# Fraud Transaction Detection Using Transactions Dataset

**PROJECT REPORT**

**Machine Learning Intern**

**Submitted By: Aarchi**

**Email Id:** aarchisoorma@gmail.com

**Internship Id:** UMIP273037

# I. INTRODUCTION

Fraud transaction detection is a machine learning project designed to identify fraudulent financial transactions. The rise of digital payments has made it crucial for banks and companies to identify any illegal conduct. Using machine learning techniques, this research examines transaction patterns and creates a predictive model to categorize transactions as either fraudulent or legitimate.

On the basis of past transaction data, the system may identify anomalous activity by utilizing algorithms such as Logistic Regression, Decision Trees, Random Forest, and Boosting techniques. Reducing monetary losses and improving the security of online transactions are the objectives.

Identifying suspicious activity, stopping fraudulent transaction attempts, and reducing risks like chargebacks are the main goals of fraud detection. By analyzing data and identifying anomalies instantly, cutting-edge technologies like machine learning are essential in the fight against fraudulent transactions. This method is particularly important for detecting fraud in online transactions, where the digital nature of payments increases the risk.

## ➢ Real-Life Example of Fraud Transaction Detection

**Case: Credit Card Fraud at an International Retail Store**

John, a frequent shopper from New York, usually makes small purchases at local grocery stores and online marketplaces. One day, his bank detects a **$5,000 purchase in a luxury store in Paris**, while John is still in New York.

**How the Fraud Detection System Works:**

1. **Transaction Monitoring:** The bank's system tracks all transactions in real-time.
2. **Anomaly Detection:**
    - John has never made international transactions.
    - The sudden high-value purchase is unusual.
    - The system flags it as **potential fraud**.
3. **Instant Security Action:**
    - The bank **declines the transaction** temporarily.
    - John receives a security alert via SMS and email: *"Did you attempt a $5,000 purchase in Paris?"*
    - John responds "NO," and the bank **blocks the card** to prevent further fraud.

# II. PROCEDURE

## 1. Python Libraries:

In this project on **fraud transaction detection**, we need various Python libraries to handle data processing, visualization, model building, and evaluation. The following code imports all the necessary libraries:

**a. Data Handling and Processing Libraries:**

- **Pandas (pd):** In this project, pandas help read the transaction dataset (.csv file), perform feature selection, and pre-process the data.
- **NumPy (np):** NumPy is used for handling large arrays and performing mathematical operations.

**b. Data Visualization Libraries:**

- **Seaborn (sns):** Seaborn is useful for creating heatmaps, correlation matrices, and other plots to understand relationships in the dataset.
- **Matplotlib.pyplot (plt):** It is helpful for plotting graphs such as histograms, scatter plots, and trend lines to analyse transaction trends and fraud patterns.

**c. Machine Learning Model Libraries:**

**Scikit-learn (sklearn)** – Used for:

- **Train-test splitting (train_test_split)**
- **Machine learning models**: AdaBoost, Gradient Boosting, Random Forest, Decision Tree, Logistic Regression, Naïve Bayes.
- **Performance evaluation metrics**: Accuracy, Confusion Matrix, F1 Score, Recall, Precision.
- **Data preprocessing**: Label encoding for categorical variables and scaling numerical data.

```
In [1]: import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import AdaBoostClassifier,GradientBoostingClassifier,RandomForestClassifier
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.linear_model import LogisticRegression
        from sklearn.naive_bayes import GaussianNB
        from sklearn.metrics import accuracy_score,confusion_matrix,f1_score,recall_score,precision_score
        from sklearn.preprocessing import LabelEncoder,StandardScaler
```

*Figure1: Python Libraries*

## 2. Loading the datasets:

The code loads the transaction dataset into a Pandas DataFrame. The data.head (5) function displays the first 5 rows, helping to understand the structure, column names, and data distribution. This step is crucial for identifying patterns, missing values, or inconsistencies before pre-processing and model training in fraud detection.

```
In [2]: data=pd.read_csv("Bank_Transaction_Fraud_Detection.csv")
        data.head(10)
```

Out[2]:

| | Customer_ID | Customer_Name | Gender | Age | State | City | Bank_Branch | Account_Type | Transaction_ID | Transaction_Date | ... | Merc |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | d5f6ec07-d69e-4f47-b9b4-7c58ff17c19e | Osha Tella | Male | 60 | Kerala | Thiruvananthapuram | Thiruvananthapuram Branch | Savings | 4fa3208f-9e23-42dc-b330-844829d0c12c | 23-01-2025 | ... | |
| 1 | 7c14ad51-781a-4db9-b7bd-67439c175262 | Hredhaan Khosla | Female | 51 | Maharashtra | Nashik | Nashik Branch | Business | c9de0c06-2c4c-40a9-97ed-3c7b8f97c79c | 11-01-2025 | ... | |
| 2 | 3a73a0e5-d4da-45aa-85f3-528413900a35 | Ekani Nazareth | Male | 20 | Bihar | Bhagalpur | Bhagalpur Branch | Savings | e41c55f9-c016-4ff3-872b-cae72467c75c | 25-01-2025 | ... | |
| 3 | 7902f4ef-9050-4a79-857d-9c2ea3181940 | Yamini Ramachandran | Female | 57 | Tamil Nadu | Chennai | Chennai Branch | Business | 7f7ee11b-ff2c-45a3-802a-49bc47c02ecb | 19-01-2025 | ... | |
| 4 | 3a4bba70-d9a9-4c5f-8b92-1735fd8c19e9 | Kritika Rege | Female | 43 | Punjab | Amritsar | Amritsar Branch | Savings | f8e6ac6f-81a1-4985-bf12-f60967d852ef | 30-01-2025 | ... | |

*Figure 2: Loading the dataset*

## 3. Dropping Irrelevant Columns:

Dropping irrelevant columns helps reduce dimensionality and improve model performance.

