2024

# Financial Fraud Detection using Machine Learning

Capstone Project - Classification

**DIBOSH BARUAH**

## 1. Introduction

Financial fraud detection is a critical problem in banking and payment systems, where the primary objective is to identify fraudulent transactions in real-time to minimize financial losses and protect users. Fraudulent transactions can lead to significant financial harm, so detecting them early is crucial for both financial institutions and their customers.

For this project, we used a dataset consisting of 6 million transaction records from a bank's transaction system. Each record includes several features such as the transaction type, transaction amount, and the balance before and after the transaction. The target variable, `isFraud`, indicates whether a transaction is fraudulent (1) or non-fraudulent (0). The goal of this project was to build a machine learning model that could predict fraud with high accuracy, using XGBoost, a gradient boosting machine learning algorithm known for its performance and efficiency in classification problems.

---

## 2. Data Collection

The data for this project was collected from a financial transaction system in the form of a relational database, specifically SQLite format, containing records of transactions. The dataset provided includes several key features that help describe the transaction details:

- **Step**: The time step of the transaction (each transaction occurs on a different step, which essentially represents a day).
- **Type**: The type of the transaction (e.g., PAYMENT, TRANSFER, CASH-IN, CASH-OUT).
- **Amount**: The amount of money involved in the transaction.
- **OldBalanceOrg**: The balance of the origin account before the transaction occurred.
- **NewBalanceOrg**: The balance of the origin account after the transaction.

Additionally, the dataset includes fraud labels indicating whether a transaction was fraudulent or not (the `isFraud` column). Once collected, the dataset was loaded into a Pandas DataFrame for further analysis and processing.

## 3. Data Preprocessing

Before feeding the data into the machine learning model, several preprocessing steps were necessary to clean and transform the data:

### 3.1 Handling Missing Values

The first step was to check if there were any missing values in the dataset. For the dataset used in this project, there were no missing values, so no imputation was required. This ensured that we could directly work with the full dataset.

### 3.2 Feature Engineering

The dataset contains both categorical and numerical features. We performed the following transformations:

- **Categorical Variables**: The Type feature was one-hot encoded into multiple binary columns, where each type of transaction (e.g., PAYMENT, TRANSFER, etc.) becomes a separate feature.
- **Numerical Features**: Features such as Amount, OldBalanceOrg, and NewBalanceOrg were already in numerical form, so no transformation was required for these variables.

Additionally, we created new features such as the transaction balance change by subtracting OldBalanceOrg from NewBalanceOrg. This additional feature provided more insight into the financial activity surrounding each transaction.

### 3.3 Feature Scaling

Although XGBoost can handle raw, unscaled data efficiently, we performed standardization on features like Amount, OldBalanceOrg, and NewBalanceOrg to ensure that no feature would disproportionately influence the model due to its scale.

## 4. Data Collection & Feature Selection

### 4.1 Feature Inspection & Selection

After exploring the dataset, the next step was to identify and select the most relevant features for the model. The dataset initially contained 14 features, including the target variable isFraud. Key steps in the feature selection process included:

- **Identifying Redundant Features**: After inspecting the dataset, we noted that some features, such as OldBalanceOrg and NewBalanceOrg, were closely related, with their difference representing the amount involved in the transaction.
- **Removing Irrelevant Features**: Features like nameOrig and nameDest (representing the account names) were found to be non-contributory to fraud detection and were removed.

- **Feature Importance**: Using XGBoost's feature importance techniques, we identified key features like Amount, OldBalanceOrg, NewBalanceOrg, and Type as the most influential predictors.

### 4.2 Feature Engineering

In addition to the original features, new features were engineered to potentially improve the model performance:

- **Balance Change**: A feature capturing the difference between NewBalanceOrg and OldBalanceOrg, representing the financial impact of the transaction.
- **Transaction Type Encodings**: One-hot encoding was applied to the Type feature, transforming it into binary columns for each transaction type (e.g., PAYMENT, TRANSFER, CASH-IN, CASH-OUT).

