 960 | 13-Oct-20 | Tuesday | 14 | Visa | Tap | £28 | POS | Entertainment | United Kingdom | India | United Kingdom | F | 51.0 | Barclays | 0 |
| **4** | #2771 031 | 13-Oct-20 | Tuesday | 23 | Visa | CVC | £91 | Online | Electronics | USA | USA | United Kingdom | M | 38.0 | Halifax | 1 |

The dataset contains 100,000 rows and 16 columns, indicating that it includes 100,000 transactions with 16 features describing each transaction. This provides a substantial amount of data for analysis and model training. df.head() previews the first five rows, showcasing various features.



```
df.describe
```

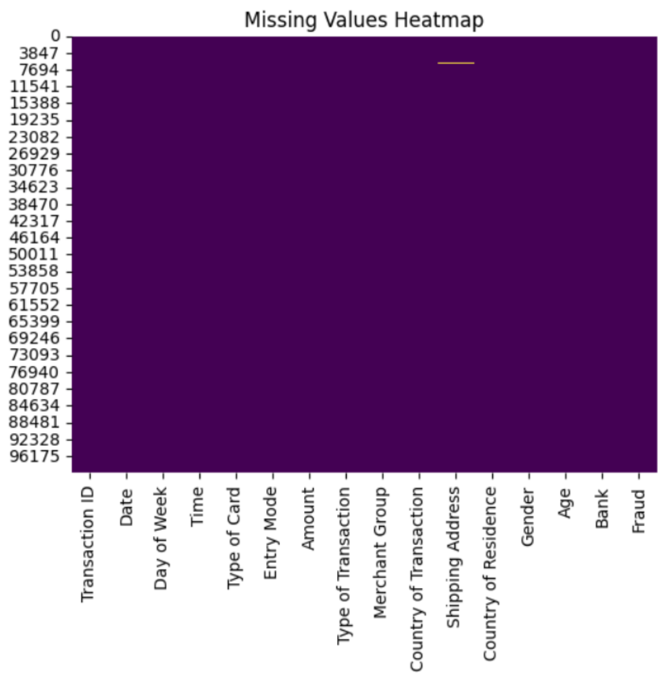
```
<bound method NDFrame.describe of       Transaction ID      Date Day of Week  Time Type of Card Entry Mode  \
0          #3577 209  14-Oct-20   Wednesday    19         Visa        Tap
1          #3039 221  14-Oct-20   Wednesday    17   MasterCard        PIN
2          #2694 780  14-Oct-20   Wednesday    14         Visa        Tap
3          #2640 960  13-Oct-20     Tuesday    14         Visa        Tap
4          #2771 031  13-Oct-20     Tuesday    23         Visa        CVC
...              ...        ...         ...   ...          ...        ...
99995      #3203 892  13-Oct-20     Tuesday    22   MasterCard        Tap
99996      #3304 849  14-Oct-20   Wednesday    23   MasterCard        PIN
99997      #3532 129  13-Oct-20     Tuesday    11   MasterCard        PIN
99998      #3107 092  14-Oct-20   Wednesday    22         Visa        Tap
99999      #3400 711  14-Oct-20   Wednesday    16         Visa        PIN

      Amount Type of Transaction Merchant Group Country of Transaction  \
0         £5                 POS  Entertainment         United Kingdom
1       £288                 POS       Services                    USA
2         £5                 POS     Restaurant                  India
3        £28                 POS  Entertainment         United Kingdom
4        £91              Online    Electronics                    USA
...      ...                 ...            ...                    ...
99995    £15                 POS    Electronics         United Kingdom
99996     £7                 ATM       Children                 Russia
99997    £21                 ATM   Subscription         United Kingdom
99998    £25                 POS       Products         United Kingdom
99999   £226                 POS     Restaurant         United Kingdom

      Shipping Address Country of Residence Gender   Age      Bank  Fraud
0       United Kingdom       United Kingdom      M  25.2       RBS      0
1                  USA                  USA      F  49.6    Lloyds      0
2                India                India      F  42.2  Barclays      0
3                India       United Kingdom      F  51.0  Barclays      0
4                  USA       United Kingdom      M  38.0   Halifax      1
...                ...                  ...    ...   ...       ...    ...
99995   United Kingdom       United Kingdom      F  53.8   Halifax      0
99996           Russia               Russia      M  45.0  Barclays      0
99997   United Kingdom       United Kingdom      F  46.5      HSBC      0
99998   United Kingdom       United Kingdom      M  48.2  Barclays      0
99999   United Kingdom       United Kingdom      M  31.7     Monzo      0

[100000 rows x 16 columns]>
```

The descriptive summary provides a detailed overview of the dataset:

- The dataset consists of 16 columns and 100,000 rows, as previously observed.
- Key features include both categorical and numerical attributes, such as transaction details, demographic data, and the fraud indicator.
- Columns like Transaction ID, Date, and Amount give specific insights into the transactional data.
- Demographic features like Gender, Age, and Bank are valuable for identifying patterns related to fraud.
- The Fraud column, as the target variable, classifies transactions as legitimate (0) or fraudulent (1).

This exploration highlights the dataset's structure, which is well-suited for machine learning and predictive modeling. Further preprocessing and visualization will uncover deeper insights into the data distribution and patterns.

```python
df.isnull().sum()
# Visualize missing values
sns.heatmap(df.isnull(), cbar=False, cmap="viridis")
plt.title("Missing Values Heatmap")
plt.show()
```



The missing values heatmap visually represents null values in the dataset. The heatmap indicates that most of the dataset is complete, with a minimal presence of missing values in certain columns. Addressing these missing values is essential during data preprocessing to ensure the integrity and performance of machine learning models.

Filling Missing Values - There are 6 Missing Values in the amount , which we will replace by Mean.

```python
df['Amount'] = df['Amount'].str.replace('£', '', regex=False).astype(float) # Remove Currency Symbol and covert to numeric
df['Amount'].fillna(df['Amount'].mean(), inplace=True)
```

Removing Rows which has NA or missing values. Removing them because these are non-numerical Rows.

Merchant Group - 10

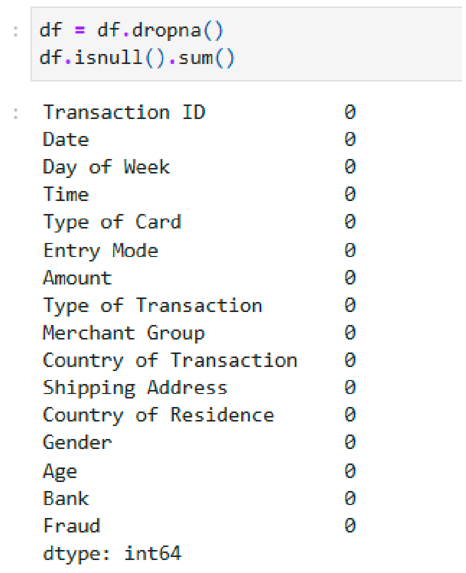
Shipping Address - 5

Gender - 4

The above code performs data preprocessing on the **"Amount"** column:

1. **Removing Currency Symbols**:
   - The str.replace('£', '', regex=False) removes the "£" symbol from the values in the "Amount" column, making it suitable for numerical analysis.
2. **Converting to Numeric**:
   - After removing the currency symbol, the .astype(float) converts the values in the column to a numeric data type (float) for further calculations and modeling.
3. **Handling Missing Values**:
   - The fillna(df['Amount'].mean(), inplace=True) replaces any missing values in the "Amount" column with the column's mean value, ensuring no null entries remain.

This step is crucial for preparing the data, as it converts the "Amount" column into a consistent, numeric format while addressing potential issues caused by missing values.

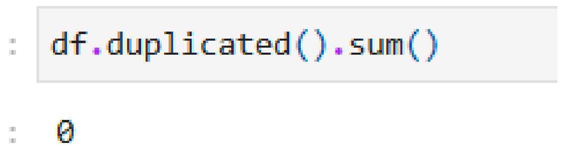


```
df = df.dropna()
df.isnull().sum()
```

```
Transaction ID            0
Date                      0
Day of Week               0
Time                      0
Type of Card              0
Entry Mode                0
Amount                    0
Type of Transaction       0
Merchant Group            0
Country of Transaction    0
Shipping Address          0
Country of Residence      0
Gender                    0
Age                       0
Bank                      0
Fraud                     0
dtype: int64
```

The missing values in the dataset have been successfully handled by using df.dropna(), which removes all rows containing null values.

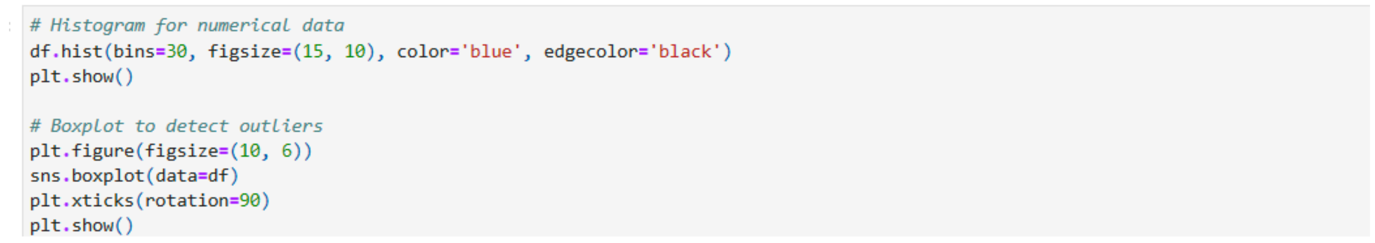
The output of df.isnull().sum() confirms that there are **no** longer any **missing values** in any of the columns. This ensures the dataset is clean and ready for further analysis and machine learning model training.