- drop ([...], axis=1) removes unnecessary columns that **do not impact fraud detection**:
  - **Identifiers:** Customer_ID, Transaction_ID, Merchant_ID
  - **Personal details:** Customer_Name, Customer_Contact, Customer_Email
  - **Currency Column:** Transaction_Currency (likely irrelevant if all transactions are in INR)

```
In [3]: data=data.drop(["Customer_ID","Customer_Name","Transaction_ID","Customer_Contact","Customer_Email","Merchant_ID","Transaction_Cur
        data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 17 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Gender                  200000 non-null  object
 1   Age                     200000 non-null  int64
 2   State                   200000 non-null  object
 3   City                    200000 non-null  object
 4   Bank_Branch             200000 non-null  object
 5   Account_Type            200000 non-null  object
 6   Transaction_Date        200000 non-null  object
 7   Transaction_Time        200000 non-null  object
 8   Transaction_Amount      200000 non-null  float64
 9   Transaction_Type        200000 non-null  object
 10  Merchant_Category       200000 non-null  object
 11  Account_Balance         200000 non-null  float64
 12  Transaction_Device      200000 non-null  object
 13  Transaction_Location    200000 non-null  object
 14  Device_Type             200000 non-null  object
 15  Is_Fraud                200000 non-null  int64
 16  Transaction_Description 200000 non-null  object
dtypes: float64(2), int64(2), object(13)
memory usage: 25.9+ MB
```

*Figure3: dropping irrelevant columns*

## 4. Statistical Summary of Dataset:

The **data.describe()** function in Pandas provides a **statistical summary** of the numerical columns in the dataset. It helps in understanding the distribution and key characteristics of the data. When executed, it returns the following statistical measures for numerical features:

- **Count**: Total number of non-null values.
- **Mean**: Average value.
- **Standard Deviation (std)**: Spread of data.
- **Min, Max**: Smallest and largest values.
- **25%, 50%, 75% (Quartiles)**: Helps understand data distribution.

**Observations from describe():**

- **Fraud is rare (5% cases labeled 1)**.
- **Transaction amounts vary widely**, with a median around ₹49,500.
- **Account balances also vary**, with most values between ₹28,000 and ₹76,000.

```
In [4]: data.describe()
```

Out[4]:

| | Age | Transaction_Amount | Account_Balance | Is_Fraud |
|---|---|---|---|---|
| count | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 |
| mean | 44.015110 | 49538.015554 | 52437.988784 | 0.050440 |
| std | 15.288774 | 28551.874004 | 27399.507128 | 0.218852 |
| min | 18.000000 | 10.290000 | 5000.820000 | 0.000000 |
| 25% | 31.000000 | 24851.345000 | 28742.395000 | 0.000000 |
| 50% | 44.000000 | 49502.440000 | 52372.555000 | 0.000000 |
| 75% | 57.000000 | 74314.625000 | 76147.670000 | 0.000000 |
| max | 70.000000 | 98999.980000 | 99999.950000 | 1.000000 |

*Figure 4: statistical summary*

## 5. Checking Missing values:

The **isnull().sum()** checks how many missing values exist in each column.
**Observation:**

- Output shows **zero missing values**, meaning data is **clean** and **ready for preprocessing**.

```
In [5]: data.isnull().sum()
Out[5]: Gender                      0
        Age                         0
        State                       0
        City                        0
        Bank_Branch                 0
        Account_Type                0
        Transaction_Date            0
        Transaction_Time            0
        Transaction_Amount          0
        Transaction_Type            0
        Merchant_Category           0
        Account_Balance             0
        Transaction_Device          0
        Transaction_Location        0
        Device_Type                 0
        Is_Fraud                    0
        Transaction_Description     0
        dtype: int64
```

*Figure 5: Checking Missing Values*

## 6. Exploratory Data Analysis:

This code performs **Exploratory Data Analysis (EDA)** by visualizing the distribution and presence of outliers in all numerical columns using a **boxplot**.

1. **Selecting Numerical Columns:**
   - **data.select_dtypes(include=['number'])** filters the dataset to include only numerical features such as transaction amount, balance, and timestamps.
2. **Creating a Boxplot:**
   - **sns.boxplot(data=numerical_cols)** generates a **boxplot**, which is useful for detecting **outliers, spread, and skewness** in numerical features.
   - The **boxplot** shows the **median, interquartile range (IQR), and potential outliers** beyond the whiskers.
3. **Figure Customization:**
   - **plt.figure(figsize=(12,6))** adjusts the figure size for better readability.
   - **plt.xticks(rotation=90)** rotates the x-axis labels to **prevent overlap** and improve clarity.
   - **plt.title("Boxplot of All Numerical Columns")** sets a meaningful title for the plot.
   - **plt.show()** displays the visualization.