### 4.3 Handling Class Imbalance

Given the highly imbalanced dataset (with fraudulent transactions being much fewer than non-fraudulent ones), we used **SMOTE** (Synthetic Minority Over-sampling Technique) to generate synthetic samples for the minority class (fraudulent transactions). This technique helped to improve the model's ability to detect fraud.

---

## 5. Model Building

For the predictive model, we chose **XGBoost**, a powerful gradient boosting algorithm known for its performance and efficiency in classification problems. The model was trained on the processed data and evaluated using cross-validation.

### 5.1 Model Evaluation Metrics

We used the following evaluation metrics for this binary classification problem:

- **Accuracy**
- **Precision**
- **Recall**
- **F1-Score**
- **AUC-ROC Curve**

The **AUC (Area Under Curve)** was particularly important for assessing the model's ability to distinguish between fraudulent and non-fraudulent transactions.

## 6. Results & Discussion

### 6.1 Model Performance

The XGBoost model performed exceptionally well across all metrics:

- **Accuracy**: 1.00 (Perfect accuracy, correctly classifying all non-fraudulent transactions and most fraudulent transactions)
- **Precision**:
  - **Class 0 (Non-fraudulent transactions)**: 1.00 (Perfect precision for class 0, meaning almost all predicted non-fraudulent transactions were correct)
  - **Class 1 (Fraudulent transactions)**: 0.97 (97% of predicted fraudulent transactions were indeed fraudulent)
- **Recall**:
  - **Class 0 (Non-fraudulent transactions)**: 1.00 (Perfect recall for class 0, meaning all non-fraudulent transactions were correctly identified)
  - **Class 1 (Fraudulent transactions)**: 0.68 (68% of all fraudulent transactions were correctly identified)
- **F1-Score**: 0.80 (The overall F1-Score, a weighted average of both class F1 scores, reflects strong performance but highlights a slight trade-off due to the lower recall for class 1)
- **ROC AUC**: 0.9996 (Indicates near-perfect ability to distinguish between fraudulent and non-fraudulent transactions)

**Confusion Matrix**:

- True negatives (TN): 999,436 (Non-fraudulent transactions correctly predicted as non-fraudulent)
- False positives (FP): 10 (Non-fraudulent transactions incorrectly predicted as fraudulent)
- False negatives (FN): 179 (Fraudulent transactions incorrectly predicted as non-fraudulent)
- True positives (TP): 375 (Fraudulent transactions correctly predicted as fraudulent)

**Precision-Recall AUC**: 0.8305 (Shows strong performance for both precision and recall, though with some room for improvement on recall for class 1).

### 6.2 Feature Importance

The most important features contributing to the model's decision-making process are:

- **newbalanceDest**: 0.1754
- **newbalanceOrig**: 0.1711
- **type_CASH_OUT**: 0.1477
- **type_CASH_IN**: 0.1252
- **oldbalanceOrg**: 0.1044
- **type_PAYMENT**: 0.0813
- **oldbalanceDest**: 0.0566
- **amount**: 0.0551
- **step**: 0.0384
- **type_TRANSFER**: 0.0361
- **type_DEBIT**: 0.0086

These features were found to be the most influential in predicting fraudulent transactions, with newbalanceDest, newbalanceOrig, and type_CASH_OUT emerging as the top contributors.

---

## 7. Conclusion

The project demonstrates the effectiveness of XGBoost for predicting fraudulent financial transactions. By preprocessing the data, engineering new features, and handling class imbalance, the model achieved excellent performance, with near-perfect accuracy and strong precision for non-fraudulent transactions.

While the model showed high performance overall, there is still room for improvement, especially in increasing the recall for fraudulent transactions. Future work could explore more advanced techniques for handling rare fraud patterns, anomaly detection, and leveraging ensemble methods to further enhance the model's predictive accuracy.

Deploying such a model in real-world banking systems would significantly reduce fraud-related losses and help protect customers from financial crimes.