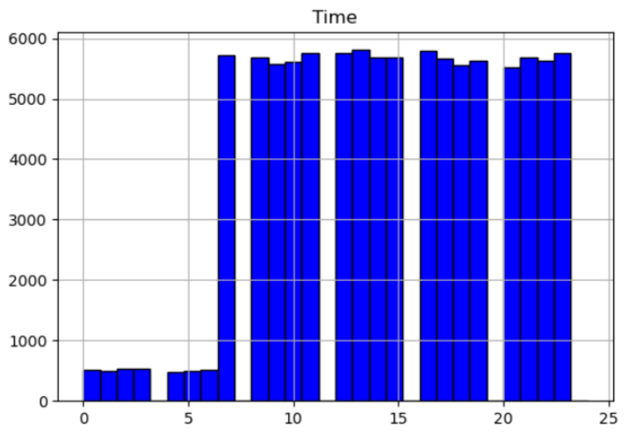
```
df.duplicated().sum()
```

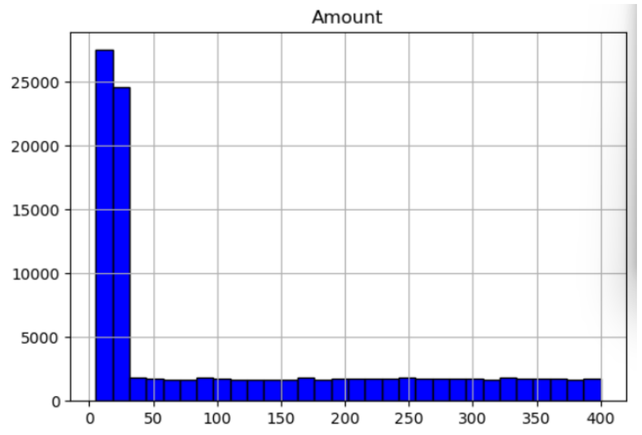
```
0
```

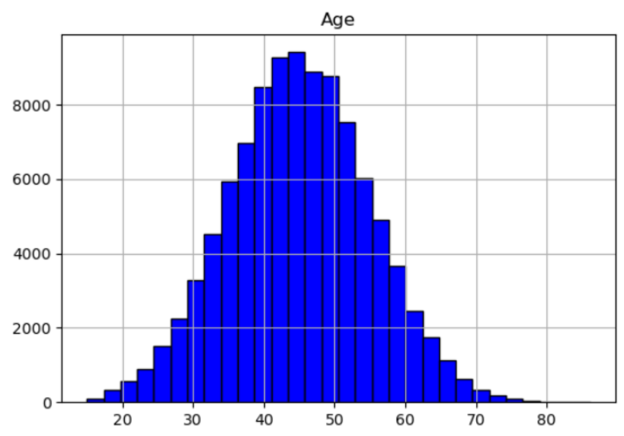
The output of df.duplicated().sum() indicates that there are **no duplicate rows** in the dataset. This ensures that the data is unique and does not contain redundant entries, which is essential for maintaining the quality and reliability of the analysis and model training.

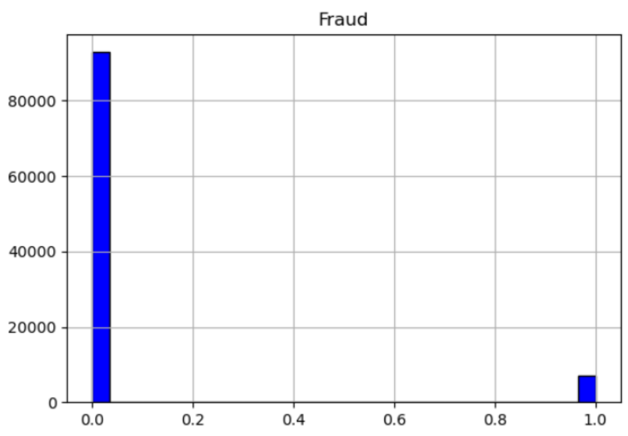


```
# Histogram for numerical data
df.hist(bins=30, figsize=(15, 10), color='blue', edgecolor='black')
plt.show()

# Boxplot to detect outliers
plt.figure(figsize=(10, 6))
sns.boxplot(data=df)
plt.xticks(rotation=90)
plt.show()
```



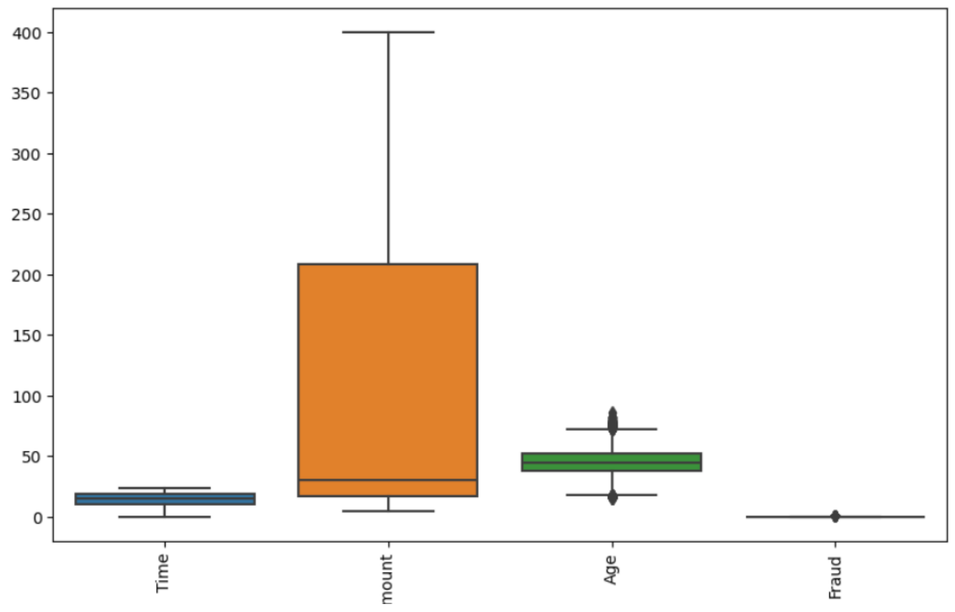






The visualizations offer key insights into the numerical data in the dataset:

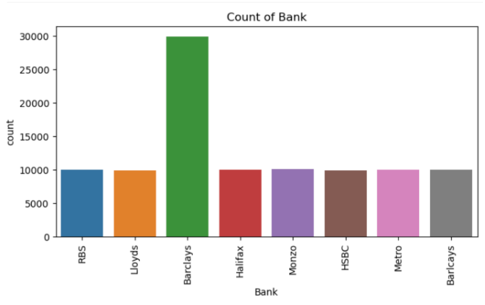
1. **Histograms**:
   - **Time**: The transactions appear evenly distributed across the day.
   - **Amount**: The majority of transactions have low amounts, with a sharp decline in frequency for higher amounts.
   - **Age**: Follows a normal distribution, with most users falling between 30 and 60 years old.
   - **Fraud**: Highlights the class imbalance, with fraudulent transactions being much less frequent than legitimate ones.

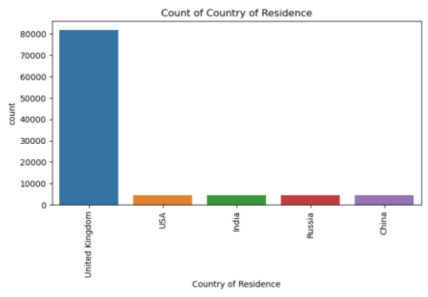


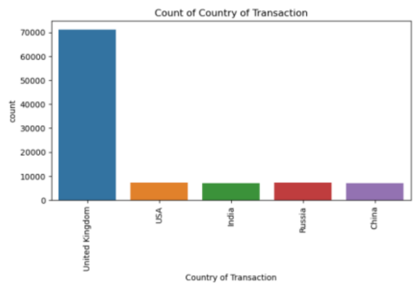
2. **Boxplot**:
   - **Amount**: Indicates the presence of outliers, with a wide interquartile range (IQR).
   - **Age**: Few minor outliers are present, but most values are concentrated within the typical age range.
   - **Fraud**: Clearly shows the class imbalance, as 0 (legitimate) dominates.

These visualizations help in understanding the data distribution, identifying outliers, and addressing the imbalanced target variable for model training.