**Observation:**

- **Outliers Detection:** Fraudulent transactions may show extreme outliers in transaction amounts, balances, or frequencies.
- **Feature Distribution:** Helps understand the spread of transaction amounts, identifying possible skewness.
- **Data Cleaning Needs:** Identifies potential issues like missing values or incorrectly scaled features.

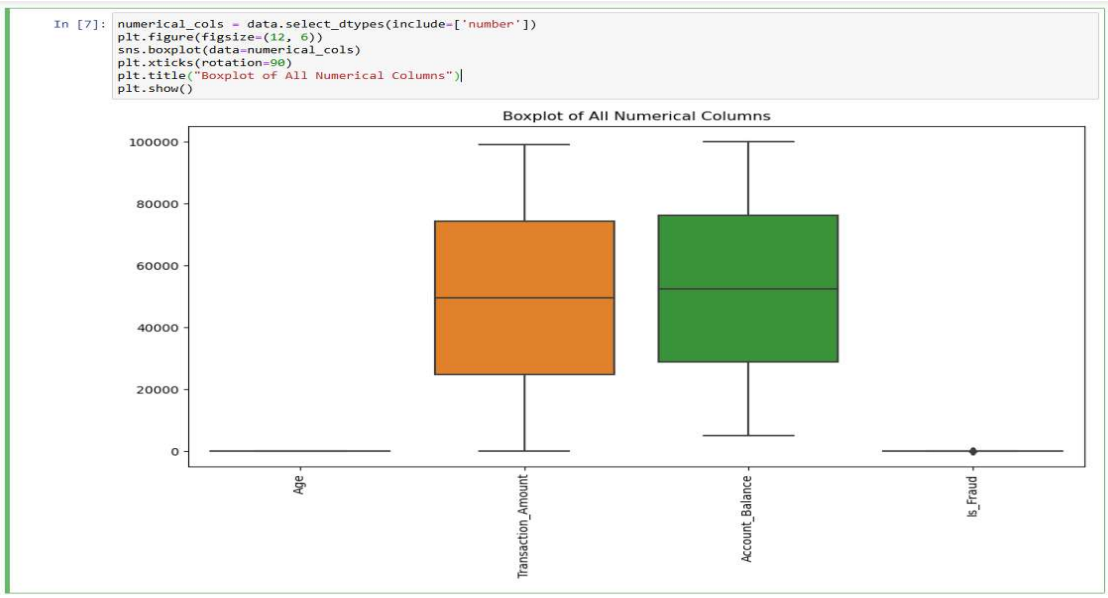


*Figure 6: EDA*

## 7. Histogram Visualization of Numerical Features:

This code generates **histograms** for all numerical columns to analyse their distribution and frequency:

1. **Selecting Numerical Columns:**
   - **data.select_dtypes(include=['number'])** extracts only numerical columns from the dataset, such as transaction amounts, balances, and timestamps.
2. **Setting Up the Grid for Subplots:**
   - **fig, axes = plt.subplots(nrows=num_cols // 3 + 1, ncols=3, figsize=(15, num_cols * 2.5))**
   - Creates a grid layout to plot multiple histograms at once, arranging them in **3 columns per row**.
   - The number of rows is calculated dynamically based on the number of numerical features.
3. **Looping Through Numerical Features:**
   - The **for** loop iterates through each numerical column and generates a histogram:
     - **axes[i].hist(data[col], bins=20, color='skyblue', edgecolor='black')**
     - Uses **20 bins** to distribute values, with a sky-blue color and black edges for clarity.
     - **set_title(), set_xlabel(),** and **set_ylabel()** add appropriate labels.
4. **Removing Empty Subplots (If Needed):**

- If the number of numerical columns is not a multiple of 3, extra empty subplots are deleted using **fig.delaxes(axes[i]).**