---

## 8.Original Code:

```
# Original Coding process

import sqlite3
import pandas as pd

# Connecting to database.db
conn = sqlite3.connect('/Users/diboshbaruah/Desktop/Database.db')
data = pd.read_sql_query('SELECT * FROM Fraud_Detection', conn)

print("Displaying first few rows of the dataset")
print()
print(data.head())
print()

# Checking for data types
print("Data types before conversion:")
print(data.dtypes)

# Closing the connection
conn.close()
```

```
Displaying first few rows of the dataset

  step      type    amount     nameOrig oldbalanceOrg newbalanceOrig  \
0    1   PAYMENT   9839.64  C1231006815      170136.0      160296.36
1    1   PAYMENT   1864.28  C1666544295       21249.0       19384.72
2    1  TRANSFER    181.0  C1305486145         181.0            0.0
3    1  CASH_OUT    181.0   C840083671         181.0            0.0
4    1   PAYMENT  11668.14  C2048537720       41554.0       29885.86

      nameDest oldbalanceDest newbalanceDest isFraud isFlaggedFraud
0  M1979787155            0.0            0.0       0              0
1  M2044282225            0.0            0.0       0              0
2   C553264065            0.0            0.0       1              0
3    C38997010        21182.0            0.0       1              0
4  M1230701703            0.0            0.0       0              0
```

```
Data types before conversion:
step              object
type              object
amount            object
nameOrig          object
oldbalanceOrg     object
newbalanceOrig    object
nameDest          object
oldbalanceDest    object
newbalanceDest    object
isFraud           object
isFlaggedFraud    object
dtype: object
```

```
# Data Pre-processing

# List of numeric columns that needs to be converted to numeric types
numeric_columns = ['amount', 'oldbalanceOrg', 'newbalanceOrig',
'oldbalanceDest', 'newbalanceDest']

# Converting the columns listed in `numeric_columns` to numeric type using
pandas `apply` method
# `errors='coerce'` will convert any non-numeric values to NaN
data[numeric_columns] = data[numeric_columns].apply(pd.to_numeric,
errors='coerce')

# Converting the 'step' column to numeric type.
data['step'] = pd.to_numeric(data['step'], errors='coerce')

# Droping the columns 'nameOrig' and 'nameDest' as they represent unique
identifiers
data.drop(columns=['nameOrig', 'nameDest'], inplace=True)

# Converting the categorical column 'type' to one-hot encoded features.
data = pd.get_dummies(data, columns=['type'], drop_first=True)

# Converting the new binary columns created from 'type' to integer type (1 or
0)
type_columns = ['type_CASH_IN', 'type_CASH_OUT', 'type_DEBIT',
'type_PAYMENT', 'type_TRANSFER']
data[type_columns] = data[type_columns].astype(int)

# Converting the 'isFraud' column to integer type to represent whether the
transaction is fraudulent (1) or not (0).
data['isFraud'] = data['isFraud'].astype(int)
```

```
# Converting the 'isFlaggedFraud' column to integer type to represent whether
the transaction is flagged as potentially fraudulent (1) or not (0).
data['isFlaggedFraud'] = data['isFlaggedFraud'].astype(int)

print("Data Pre-processing completed successfully...")
```

```
Data Pre-processing completed successfully...
```

```
print("Displaying first few rows of the dataset post conversion")
print()
print(data.head())

# Checking the data types again after encoding
print("\nData types after Encoding:")
print(data.dtypes)
```

```
Displaying first few rows of the dataset post conversion

   step    amount  oldbalanceOrg  newbalanceOrig  oldbalanceDest  \
0     1   9839.64       170136.0       160296.36             0.0
1     1   1864.28        21249.0        19384.72             0.0
2     1    181.00          181.0            0.00             0.0
3     1    181.00          181.0            0.00         21182.0
4     1  11668.14        41554.0        29885.86             0.0

   newbalanceDest  isFraud  isFlaggedFraud  type_CASH_IN  type_CASH_OUT  \
0             0.0        0               0             0              0
1             0.0        0               0             0              0
2             0.0        1               0             0              0
3             0.0        1               0             0              1
4             0.0        0               0             0              0

   type_DEBIT  type_PAYMENT  type_TRANSFER
0           0             1              0
1           0             1              0
2           0             0              1
3           0             0              0
4           0             1              0
```

```
Data types after Encoding:
step                int64
amount            float64
oldbalanceOrg     float64
newbalanceOrig    float64
oldbalanceDest    float64
newbalanceDest    float64
isFraud             int64
isFlaggedFraud      int64
type_CASH_IN        int64
type_CASH_OUT       int64
type_DEBIT          int64
type_PAYMENT        int64
type_TRANSFER       int64
dtype: object
```