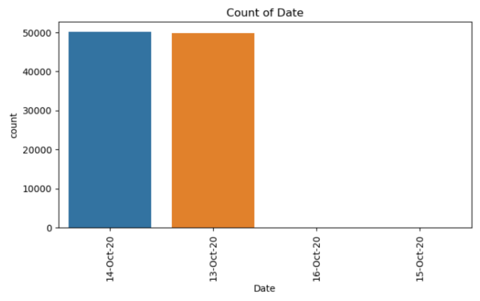
```python
# Specify columns to exclude from visualization
excluded_columns = ['Transaction ID']

# Identify categorical columns excluding the specified ones
categorical_columns = df.select_dtypes(include=['object']).columns.difference(excluded_columns)

# Plot bar plots for the remaining categorical columns
for col in categorical_columns:
    plt.figure(figsize=(8, 4))
    sns.countplot(x=col, data=df)
    plt.title(f"Count of {col}")
    plt.xticks(rotation=90)
    plt.show()
```

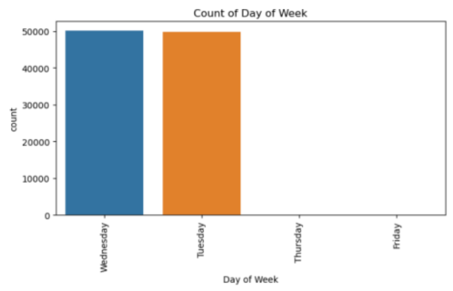


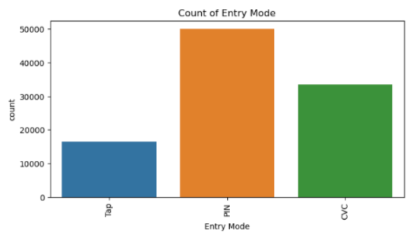


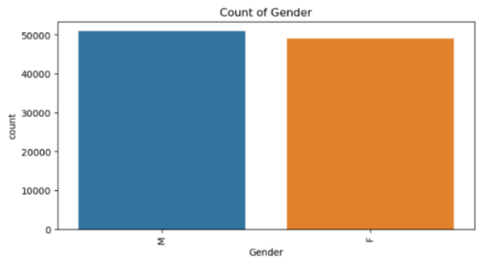


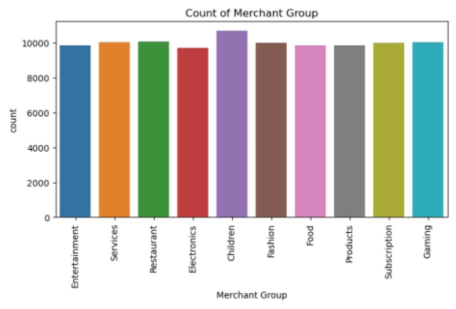
Country of Transaction

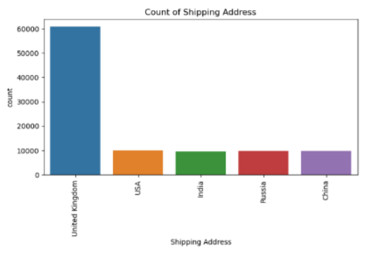


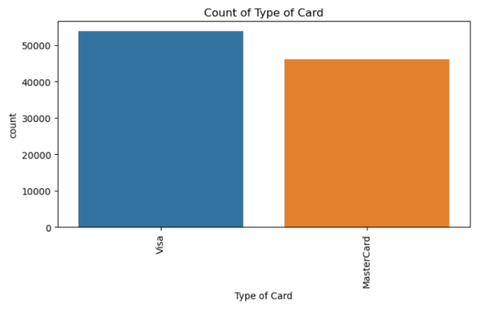










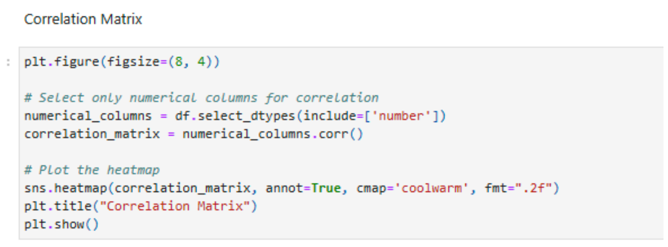


The visualizations reveal key trends in categorical data:

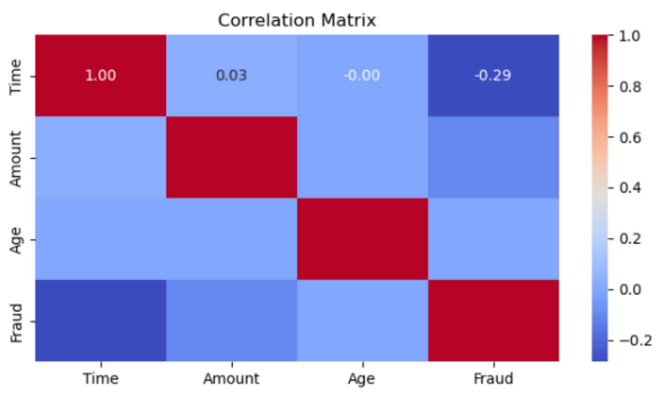
1. **Bank**: Barclays is the most commonly used bank, followed by RBS, Lloyds, and Halifax.
2. **Country of Residence**: Most users reside in the United Kingdom, with smaller representations from the USA, India, Russia, and China.
3. **Country of Transaction**: A majority of transactions occur in the United Kingdom, mirroring the residence data.
4. **Day of Week**: Transactions are distributed evenly between Tuesday and Wednesday.
5. **Entry Mode**: PIN is the most common method of card entry, followed by CVC and Tap.
6. **Gender**: Male and Female customers are nearly equally represented.
7. **Merchant Group**: Transactions are evenly distributed across merchant groups like Entertainment, Services, Electronics, and others.
8. **Shipping Address**: Most shipping addresses align with the United Kingdom, consistent with residence and transaction trends.

These insights will help in understanding user behaviour and identifying patterns associated with fraudulent transactions.

## CORRELATION MATRIX


Correlation Matrix

```
plt.figure(figsize=(8, 4))

# Select only numerical columns for correlation
numerical_columns = df.select_dtypes(include=['number'])
correlation_matrix = numerical_columns.corr()

# Plot the heatmap
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()
```



The **Correlation Matrix** heatmap provides insights into the relationships between numerical features:

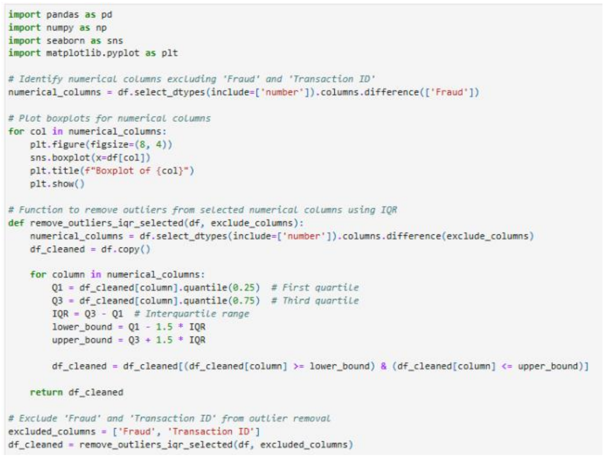
1. **Time and Fraud**: A weak negative correlation (-0.29) suggests minimal influence of transaction time on fraudulent behavior.
2. **Amount and Fraud**: A negligible correlation (0.03) indicates that the transaction amount does not strongly relate to fraud.
3. **Age and Fraud**: No notable correlation (0.00) is observed, showing that age has little impact on fraud.
4. **Inter-feature Relationships**: The numerical features, such as Time, Amount, and Age, are mostly independent of one another.

These observations suggest that the numerical features alone may not be strong predictors of fraud, emphasizing the need to consider categorical and other derived features for building effective machine learning models.

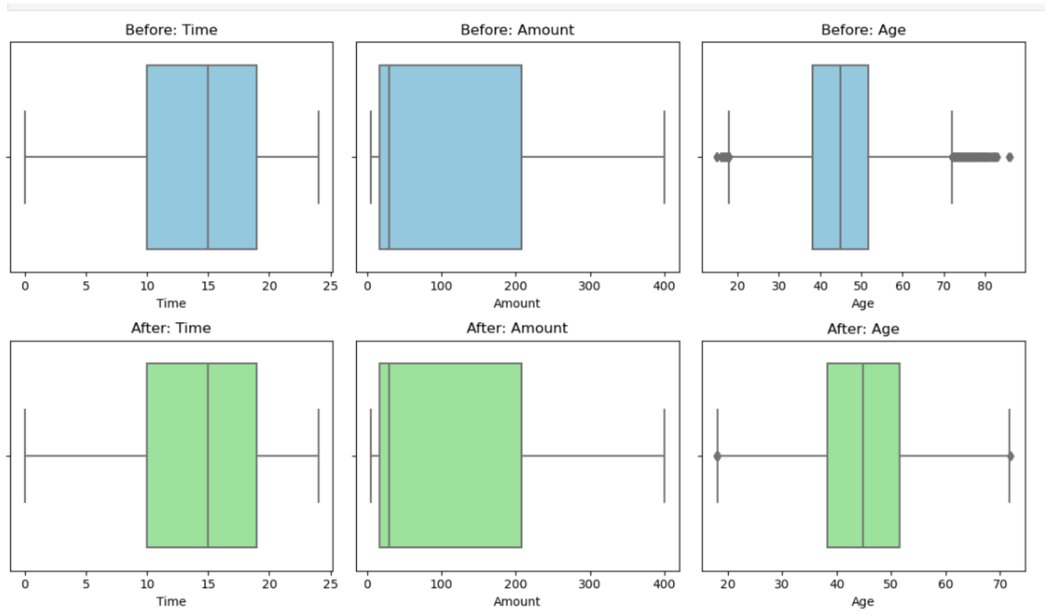
## OUTLIERS

Outliers are data points that significantly deviate from the majority of the dataset, either being much higher or lower than the expected range.