5. **Displaying the Plots:**
   - **plt.tight_layout()** adjusts spacing for better visualization.
   - **plt.show()** displays all the histograms.

```
In [8]: numerical_cols = data.select_dtypes(include=['number'])

        num_cols = len(numerical_cols.columns)
        fig, axes = plt.subplots(nrows=num_cols // 3 + 1, ncols=3, figsize=(15, num_cols * 2.5))
        axes = axes.flatten()
        for i, col in enumerate(numerical_cols.columns):
            axes[i].hist(data[col], bins=20, color='skyblue', edgecolor='black')
            axes[i].set_title(col)
            axes[i].set_xlabel("Value")
            axes[i].set_ylabel("Frequency")

        for i in range(num_cols, len(axes)):
            fig.delaxes(axes[i])

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

*Figure 7: Histogram Visualization*

- **Understanding Data Distribution:** Identifies whether features are **normally distributed, skewed, or contain outliers**.
- **Detecting Fraudulent Patterns:** Unusual peaks or heavily skewed distributions might indicate suspicious activity.
- **Feature Engineering:** Helps decide whether **scaling, transformation, or outlier handling** is necessary for machine learning models.

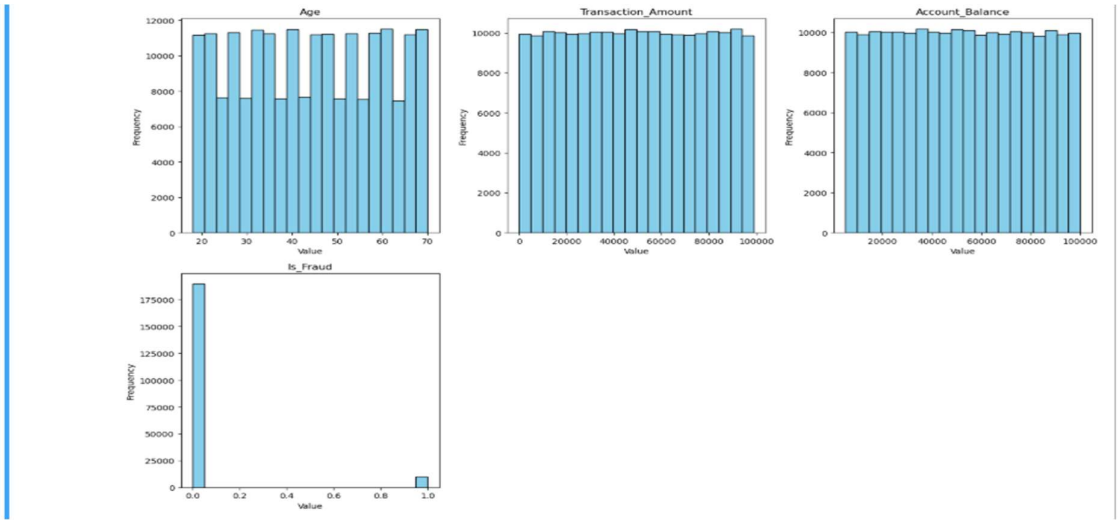


*Figure 8: Histogram Output*

## 8. Encoding Categorical Features using Label Encoding:

In this code cell, we performed **Label Encoding** to transform categorical data into numerical values, making it suitable for machine learning models. First, we identified categorical columns in the dataset using **data.select_dtypes(include=['object']).columns**, which selects all columns containing text-based values such as transaction type, merchant category, or fraud labels. Then, we initialized an empty dictionary, **label_encoders = {}**, to store the encoders for each categorical feature.

Next, we iterated through each categorical column, applied **LabelEncoder()**, and replaced the original text values with numerical labels. Each encoder was stored in **label_encoders[col]**

for future reference, ensuring that we can decode the labels later if needed. This step is essential for preparing the dataset for machine learning, as models require numerical inputs to process data efficiently. Label encoding ensures that categorical features like transaction type and location can contribute to fraud detection without causing compatibility issues in model training.

```
In [9]: categorical_cols = data.select_dtypes(include=['object']).columns
        label_encoders = {}
        for col in categorical_cols:
            le = LabelEncoder()
            data[col] = le.fit_transform(data[col])
            label_encoders[col] = le
```