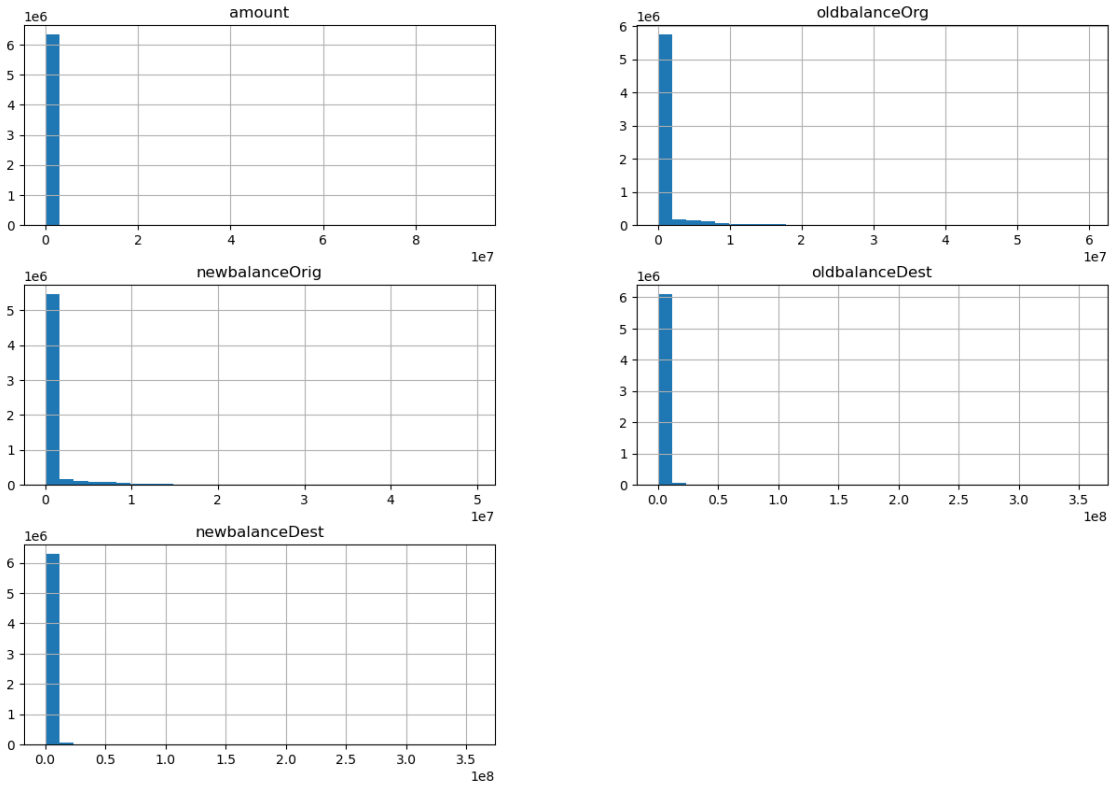
```
# Checking for missing values after encoding
print("Missing values in each column:")
print(data.isnull().sum())
```

```
Missing values in each column:
step                   0
amount                 0
oldbalanceOrg          0
newbalanceOrig    267230
oldbalanceDest    203603
newbalanceDest         0
isFraud                0
isFlaggedFraud         0
type_CASH_IN           0
type_CASH_OUT          0
type_DEBIT             0
type_PAYMENT           0
type_TRANSFER          0
dtype: int64
```

```
# Calculating skewness for numerical columns
import matplotlib.pyplot as plt

skewness = data[['amount', 'oldbalanceOrg', 'newbalanceOrig',
'oldbalanceDest', 'newbalanceDest']].skew()
print("Skewness of Numerical Columns:")
print(skewness)
print()
```

```
# Plotting histograms for numerical columns
data[['amount', 'oldbalanceOrg', 'newbalanceOrig', 'oldbalanceDest',
'newbalanceDest']].hist(bins=30, figsize=(15, 10))
plt.show()
```

```
Skewness of Numerical Columns:
amount            30.993949
oldbalanceOrg      5.249136
newbalanceOrig     5.176528
oldbalanceDest    20.093165
newbalanceDest    19.352302
dtype: float64
```




```
import numpy as np
from sklearn.impute import SimpleImputer

# Handling missing values using median imputation for skewed columns
imputer = SimpleImputer(strategy='median')
data[['newbalanceOrig', 'oldbalanceDest']] =
imputer.fit_transform(data[['newbalanceOrig', 'oldbalanceDest']])
```

```python
# Re-checking missing values after imputation
print("Missing values after imputation:")
print(data[numeric_columns].isnull().sum())
```