Detecting and handling outliers is a crucial step in data preprocessing because they can misrepresent statistical analyses, bias machine learning models, and impact the accuracy of predictions. Properly addressing outliers ensures that the dataset is clean, reliable, and representative of the underlying patterns.



```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Identify numerical columns excluding 'Fraud' and 'Transaction ID'
numerical_columns = df.select_dtypes(include=['number']).columns.difference(['Fraud'])

# Plot boxplots for numerical columns
for col in numerical_columns:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x=df[col])
    plt.title(f"Boxplot of {col}")
    plt.show()

# Function to remove outliers from selected numerical columns using IQR
def remove_outliers_iqr_selected(df, exclude_columns):
    numerical_columns = df.select_dtypes(include=['number']).columns.difference(exclude_columns)
    df_cleaned = df.copy()

    for column in numerical_columns:
        Q1 = df_cleaned[column].quantile(0.25)  # First quartile
        Q3 = df_cleaned[column].quantile(0.75)  # Third quartile
        IQR = Q3 - Q1  # Interquartile range
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        df_cleaned = df_cleaned[(df_cleaned[column] >= lower_bound) & (df_cleaned[column] <= upper_bound)]

    return df_cleaned

# Exclude 'Fraud' and 'Transaction ID' from outlier removal
excluded_columns = ['Fraud', 'Transaction ID']
df_cleaned = remove_outliers_iqr_selected(df, excluded_columns)
```



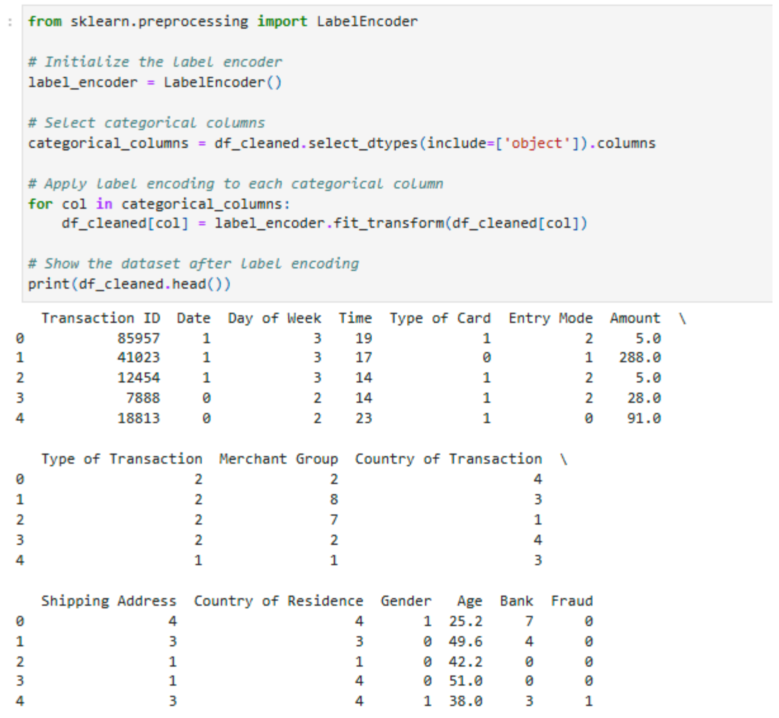
Outliers were detected and removed from the dataset using the **Interquartile Range (IQR)** method. The IQR-based approach calculates the lower and upper bounds for each numerical column, excluding irrelevant features such as Fraud and Transaction ID. Data points falling outside the range $\boldsymbol{Q1 - 1.5 \times IQR}$ and $\boldsymbol{Q3 + 1.5 \times IQR + 1.5 \times IQR}$ were classified as outliers and removed to ensure cleaner and more reliable data.

Visualizations before and after outlier removal, represented using boxplots, highlight the changes in distributions for key columns such as Amount and Age. The process effectively reduced the presence of extreme values while maintaining the integrity of the data.

## PRE-PROCESSING

### LABEL ENCODING

Label Encoding is a preprocessing technique used to convert categorical variables into numerical values by assigning a unique integer to each category. It simplifies categorical data, making it suitable for machine learning algorithms that require numerical input.



```python
from sklearn.preprocessing import LabelEncoder

# Initialize the label encoder
label_encoder = LabelEncoder()

# Select categorical columns
categorical_columns = df_cleaned.select_dtypes(include=['object']).columns

# Apply label encoding to each categorical column
for col in categorical_columns:
    df_cleaned[col] = label_encoder.fit_transform(df_cleaned[col])

# Show the dataset after label encoding
print(df_cleaned.head())
```

```
   Transaction ID  Date  Day of Week  Time  Type of Card  Entry Mode  Amount  \
0           85957     1            3    19             1           2     5.0
1           41023     1            3    17             0           1   288.0
2           12454     1            3    14             1           2     5.0
3            7888     0            2    14             1           2    28.0
4           18813     0            2    23             1           0    91.0

   Type of Transaction  Merchant Group  Country of Transaction  \
0                    2               2                      4
1                    2               8                      3
2                    2               7                      1
3                    2               2                      4
4                    1               1                      3

   Shipping Address  Country of Residence  Gender   Age  Bank  Fraud
0                 4                     4       1  25.2     7      0
1                 3                     3       0  49.6     4      0
2                 1                     1       0  42.2     0      0
3                 1                     4       0  51.0     0      0
4                 3                     4       1  38.0     3      1
```

To prepare the dataset for machine learning algorithms, categorical variables were converted into numerical values using **Label Encoding**. Label encoding assigns a unique integer to each category, transforming the categorical columns into a format that can be processed by ML models.

For instance, columns like Type of Card, Merchant Group, and Country of Residence, which originally contained string values, were replaced with corresponding numeric labels.

After label encoding, all categorical features are represented numerically, making the dataset fully numerical and ready for model training.

### INSIGHTS

1. **Compatibility for Machine Learning Models**: Label encoding ensures that categorical variables are compatible with machine learning algorithms, which typically require numerical inputs.
2. **Efficient Transformation**: The encoding preserves the structure of categorical variables without introducing additional complexity, such as one-hot encoding would.
3. **Model-Ready Dataset**: With all features in numeric form, the dataset is now fully prepared for training machine learning classifiers, ensuring seamless processing during modeling.

This step ensures a smooth transition from data preprocessing to model building, enabling effective utilization of categorical features in predictive analysis.

### Feature and Target Separation

```python
# Separate features and target
X = df_cleaned.drop(columns=['Fraud'])
y = df_cleaned['Fraud']
```

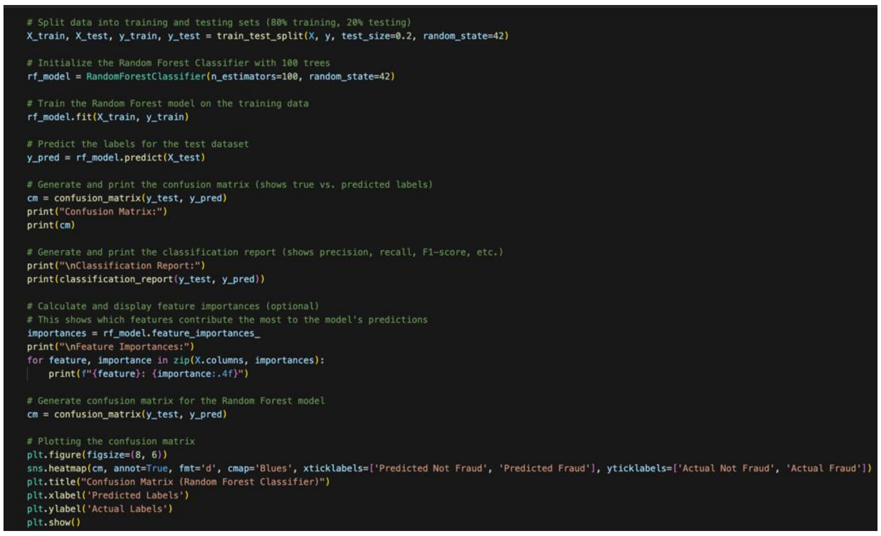
The code separates the dataset into **features (X)** and the **target variable (y)**:

- **X:** Contains all the independent features, excluding the Fraud column, which represents the predictors used by the model.
- **y:** Represents the dependent variable, Fraud, which is the **target** the machine learning model will predict.

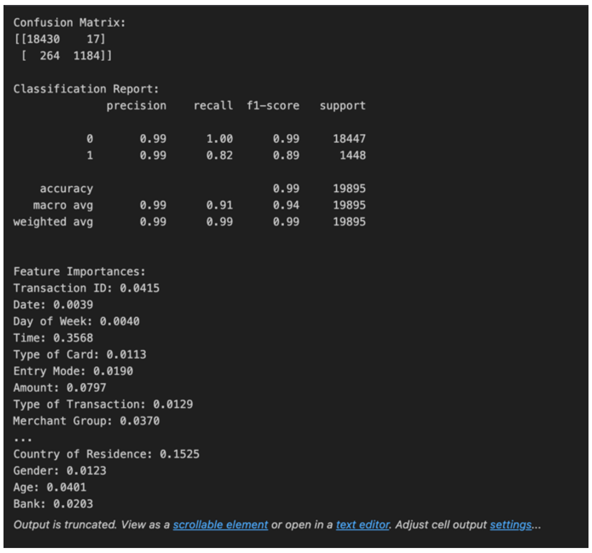
This step is crucial for preparing the dataset for training, as it ensures the model focuses on learning patterns in the features (X) to predict the target (y).