```
In [10]: data.head(5)
```

Out[10]:

| | Gender | Age | State | City | Bank_Branch | Account_Type | Transaction_Date | Transaction_Time | Transaction_Amount | Transaction_Type |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 60 | 15 | 127 | 127 | 2 | 22 | 52151 | 32415.45 | 3 |
| **1** | 0 | 51 | 18 | 100 | 100 | 0 | 10 | 55958 | 43622.60 | 0 |
| **2** | 1 | 20 | 4 | 13 | 13 | 2 | 24 | 10257 | 63062.56 | 0 |
| **3** | 0 | 57 | 28 | 22 | 22 | 0 | 18 | 40370 | 14000.72 | 2 |
| **4** | 0 | 43 | 25 | 7 | 7 | 2 | 29 | 60052 | 18335.16 | 3 |

*Figure 9: Label Encoding*

**Head():**

The function **data.head(5)** displays the **first five rows** of the modified dataset after applying **Label Encoding**. This helps us verify that categorical features have been successfully converted into numerical values and allows for an initial inspection of the dataset structure.

**Observations:**

1. **Categorical Columns are Encoded:** Previously text-based columns (e.g., transaction type, merchant category) are now represented as numbers.
2. **Dataset Structure is Preserved:** The dataset retains the same number of rows and columns, but categorical features are now numerical.
3. **Potential Data Patterns:** We can start analyzing trends in transaction amounts, balances, and fraud labels.
4. **Missing or Incorrect Data:** By viewing the first few rows, we can check for anomalies such as missing values or incorrectly encoded data.

## 9. Correlation Heatmap for Feature Relationships:

The function data.corr() computes the **correlation matrix** of numerical columns in the dataset, showing how strongly different features are related to each other. The sns.heatmap() function from Seaborn then visualizes this correlation matrix as a **heatmap**, making it easier to identify patterns and relationships.

**Observations for Correlation Heatmap:**

1. **Diagonal Line Represents Perfect Correlation (1.0):**
   - The white diagonal line indicates that each feature has a perfect correlation with itself.
2. **Weak Correlation Between Most Features:**
   - The majority of the heatmap is dark-colored, indicating weak or no correlation between most features.
   - This suggests that features are relatively independent of each other.

3. **Few Strong Correlations Identified:**
    - Some lighter or white blocks outside the diagonal indicate a stronger correlation between certain features.
    - For example, features like **Transaction_Amount, Account_Balance, and Transaction_Time** might show some relationship with other financial attributes.
4. **Potentially Redundant Features:**
    - If two features have a very high correlation (close to 1), one of them might be redundant and could be removed to reduce dimensionality.
5. **Fraud Detection Analysis:**
    - The correlation of **"Is_Fraud"** with other features is crucial.
    - If it shows a strong correlation with specific features (e.g., transaction amount, merchant category, or transaction device), those features are valuable predictors for fraud detection.
6. **Negative Correlations:**
    - Darker negative-colored blocks suggest inverse relationships, meaning when one feature increases, the other tends to decrease.

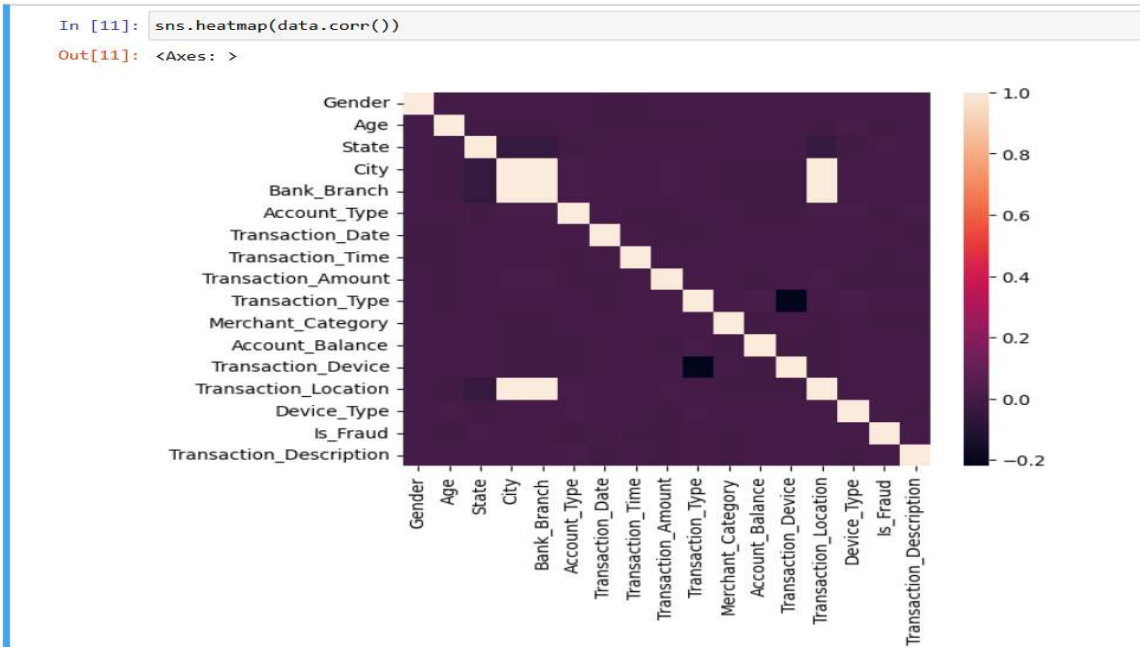


*Figure 10: Correlation Heatmap for feature relationships*

## 10. Correlation Matrix Analysis:

The data.corr() function computes the correlation coefficients between all numerical features in the dataset. Correlation values range from **-1 to 1**, where:

- **1** indicates a perfect positive correlation (as one variable increases, the other also increases).
- **-1** indicates a perfect negative correlation (as one variable increases, the other decreases).
- **0** means no correlation between the variables.

**Observations:**

1. **Weak Correlation Across Features:**
    - Most correlation values are close to **0**, indicating weak relationships between variables.

- No highly correlated pairs suggest minimal redundancy in the dataset.

2. **"Is_Fraud" Feature Shows Minimal Correlation:**
   - The target variable **"Is_Fraud"** has low correlation values with all features, implying that fraud detection may rely on complex interactions rather than single-variable relationships.
3. **High Correlation Between Some Features:**
   - Features like **"City"** and **"Transaction_Location"** show a strong positive correlation, suggesting potential redundancy.
4. **Feature Selection Implications:**
   - Since no highly correlated features exist, dimensionality reduction techniques like PCA or feature elimination may not be necessary.
   - Further feature engineering or model-based feature selection might be required to improve fraud detection performance.