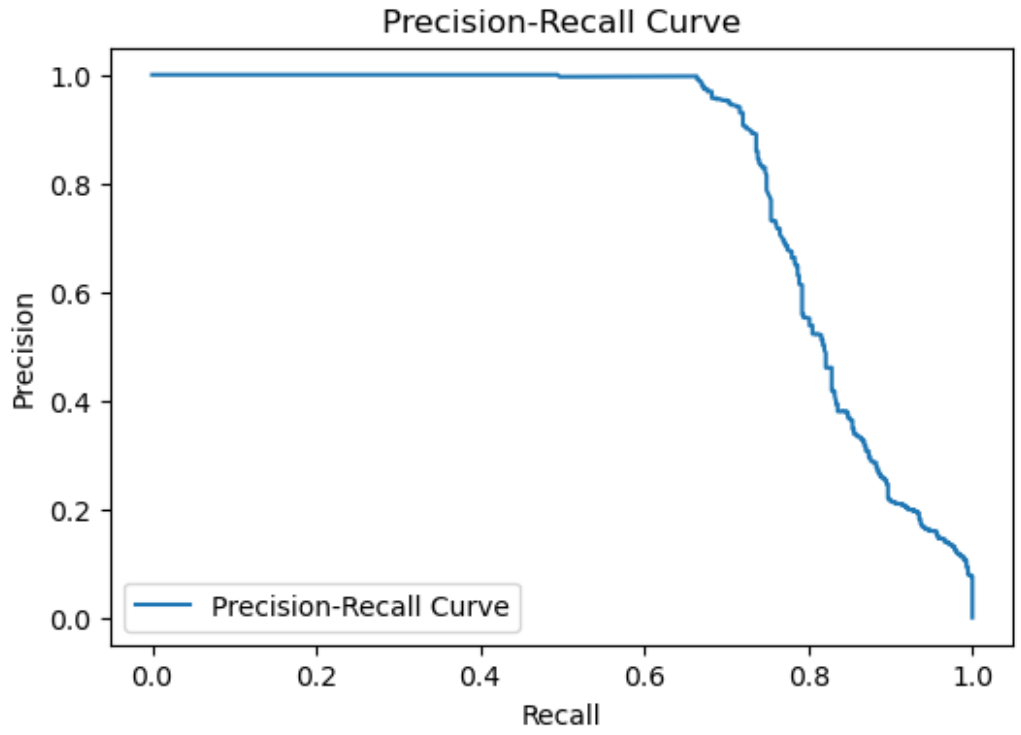
```
Missing values after imputation:
amount            0
oldbalanceOrg     0
newbalanceOrig    0
oldbalanceDest    0
newbalanceDest    0
dtype: int64
```

```python
# Model Train-Test

import xgboost as xgb
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score, f1_score, precision_recall_curve, auc
import matplotlib.pyplot as plt
import joblib

# Splitting data into train, validation, and live sets
train_data = data.iloc[:4000000]
validation_data = data.iloc[4000000:5000000]
live_data = data.iloc[5000000:]

# Defining feature sets and targets
X_train = train_data.drop(columns=['isFraud', 'isFlaggedFraud'])
y_train = train_data['isFraud']

X_validation = validation_data.drop(columns=['isFraud', 'isFlaggedFraud'])
y_validation = validation_data['isFraud']

print()
print("Model Training started using XGBoost...")
print()

# Training the XGBoost model
model = xgb.XGBClassifier(
    eval_metric='logloss',
    n_estimators=100,
    max_depth=6,
    learning_rate=0.1,
    subsample=0.8,
    colsample_bytree=0.8
)
model.fit(X_train, y_train)
```