## RANDOM FOREST

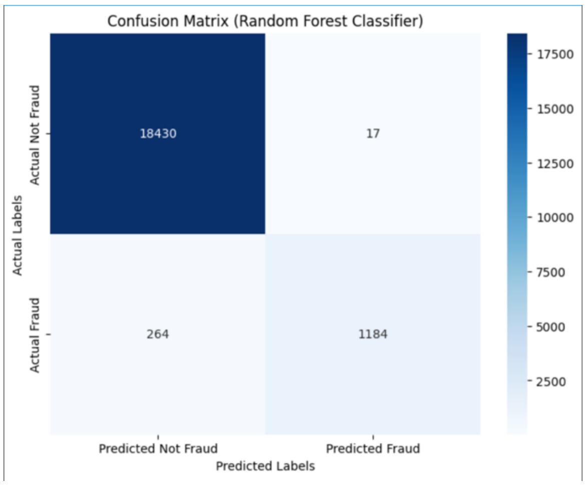
### Code



```python
# Split data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Random Forest Classifier with 100 trees
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the Random Forest model on the training data
rf_model.fit(X_train, y_train)

# Predict the labels for the test dataset
y_pred = rf_model.predict(X_test)

# Generate and print the confusion matrix (shows true vs. predicted labels)
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Generate and print the classification report (shows precision, recall, F1-score, etc.)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Calculate and display feature importances (optional)
# This shows which features contribute the most to the model's predictions
importances = rf_model.feature_importances_
print("\nFeature Importances:")
for feature, importance in zip(X.columns, importances):
    print(f"{feature}: {importance:.4f}")

# Generate confusion matrix for the Random Forest model
cm = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted Not Fraud', 'Predicted Fraud'], yticklabels=['Actual Not Fraud', 'Actual Fraud'])
plt.title("Confusion Matrix (Random Forest Classifier)")
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```



```
Confusion Matrix:
[[18430    17]
 [  264  1184]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     18447
           1       0.99      0.82      0.89      1448

    accuracy                           0.99     19895
   macro avg       0.99      0.91      0.94     19895
weighted avg       0.99      0.99      0.99     19895


Feature Importances:
Transaction ID: 0.0415
Date: 0.0039
Day of Week: 0.0040
Time: 0.3568
Type of Card: 0.0113
Entry Mode: 0.0190
Amount: 0.0797
Type of Transaction: 0.0129
Merchant Group: 0.0370
...
Country of Residence: 0.1525
Gender: 0.0123
Age: 0.0401
Bank: 0.0203
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```



The Random Forest model is intended to detect fraudulent credit card transactions. The following describes the behaviour and performance of the model:

1. **Confusion Matrix Analysis**
   The following is revealed by the confusion matrix:

   The confusion matrix reveals the following:

- **True Negatives (18430):** Non-fraudulent transactions correctly identified as not fraudulent.
- **False Positives (17):** Non-fraudulent transactions incorrectly classified as fraudulent.
- **False Negatives (264):** Fraudulent transactions incorrectly classified as not fraudulent.
- **True Positives (1184):** Fraudulent transactions correctly identified as fraudulent.

This indicates that the model is highly accurate but still struggles slightly with identifying all fraudulent cases (264 were overlooked).

2. **Performance Measure**
   According to the classification report:

   (Fraud Class-1) **Accuracy: 99%**
   This indicates that 99% of the time, the model is right when it flags a transaction as fraudulent.

   (Fraud Class-1) **Recall: 82%**
   Eighty-two percent of the real fraudulent transactions are successfully identified by the model.

   (Fraud Class-1) **F1-Score: 89%**
   The F1-score shows good overall performance in identifying fraud by striking a balance between precision and recall.

   **99% Overall accuracy**
   On the complete dataset, the model exhibits remarkable performance.

3. **Feature Importances**

The Random Forest Classifier ranks the features by importance. For this dataset:

- **Key Features:**
    - **Time:** Most significant **(35.68%)**, indicating when the transaction occurred strongly correlates with fraud.
    - **Country of Residence (15.25%) & Country of Transaction (13.07%):** Suggesting fraud patterns vary significantly based on geography.
    - **Amount (7.97%):** Larger transaction amounts might have higher fraud likelihood.
    - **Shipping Address (7.79%):** Indicates a link between addresses and fraudulent transactions.

These insights allow investigators to focus on specific factors like time of day or geographic trends to improve fraud detection.

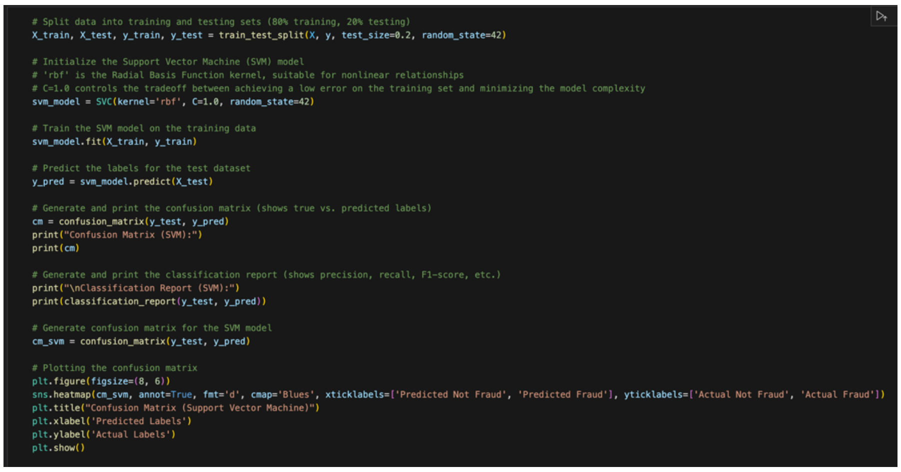
4. **Model Insights**
   **Strengths:**
    - High precision guarantees that transactions that are flagged are probably fraudulent.
    - High accuracy throughout the dataset, reducing the quantity of inaccurate forecasts.

### WHY CHOOSE THE RANDOM FOREST MODEL

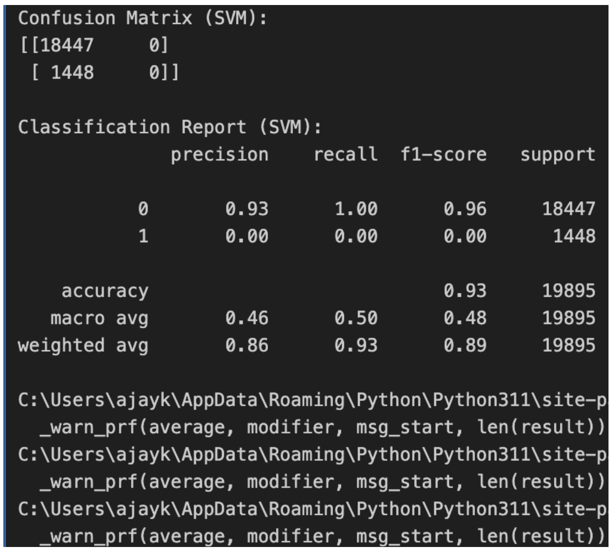
- Better Fraud Detection: High recall (82%) for fraud cases, crucial for minimizing financial loss.
- Handles Imbalanced Data: Works well with techniques like class weighting or sampling.
- Feature Importance: Provides insights into which factors drive fraud.
- Scalability: Efficient for large datasets and less prone to overfitting

## SUPPORT VECTOR MACHINE

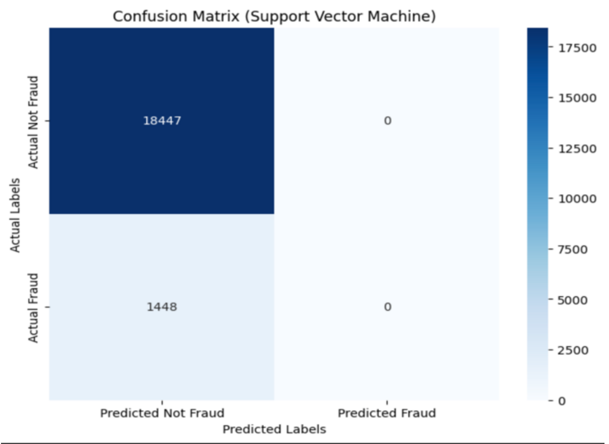
### Code



```python
# Split data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Support Vector Machine (SVM) model
# 'rbf' is the Radial Basis Function kernel, suitable for nonlinear relationships
# C=1.0 controls the tradeoff between achieving a low error on the training set and minimizing the model complexity
svm_model = SVC(kernel='rbf', C=1.0, random_state=42)

# Train the SVM model on the training data
svm_model.fit(X_train, y_train)

# Predict the labels for the test dataset
y_pred = svm_model.predict(X_test)

# Generate and print the confusion matrix (shows true vs. predicted labels)
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix (SVM):")
print(cm)

# Generate and print the classification report (shows precision, recall, F1-score, etc.)
print("\nClassification Report (SVM):")
print(classification_report(y_test, y_pred))

# Generate confusion matrix for the SVM model
cm_svm = confusion_matrix(y_test, y_pred)

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm_svm, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted Not Fraud', 'Predicted Fraud'], yticklabels=['Actual Not Fraud', 'Actual Fraud'])
plt.title("Confusion Matrix (Support Vector Machine)")
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```



```
Confusion Matrix (SVM):
[[18447     0]
 [ 1448     0]]

Classification Report (SVM):
              precision    recall  f1-score   support

           0       0.93      1.00      0.96     18447
           1       0.00      0.00      0.00      1448

    accuracy                           0.93     19895
   macro avg       0.46      0.50      0.48     19895
weighted avg       0.86      0.93      0.89     19895