```
In [12]: data.corr()
```

Out[12]:

| | Gender | Age | State | City | Bank_Branch | Account_Type | Transaction_Date | Transaction_Time | Transaction_Amount | Trans |
|---|---|---|---|---|---|---|---|---|---|---|
| **Gender** | 1.000000 | 0.001692 | 0.002899 | -0.000299 | -0.000299 | 0.001260 | -0.003731 | -0.001941 | 0.001468 | |
| **Age** | 0.001692 | 1.000000 | -0.004638 | -0.002284 | -0.002284 | -0.000281 | -0.001401 | -0.001294 | -0.003087 | |
| **State** | 0.002899 | -0.004638 | 1.000000 | -0.046278 | -0.046278 | -0.002314 | 0.002415 | 0.000640 | 0.002480 | |
| **City** | -0.000299 | -0.002284 | -0.046278 | 1.000000 | 1.000000 | 0.007245 | 0.002291 | 0.000812 | 0.005674 | |
| **Bank_Branch** | -0.000299 | -0.002284 | -0.046278 | 1.000000 | 1.000000 | 0.007245 | 0.002291 | 0.000812 | 0.005674 | |
| **Account_Type** | 0.001260 | -0.000281 | -0.002314 | 0.007245 | 0.007245 | 1.000000 | 0.002025 | -0.001731 | -0.004737 | |
| **Transaction_Date** | -0.003731 | -0.001401 | 0.002415 | 0.002291 | 0.002291 | 0.002025 | 1.000000 | 0.000642 | -0.001282 | |
| **Transaction_Time** | -0.001941 | -0.001294 | 0.000640 | 0.000812 | 0.000812 | -0.001731 | 0.000642 | 1.000000 | 0.001743 | |
| **Transaction_Amount** | 0.001468 | -0.003087 | 0.002480 | 0.005674 | 0.005674 | -0.004737 | -0.001282 | 0.001743 | 1.000000 | |
| **Transaction_Type** | 0.001339 | -0.001291 | -0.000130 | 0.000101 | 0.000101 | 0.001876 | -0.001170 | -0.000269 | 0.001788 | |
| **Merchant_Category** | 0.002233 | -0.000381 | 0.001078 | -0.001047 | -0.001047 | 0.000169 | 0.002922 | 0.000318 | -0.000628 | |
| **Account_Balance** | -0.000392 | 0.000269 | 0.000136 | -0.002628 | -0.002628 | -0.001506 | 0.001641 | 0.002224 | -0.001735 | |
| **Transaction_Device** | 0.002109 | 0.002429 | 0.001103 | -0.001441 | -0.001441 | -0.003000 | 0.002933 | 0.002040 | -0.000665 | |
| **Transaction_Location** | -0.000261 | -0.002291 | -0.044936 | 0.999910 | 0.999910 | 0.007216 | 0.002271 | 0.000809 | 0.005656 | |
| **Device_Type** | -0.000534 | 0.004997 | -0.001835 | -0.000052 | -0.000052 | 0.004613 | -0.000271 | -0.000822 | -0.001315 | |
| **Is_Fraud** | 0.000649 | -0.001517 | 0.005716 | 0.002800 | 0.002800 | -0.002592 | -0.000135 | -0.001909 | -0.002100 | |
| **Transaction_Description** | -0.000869 | -0.000346 | 0.001027 | 0.001547 | 0.001547 | 0.005007 | -0.003088 | -0.000427 | -0.003523 | |

*Figure 11: Correlation Matrix Analysis*

## 11. Classification Models:

In fraud detection, various machine learning classifiers are used to distinguish fraudulent transactions from legitimate ones. Below is a brief explanation of the classifiers defined in the models dictionary:

```
In [13]: models={
             "Random Forest Classifier":RandomForestClassifier(),
             "Decision Tree Classifier":DecisionTreeClassifier(),
             "Ada Boost Classifier":AdaBoostClassifier(),
             "Logistic Regression":LogisticRegression(),
             "Gradiant Boosting Classifier":GradientBoostingClassifier(),
             "Naive Bayes Classifier":GaussianNB()
         }
```

*Figure 12: Classifier*

1. **Random Forest Classifier (RandomForestClassifier())**
   - An ensemble learning method that combines multiple decision trees to improve accuracy and reduce overfitting.
   - Works well with large datasets and captures complex patterns in transaction data.
2. **Decision Tree Classifier (DecisionTreeClassifier())**

- A tree-based model that splits data based on feature conditions to classify transactions as fraud or non-fraud.
- Simple and interpretable but prone to overfitting if not properly tuned.

3. **AdaBoost Classifier (AdaBoostClassifier())**
    - An ensemble method that builds a strong classifier by combining multiple weak learners (often decision trees).
    - Assigns higher weights to misclassified instances, improving model performance over iterations.
4. **Logistic Regression (LogisticRegression())**
    - A statistical model that predicts probabilities based on a linear relationship between input features and fraud likelihood.
    - Works well with structured numerical data but may struggle with complex fraud patterns.
5. **Gradient Boosting Classifier (GradientBoostingClassifier())**
    - An advanced boosting technique that sequentially improves weak models by minimizing errors from previous iterations.
    - Effective for handling imbalanced datasets and detecting fraud through feature importance analysis.
6. **Naïve Bayes Classifier (GaussianNB())**
    - A probabilistic classifier based on Bayes' theorem, assuming feature independence.
    - Suitable for text-based or categorical fraud detection but may not capture complex dependencies in transaction data.

## 12. Data Splitting and Feature Scaling for Fraud Detection:

In this code segment, we are preparing the dataset for training and testing machine learning models.

- **Feature and Target Separation**
    - X contains all features except "Is_Fraud", which is the target variable (y).
- **Train-Test Split**
    - The dataset is split into **75% training** and **25% testing** for model evaluation.
- **Feature Scaling (Standardization)**
    - Standardizes numerical values to a **mean of 0 and standard deviation of 1** for better model performance.
- This Helps prevent bias towards larger numerical values.
- Ensures models like Logistic Regression and Gradient Boosting perform efficiently.
- Standardization maintains consistency across training and testing data.