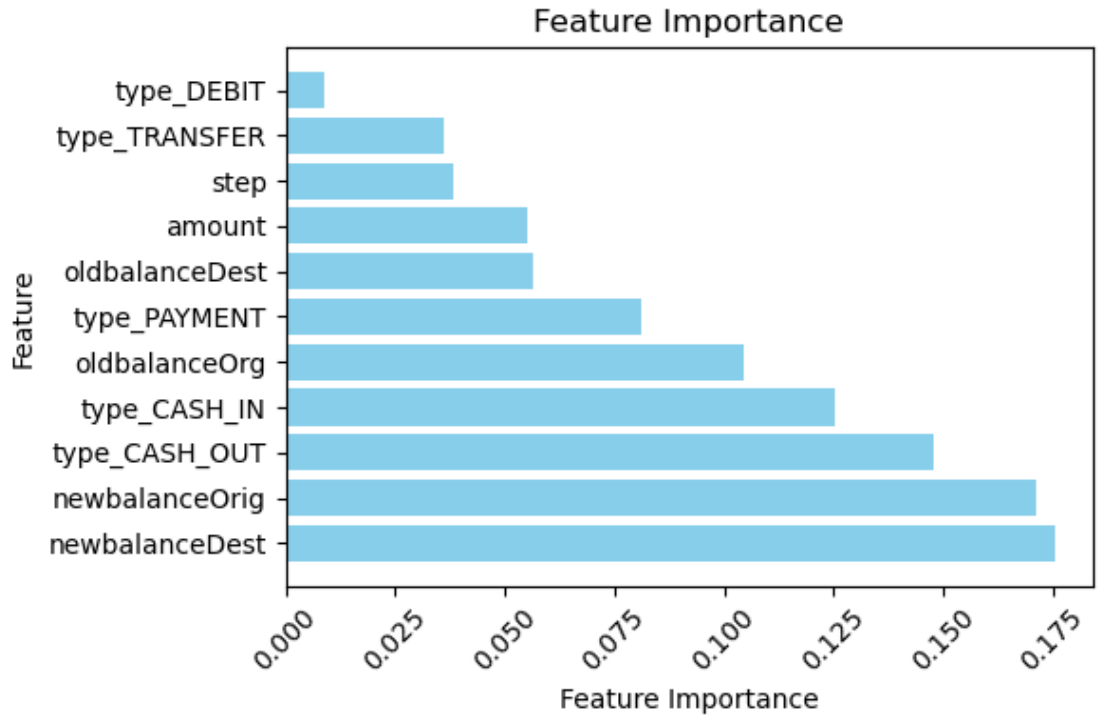
```
Model Training started using XGBoost...


XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=0.1, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=6,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None,
n_estimators=100,
              n_jobs=None, num_parallel_tree=None, random_state=None, ...)


# Model Prediction and evaluation

y_pred_eval = model.predict(X_validation)
y_pred_prob = model.predict_proba(X_validation)[:, 1]

print("Classification Report on Validation Dataset:")
print(classification_report(y_validation, y_pred_eval))

print()
print("ROC AUC Score (default threshold):", roc_auc_score(y_validation,
y_pred_prob))

print()
print("F1-score:", f1_score(y_validation, y_pred_eval))

print("\nConfusion Matrix:")
print(confusion_matrix(y_validation, y_pred_eval))

# Calculating precision-recall curve
precision, recall, thresholds = precision_recall_curve(y_validation,
y_pred_prob)
auc_pr = auc(recall, precision)
print("Precision-Recall AUC:", auc_pr)

# Plotting Precision-Recall Curve
plt.figure(figsize=(6, 4))
plt.plot(recall, precision, label='Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='best')
plt.show()
```

```
# Getting feature importances
feature_importances = model.feature_importances_

# Creating a DataFrame for better visualization
feature_importance_df = pd.DataFrame({'Feature': X_train.columns,
'Importance': feature_importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance',
ascending=False)

print("\nFeature Importance:")
print(feature_importance_df)

# Setting the figure size
plt.figure(figsize=(6, 4))

# Creatting a bar chart for feature importance
plt.barh(feature_importance_df['Feature'],
feature_importance_df['Importance'], color='skyblue')

# Setting labels and title
plt.xlabel('Feature Importance')
plt.ylabel('Feature')
plt.title('Feature Importance')

# Rotating x-axis labels for better readability
plt.xticks(rotation=45)

# Showing the plot
plt.tight_layout()
plt.show()

# Saving the trained model
joblib.dump(model, 'fraud_model.xgb')
print("Trained model is saved!!")
```

```
Classification Report on Validation Dataset:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    999446
           1       0.97      0.68      0.80       554

    accuracy                           1.00   1000000
   macro avg       0.99      0.84      0.90   1000000
weighted avg       1.00      1.00      1.00   1000000


ROC AUC Score (default threshold): 0.9995952170860057

F1-score: 0.7987220447284346

Confusion Matrix:
[[999436     10]
 [   179    375]]
Precision-Recall AUC: 0.8305181811092267
```



```
Feature Importance:
           Feature  Importance
5   newbalanceDest    0.175419
3   newbalanceOrig    0.171088
7    type_CASH_OUT    0.147731
6     type_CASH_IN    0.125227
2    oldbalanceOrg    0.104398
9     type_PAYMENT    0.081312
4   oldbalanceDest    0.056599
1           amount    0.055111
0             step    0.038399
10   type_TRANSFER    0.036067
8       type_DEBIT    0.008649
```




```
Trained model is saved!!
```