C:\Users\ajayk\AppData\Roaming\Python\Python311\site-p
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ajayk\AppData\Roaming\Python\Python311\site-p
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\ajayk\AppData\Roaming\Python\Python311\site-p
  _warn_prf(average, modifier, msg_start, len(result))
```



1. **Confusion Matrix Analysis(SVM Model)**

The confusion matrix indicates:

- **True Negatives (18447):** Non-fraudulent transactions correctly identified as not fraudulent.
- **False Positives (0):** Non-fraudulent transactions misclassified as fraudulent.
- **False Negatives (1448):** Fraudulent transactions incorrectly classified as not fraudulent.
- **True Positives (0):** No fraudulent transactions were correctly identified as fraudulent.

This means the SVM model is entirely failing to detect fraudulent transactions (class 1).

2. **Performance Metrics**

From the classification report:

- **Precision (Fraud Class - 1): 0.00**
  The model did not predict any fraudulent transactions, resulting in undefined precision for fraud detection.
- **Recall (Fraud Class - 1): 0.00**
  The model failed to identify any of the actual fraudulent transactions.
- **F1-Score (Fraud Class - 1): 0.00**
  The F1-score, which balances precision and recall, is also 0 because both are 0.

- **Overall Accuracy: 93%**
  The high accuracy is misleading due to the model's bias toward the majority class (`not fraud`) in this imbalanced dataset.
- **Macro Average Recall: 0.50**
  Highlights the skewed performance across the two classes.

### 3. Model Insights

**Strengths:**

- High accuracy for non-fraudulent transactions.
- Good at learning patterns in the majority class.
- Robust to overfitting for well-tuned models.

**WHY CHOOSE THE SVM MODEL**

- High Accuracy for Legitimate Transactions: 100% recall for non-fraud cases, reducing false positives.
- Complex Boundaries: Captures non-linear relationships effectively.
- Better for Smaller Datasets: Performs well when data is carefully preprocessed.

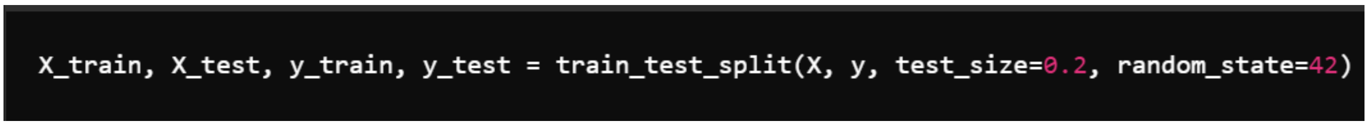
## XGBoost

**Description**- XGBoost (Extreme Gradient Boosting) is a decision tree-based ensemble learning technique that improves weak models to predict outcomes. It is particularly successful for huge datasets and classification problems, such predicting fraud (binary: 0 = not fraud, 1 = fraud) utilizing characteristics such as transaction amount, card type, and user location.

**Explanation-**

• Step 1: **Divide data into training and testing sets**
The dataset is separated in two parts: 80% for training the model and 20% for testing its correctness. This enables an unbiased assessment of the model's performance on unseen data.

**Code:**



```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

• Step 2- Initialize and train the XGBoost model.
An XGBoost classifier is developed and trained using the training data. The model employs options such as use_label_encoder=False (to avoid deprecated warnings), eval_metric='logloss' (to assess model performance using log-loss), and random_state=42 for reproducibility.

**Code:**

```
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb_model.fit(X_train, y_train)
```

Step 3- Make Predictions
The trained XGBoost model is used to predict whether or not the test dataset contains fraud.

**Code**:

```
y_pred = xgb_model.predict(X_test)
```

• Step 4- Evaluate the Model with a Confusion Matrix and a Classification Report.
The confusion matrix and classification report describe the model's accuracy and include measures like precision, recall, and F1-score for both classes (fraud and non-fraud).

**Code**:

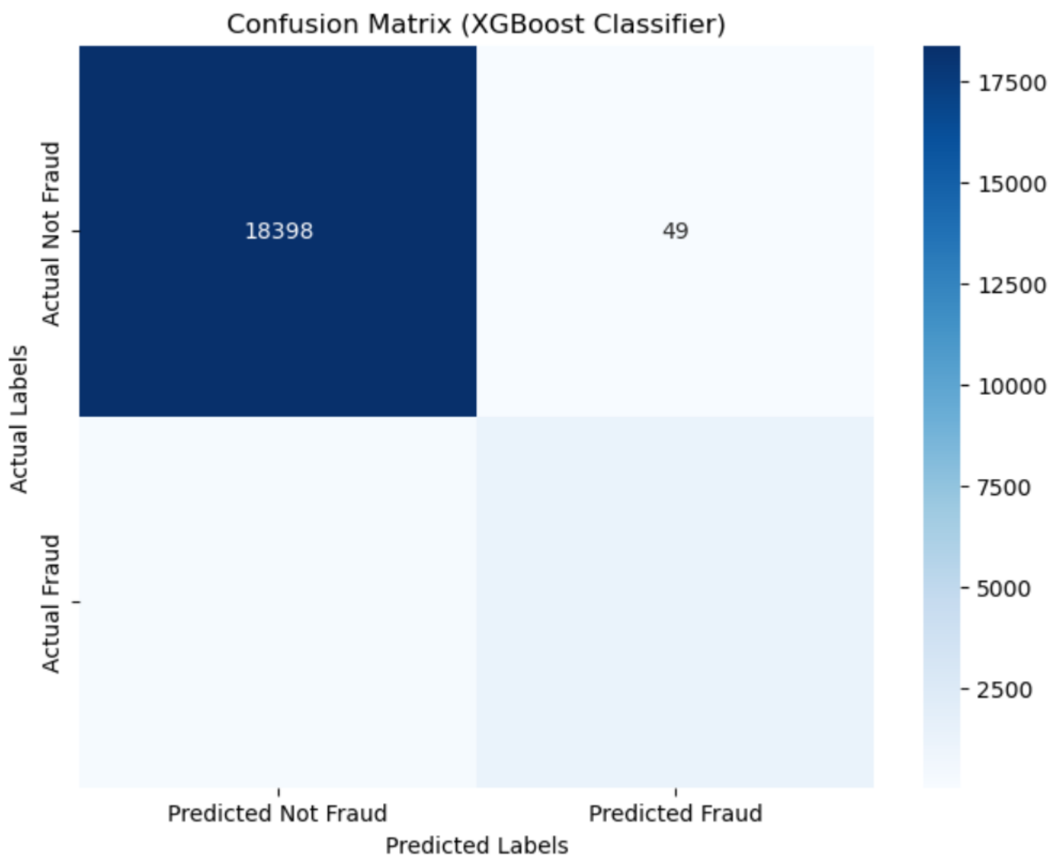
```
cm = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

• Step 5- Visualize the Confusion Matrix.
A heatmap is constructed to visually depict the confusion matrix, emphasizing genuine versus forecasted fraud incidents.

**Code:**

```
plt.figure(figsize=(8, 6))
sns.heatmap(cm_xgb, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted Not Fraud', 'Predicted Fraud'], yticklabels=['Actual Not Fraud', 'Actual Fraud']
plt.title("Confusion Matrix (XGBoost Classifier)")
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```

**Confusion Matrix-**



This confusion matrix depicts the performance of the XGBoost classifier in recognizing fraud (1) and non-fraud (0) transactions:

1. **True Negatives (top left: 18,398)**
   The model correctly forecasted 18,398 transactions as "Not Fraud" even if they were not fraudulent.
2. **False Positives (Top-right: 49)**
   The model mistakenly classified 49 transactions as "Fraud" when they were not fraudulent.
3. **False Negatives (bottom-left: 240)**
   The model misidentified 240 transactions as "Not Fraud" when they were indeed fraudulent.
4. **True Positives (bottom-right: 1,208)**
   The model properly classified 1,208 transactions as "Fraud" when they were in fact fraudulent.

XGBoost excels at huge datasets and challenging classification tasks thanks to its ensemble learning technique and ability to manage missing data and non-linear correlations.

## Why Choose XGBoost

• XGBoost performs exceptionally well with high precision and recall, which are essential for detecting fraud cases without overloading the model with false positives.

• It is particularly good at binary classification tasks, particularly in imbalanced datasets like fraud detection.

• It is highly effective for large datasets and complex classification tasks because of its ensemble learning approach and capacity to handle missing data and non-linear relationships.

In conclusion, XGBoost was selected because of its remarkable classification performance in fraud detection tasks, its efficacy on imbalanced datasets, and its capacity to manage intricate relationships in the data.

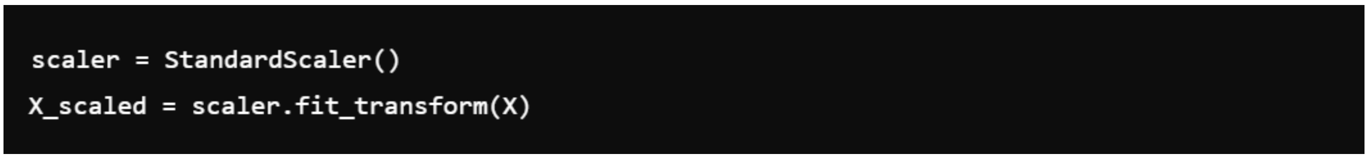
## k-Nearest Neighbors (k-NN)

**Description -** k-Nearest Neighbors (k-NN) is a non-parametric classification technique that assigns labels based on the majority vote of the nearest data points. In this scenario, k-NN predicts whether a transaction is fraudulent (binary: 0 = not fraud, 1 = fraud) using data such as transaction amount and card type. The confusion matrix and classification report evaluate the model's accuracy and detailed performance data.

**Explanation-**

• Step 1- **Scale the Features**
Feature scaling is used to ensure that all features have comparable ranges, which is critical for distance-based algorithms such as k-NN.