```
In [14]: x=data.drop(["Is_Fraud"],axis=1)
         y=data["Is_Fraud"]
         x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=42)
         ss=StandardScaler()
         x_train=ss.fit_transform(x_train)
         x_test=ss.transform(x_test)
```

*Figure 13: Data Splitting and Feature Scaling*

## 13. Model Training and Evaluation:

Model training and evaluation are essential steps in developing a reliable fraud detection system. These steps ensure that the machine learning models generalize well to new data and effectively identify fraudulent transactions.

During the training phase, machine learning models learn patterns from the historical transaction data. The dataset is first split into training and testing sets, with the training set used to fit the model. The training data contains both fraudulent and non-fraudulent transactions, allowing the models to understand the underlying trends and relationships between features. After training, the models are tested on the unseen test data to evaluate their performance.

1. **Initializing Metrics Dictionary:**
   - The model name, accuracy, precision, and recall are stored in a dictionary metrics.
2. **Model Training & Prediction:**
   - Fit(x_train, y_train) is used to train each model.
   - Predict(x_test) is used to make predictions on the test set.
3. **Assessment of Performance:**
   - Computes evaluation metrics:
     - Accuracy: Indicates general accuracy.
     - Precision: Determines the proportion of anticipated fraud cases that turn out to be fraud.
     - Recall: Evaluates the capacity to identify instances of fraud.
4. **Storing & Printing Results:**
   - The dictionary contains the calculated metrics.
   - The precision of each model is printed.

```
In [ ]: metrics = {
            'Model': [],
            'Accuracy': [],
            'Precision': [],
            'Recall':[]
        }
        for name,model in models.items():
            model.fit(x_train,y_train)
            y_pred=model.predict(x_test)
            acc = accuracy_score(y_test, y_pred)
            f1 = f1_score(y_test, y_pred)
            recall=recall_score(y_test,y_pred)
            precision=precision_score(y_test,y_pred)
            metrics['Model'].append(name)
            metrics['Accuracy'].append(acc * 100)
            metrics['Precision'].append(precision * 100)
            metrics['Recall'].append(recall * 100)
            print(f"Model: {name}")
            print(f"Accuracy: {acc*100}")
            print("-" * 30)
```

*Figure 14: Model Training and Evaluation*

**Output Observation of Model Training and Evaluation:**

Output displays the results of multiple machine learning models used for fraud detection. The script trains and evaluates different classifiers, including Random Forest, Decision Tree, AdaBoost, and Logistic Regression, and prints their accuracy scores.

**Accuracy Scores:**

- **Random Forest Classifier: 94.88%**
- **Decision Tree Classifier: 88.77%**
- **AdaBoost Classifier: 94.88%**
- **Logistic Regression: 94.88%**
- **Naïve Bayes Classifier: 94.88%**

These accuracy values indicate that most models are performing well, with Random Forest, AdaBoost, and Logistic Regression achieving high accuracy. However, accuracy alone is not sufficient for evaluating fraud detection performance, as false negatives (missed fraud cases) are critical.

```
C:\Users\lenovo\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision is ill-d
efined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

Model: Random Forest Classifier
Accuracy: 94.88600000000001
--------------------------------
Model: Decision Tree Classifier
Accuracy: 88.69200000000001
--------------------------------
Model: Ada Boost Classifier
Accuracy: 94.882
--------------------------------
Model: Logistic Regression
Accuracy: 94.88600000000001
--------------------------------

C:\Users\lenovo\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision is ill-d
efined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

Model: Gradiant Boosting Classifier
Accuracy: 94.884
--------------------------------
Model: Naive Bayes Classifier
Accuracy: 94.88600000000001
--------------------------------

C:\Users\lenovo\anaconda3\Lib\site-packages\sklearn\metrics\_classification.py:1469: UndefinedMetricWarning: Precision is ill-d
efined and being set to 0.0 due to no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

*Figure 15: Output of Model Evaluation*

## 14. Visualization of Model Accuracy Comparisons:

Here we have created a **bar chart** to compare the accuracy of different machine learning models used for fraud detection.

- **metrics_df = pd.DataFrame(metrics)**: Converts the collected model evaluation metrics into a DataFrame.
- **plt.figure(figsize=(10, 6))**: Sets the figure size for better visibility.
- **plt.bar()**: Plots a bar chart with models on the x-axis and their accuracy (%) on the y-axis.
- **plt.xticks(rotation=45)**: Rotates model names for readability.
- **plt.ylim(0, 100)**: Ensures the accuracy values stay within a 0-100% range.
- **plt.grid(axis='y', linestyle='--', alpha=0.7)**: Adds a dashed gridline for better readability.
- **plt.show()**: Displays the final chart.

```
In [16]: metrics_df = pd.DataFrame(metrics)
         plt.figure(figsize=(10, 6))
         plt.bar(metrics_df['Model'], metrics_df['Accuracy'], color='skyblue', width=0.6)
         plt.title('Model Accuracy Comparison', fontsize=16)
         plt.ylabel('Accuracy (%)', fontsize=14)
         plt.xlabel('Model', fontsize=14)
         plt.xticks(rotation=45)
         plt.ylim(0, 100)
         plt.grid(axis='y', linestyle='--', alpha=0.7)
         plt.show()
```

*Figure 16: Model Accuracy*

This bar chart is essential for assessing how well the model detects fraudulent transactions.