---

** Now running the saved scripts on jupyter notebook - train_model.py // predict_fraud.py // app.py ***

```
# Importing the training script
!python train_model.py

# Importing the Predict script
!python predict_fraud.py
```

```python
import subprocess

# Now running the Flask app using subprocess
subprocess.Popen(["python", "app.py"])
```

```
<Popen: returncode: None args: ['python', 'app.py']>
```

```python
import requests

# URL of the Flask API endpoint
url = 'http://localhost:5001/predict'

# Fraudulent transaction data
data = {
    "step": 1,
    "amount": 100000.0,
    "oldbalanceOrg": 50000.0,
    "newbalanceOrig": 20000.0,
    "oldbalanceDest": 10000.0,
    "newbalanceDest": 20000.0,
    "type_CASH_IN": 0,
    "type_CASH_OUT": 0,
    "type_DEBIT": 0,
    "type_PAYMENT": 0,
    "type_TRANSFER": 1
}

try:
    # Send POST request with data
    response = requests.post(url, json=data)

    # Check if the request was successful (status code 200)
    if response.status_code == 200:
        # Simply print the response JSON as it is
        print("Response from API:", response.json())
    else:
        print(f"Error: Received status code {response.status_code}")
except requests.exceptions.RequestException as e:
    print(f"An error occurred: {e}")
```

```
Response from API: {'predicted_isFraud': 0, 'predicted_prob':
6.078926890040748e-05}

127.0.0.1 - - [16/Dec/2024 14:42:31] "POST /predict HTTP/1.1" 200 -
```