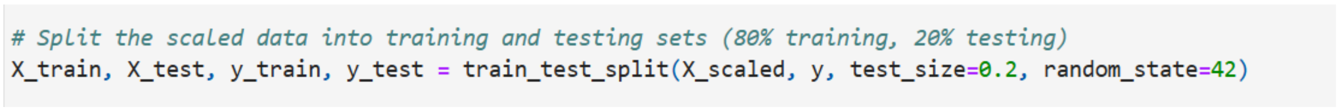
**Code:**



```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

• Step 2- **Separate the data into training and testing sets**
The dataset is separated into 80% training and 20% testing, allowing for an unbiased evaluation of the model's performance on previously unknown data.

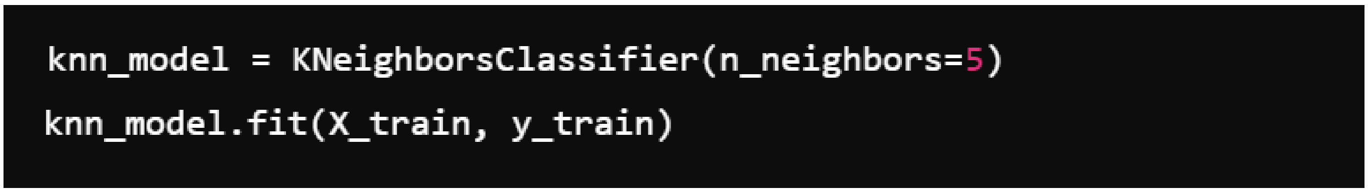
**Code:**



```
# Split the scaled data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Step 3-**Initialize and train the k-NN model**
A k-NN classifier is built and trained with the training data. The number of neighbors (k = 5) was chosen to strike a compromise between performance and computational expense.

**Code:**



```
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
```

• Step 4- **Make Predictions**
Based on the test data, the trained k-NN model predicts whether a transaction is fraudulent.

**Code:**

```
y_pred = knn_model.predict(X_test)
```

• Step 5- **Evaluate the Model with a Confusion Matrix and Classification Report**
The confusion matrix and classification report include detailed metrics for evaluating the k-NN model's performance, including as precision, recall, and F1-score.

**Code:**

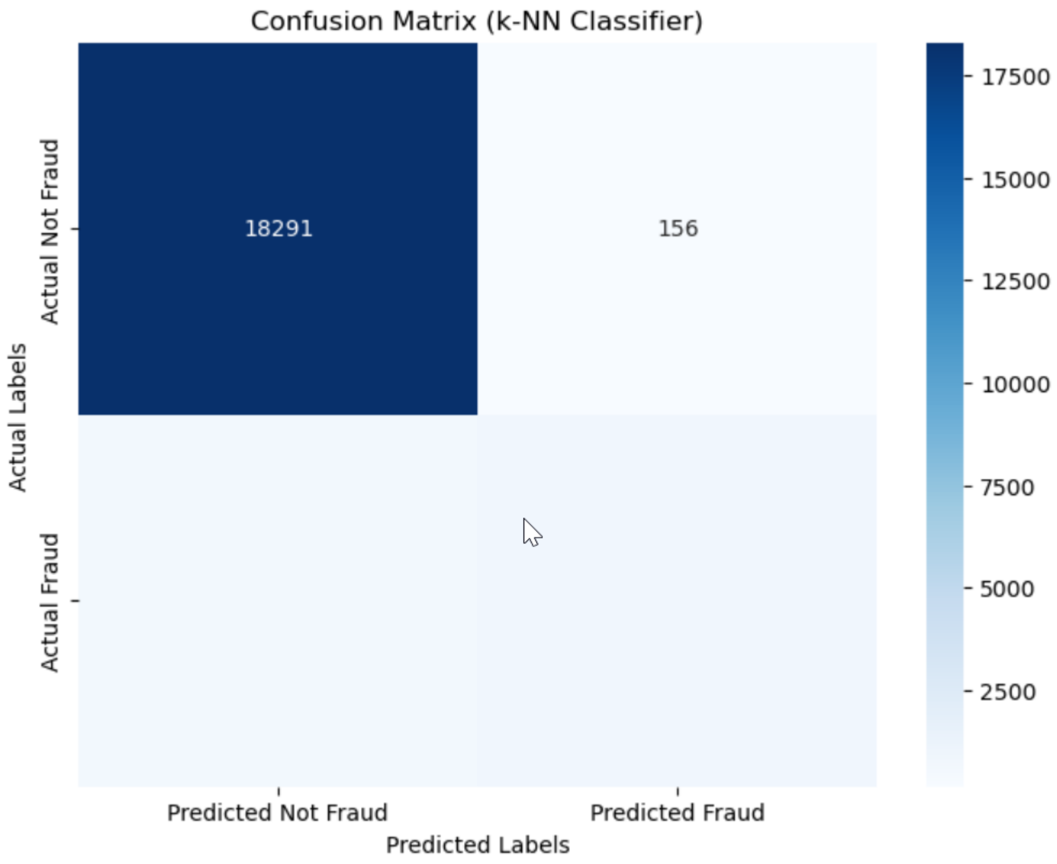
```
cm = confusion_matrix(y_test, y_pred)
print(classification_report(y_test, y_pred))
```

• Step 6-**Visualize the Confusion Matrix**
A heatmap is constructed to visually represent the confusion matrix, highlighting real versus expected fraud cases.

**Code:**

```
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Blues', xticklabels=['Predicted Not Fraud', 'Predicted Fraud'], yticklabels=['Actual Not Fraud', 'Actual Fraud']
plt.title("Confusion Matrix (k-NN Classifier)")
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```

**Confusion Matrix**



This confusion matrix demonstrates the performance of the k-NN classifier in recognizing fraudulent (1) and non-fraudulent (0) transactions:

1. **True Negatives (top left: 18,291)**
   The model correctly forecasted 18,291 transactions as "Not Fraud" even if they were not fraudulent.
2. **False Positives (Top-right: 156)**
   The model mistakenly classified 156 transactions as "Fraud" when they were not fraudulent.
3. **False Negatives (bottom-left: 590)**
   The model misidentified 590 transactions as "Not Fraud" when they were indeed fraudulent.
4. **True Positives (bottom-right: 858)**
   The model properly identified 858 transactions as "Fraud" when they were in fact fraudulent.

## Why Choose k-NN?

• k-NN is a straightforward technique that requires no assumptions regarding data distribution.

• k-NN works well with smaller datasets and may effectively detect fraud when combined with feature scaling.
• It catches local patterns in data, which is important for spotting fraud trends that may not be obvious with global features.

In conclusion k-NN was chosen since it is simple and effective on smaller datasets. However, it may not scale as well with huge datasets or high-dimensional data as other models, such as XGBoost.

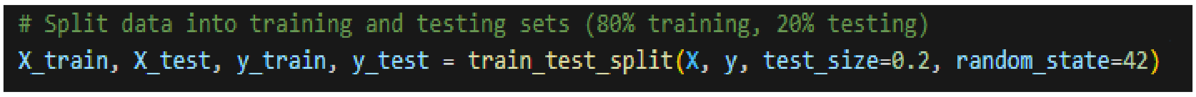
# Naive Bayes

**Description -** Naive Bayes is a probabilistic machine learning algorithm based on Bayes' Theorem, assuming independence between features. In the dataset provided, it predicts fraud (binary: 0 = not fraud, 1 = fraud) using features like card type, amount, age, and country. The confusion matrix evaluates its classification performance on test data.

**Explanation -**

- Step 1 - **Split the Data into Training and Testing Sets**
  The dataset is divided into 80% for training the model and 20% for testing its accuracy.

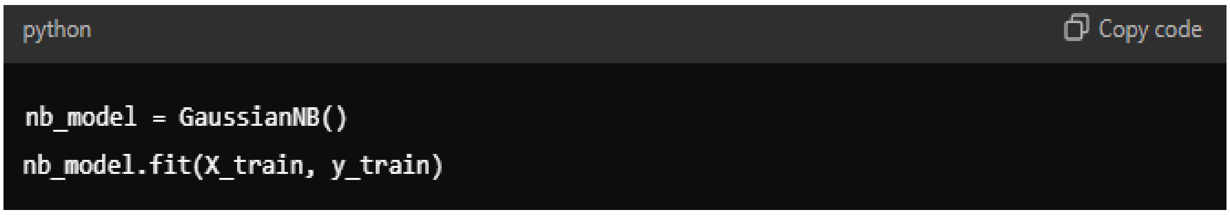
  **Code:**



```python
# Split data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Step 2 - **Initialize and Train the Naive Bayes Model**
  A Naive Bayes classifier (GaussianNB for continuous data) is created and trained on the training data.

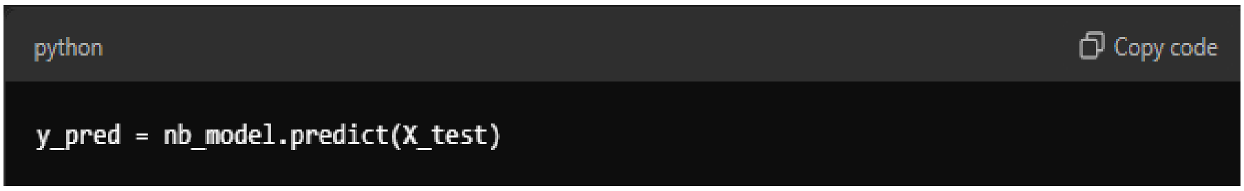
  **Code:**



```python
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
```

- Step 3 - **Make Predictions**
  The trained model predicts the fraud (or not) labels for the test dataset.