1. **Model Comparison Based on Accuracy**
   - Several machine learning models **(Random Forest, Decision Tree, AdaBoost, Logistic Regression, etc.)** are graphically compared in the chart according to their accuracy.
   - A higher accuracy indicates that more fraudulent and non-fraudulent transactions are accurately classified by the model.
   - It helps identify the most reliable model for fraud detection.
2. **Decision Making for Model Selection**
   - A highly effective model that reduces false negatives—fraud situations categorized as non-fraud—is necessary for fraud detection.
   - A model may not be able to differentiate between fraudulent and authentic transactions if its accuracy is low.
   - This helps in selecting the best-performing model for deployment in real-world fraud detection systems.
3. **Identifying the Need for Further Evaluation**
   - Accuracy alone isn't enough for fraud detection, as fraudulent transactions are often rare (**class imbalance problem**).
   - If all models have high accuracy but low precision/recall, the system may still **miss actual fraud cases**.
   - This visualization helps in deciding whether further evaluation using **Precision, Recall, and F1-score** is necessary.
4. **Improving Fraud Detection Models**
   - If the accuracy is **not satisfactory**, it indicates the need for **feature engineering, hyperparameter tuning, or using a different algorithm** to improve fraud detection performance.
   - The insights gained from this visualization guide the next steps in optimizing the fraud detection system.

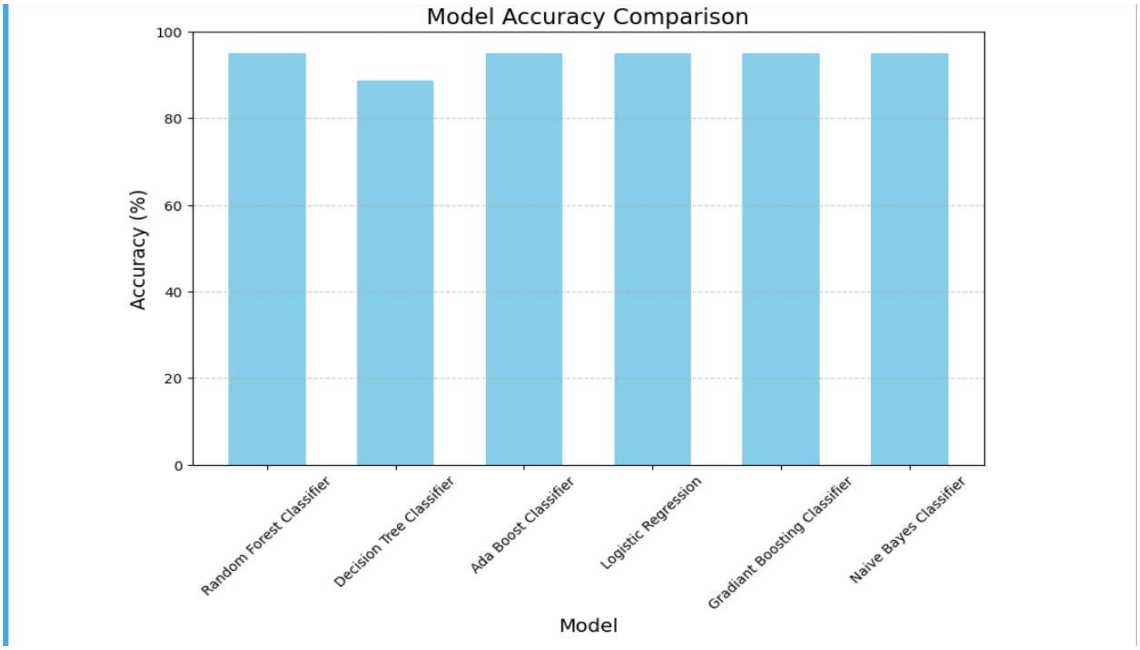


*Figure 17: Visualization of Model Accuracy*

# III. **RESULT**

After implementing multiple machine learning models for fraud detection, the final output consists of:

1. **Model Performance Evaluation:**

The evaluation metrics for different classifiers, including **Accuracy, Precision, and Recall**, help in determining the best model for fraud detection. The models tested include:

- **Random Forest Classifier: 94.88%**
- **Decision Tree Classifier: 88.77%**
- **AdaBoost Classifier: 94.88%**
- **Logistic Regression: 94.88%**
- **Naïve Bayes Classifier: 94.88%**
- **Ada Boost Classifier: 94.88%**

The **best model** is selected based on **higher accuracy, recall, and precision** scores.

2. **Fraud Detection Predictions**

After training, the **selected model** is used to predict whether a transaction is fraudulent or not.

- **Input:** Transaction details (Amount, Location, Device Type, etc.)
- **Output:**
  - **0** → Legitimate Transaction
  - **1** → Fraudulent Transaction