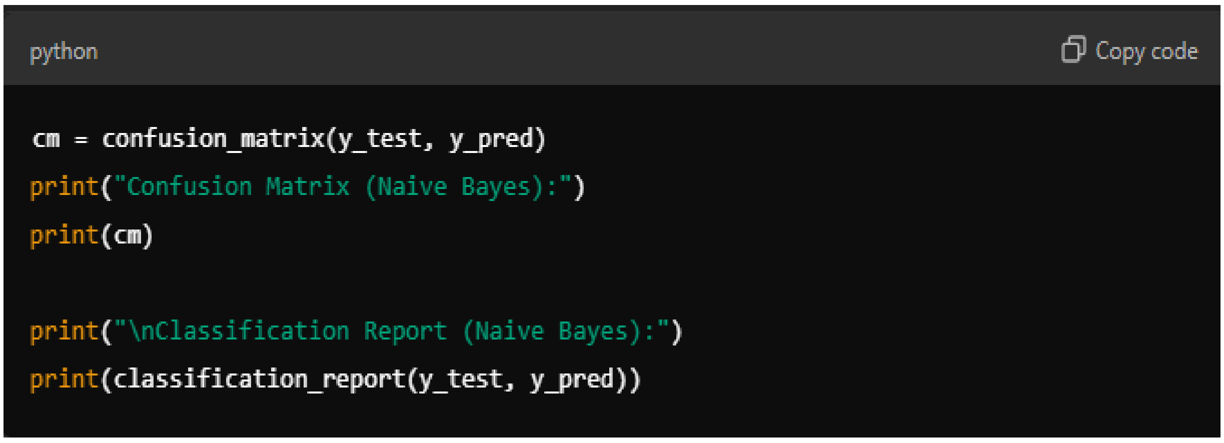
  **Code:**



```python
y_pred = nb_model.predict(X_test)
```

- Step 4 - **Evaluate the Model Using a Confusion Matrix and Classification Report**
  The confusion matrix and classification report summarize the model's accuracy and detailed metrics like precision, recall, and F1-score.

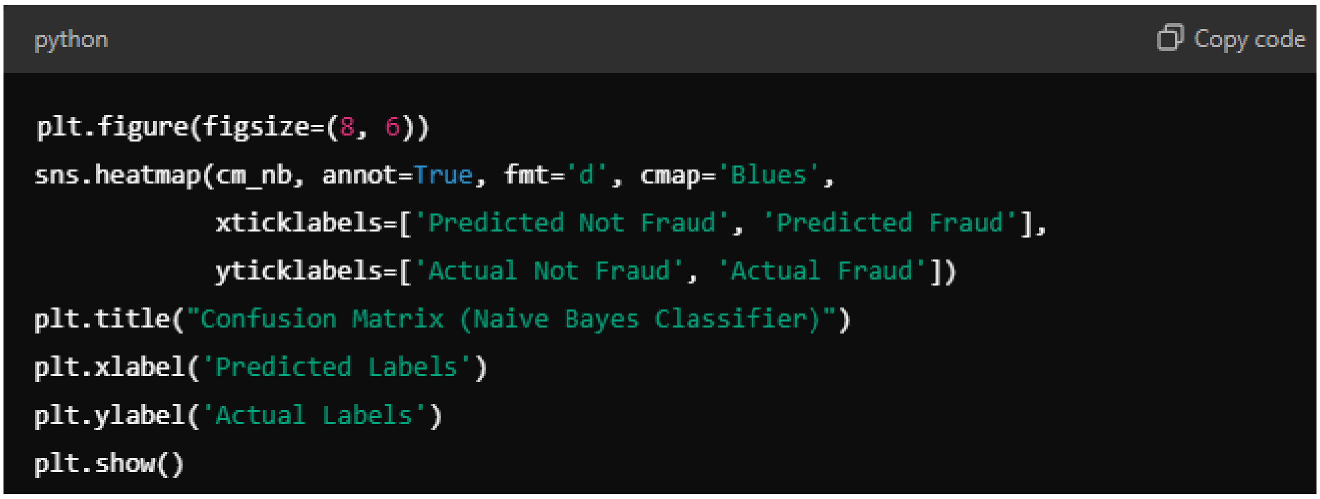
**Code -**



```python
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix (Naive Bayes):")
print(cm)

print("\nClassification Report (Naive Bayes):")
print(classification_report(y_test, y_pred))
```

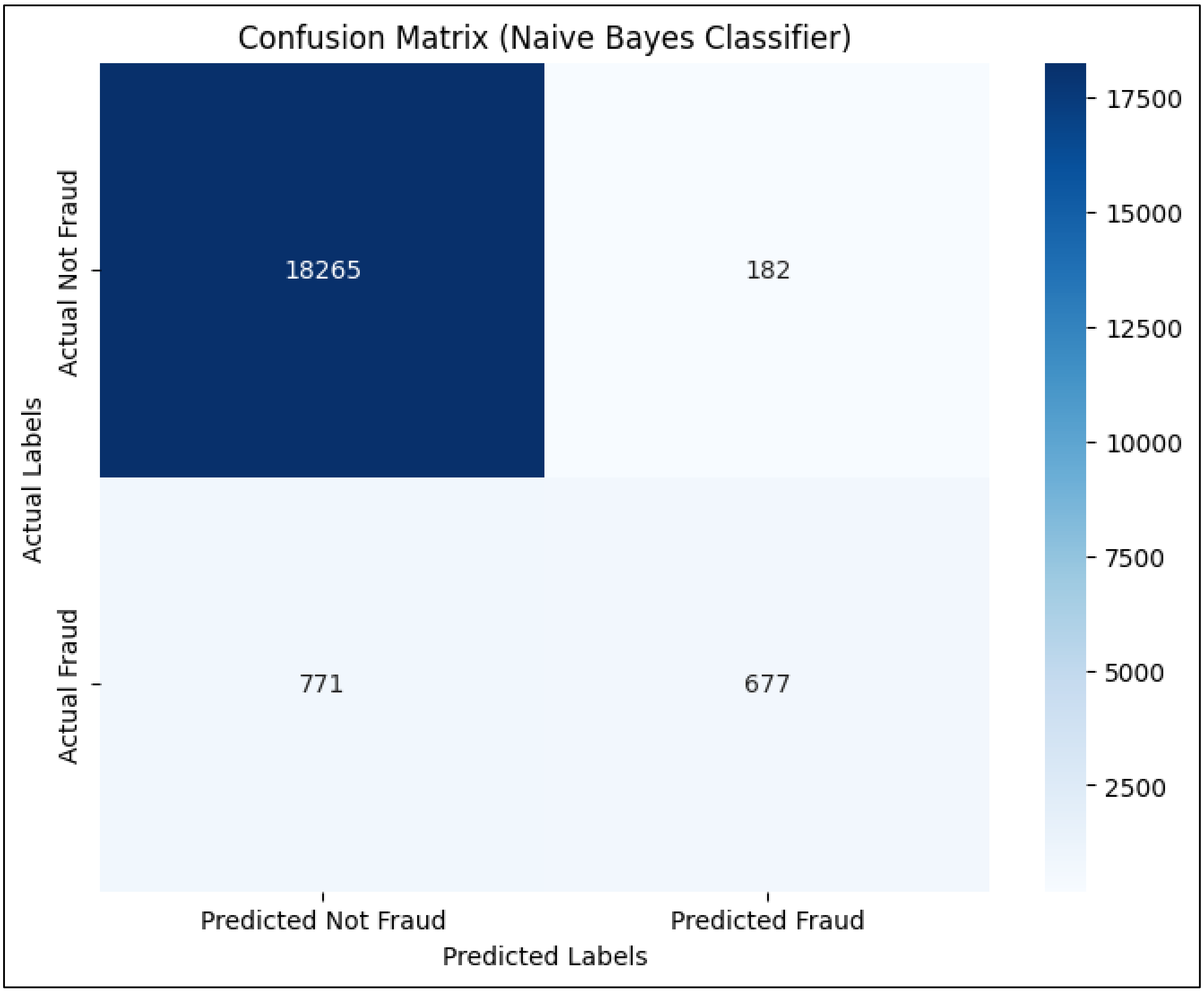
- Step 5 - **Visualize the Confusion Matrix**
  A heatmap is created to visually display the confusion matrix, showing actual vs. predicted fraud cases.

**Code -**



```python
plt.figure(figsize=(8, 6))
sns.heatmap(cm_nb, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted Not Fraud', 'Predicted Fraud'],
            yticklabels=['Actual Not Fraud', 'Actual Fraud'])
plt.title("Confusion Matrix (Naive Bayes Classifier)")
plt.xlabel('Predicted Labels')
plt.ylabel('Actual Labels')
plt.show()
```

**Confusion Matrix -**



This confusion matrix represents the performance of the Naive Bayes classifier in detecting fraud (1) and non-fraud (0) transactions:

1. **True Negatives (Top-Left: 18265)**
   The model correctly predicted 18,265 transactions as "Not Fraud" when they were indeed not fraudulent.
2. **False Positives (Top-Right: 182)**
   The model incorrectly predicted 182 transactions as "Fraud" when they were actually not fraudulent.
3. **False Negatives (Bottom-Left: 771)**
   The model incorrectly predicted 771 transactions as "Not Fraud" when they were actually fraudulent.
4. **True Positives (Bottom-Right: 677)**
   The model correctly predicted 677 transactions as "Fraud" when they were indeed fraudulent.

## Why Choose Naïve Bayes

- **Naive Bayes** can handle this variety: Gaussian Naive Bayes works well with numerical features, and categorical data can be encoded for processing.
- The target variable, Fraud, is binary (0 = Not Fraud, 1 = Fraud).
- Naive Bayes excels in binary classification tasks, especially when paired with imbalanced datasets like this one, where fraudulent cases are typically rarer.

Conclusion – **Naïve Bayes** is chosen because it's fast, works well with mixed feature types, handles binary imbalanced classification effectively, and offers interpretability—all of which are ideal for this dataset and problem.