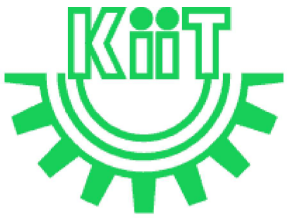
Mini Project Report
On
Fraud Detection using Python

Submitted By

Name: Harshit Raj
Roll No.:- 2105544
Name: Kishu Kumar
Roll No. : 2105549
Name: Aniket Talukdar
Roll No. : 22057011

School of Computer Engineering
KIIT - DU



KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY (KIIT)

Deemed to be University U/S 3 of UGC Act, 1956

Table of Contents

_

Introduction

Fraudulent activities pose a significant threat to financial institutions, causing billions of dollars in losses annually. Detecting and preventing fraud in bank payments is paramount to safeguarding the integrity of the financial system and protecting customers' assets. Traditional methods of fraud detection, such as rule-based systems, have limitations in capturing sophisticated fraudulent behaviors and often result in high false positive rates.

Machine learning (ML) techniques offer a promising solution to enhance fraud detection by leveraging the power of data analytics and pattern recognition. By analyzing historical transaction data, ML algorithms can identify subtle anomalies and patterns indicative of fraudulent activities. Python, with its rich ecosystem of libraries such as scikit-learn, imbalanced-learn, and XGBoost, provides a robust framework for implementing ML-based fraud detection systems.

In this project, we tackle the challenge of fraud detection in bank payments using Python and ML techniques. We leverage the Banksim dataset, a synthetically generated dataset containing payment transactions, to develop and evaluate our fraud detection models. Through exploratory data analysis (EDA), data preprocessing, and advanced ML algorithms such as K-Neighbours Classifier, Random Forest Classifier, and XGBoost Classifier, we aim to build accurate and efficient fraud detection models.

By combining rule-based systems with ML algorithms, we enhance the precision and reliability of fraud detection, minimizing false positives and false negatives. Through this approach, we empower financial institutions to mitigate the risks associated with fraudulent activities, safeguarding the interests of both businesses and customers.

INDIVIDUAL CONTRIBUTION-:

Harshit Raj (2105544)- XGBOOST- Classifier
Kishu Kumar (2105549)- KNN Classifier
Aniket Talukdar (22057011)- Random Forest Classifier

Problem Statement & Objectives

Issues Identified:

1. Imbalanced Datasets: The Banksim dataset may suffer from class imbalance, where the number of fraudulent transactions is significantly lower than the number of legitimate transactions. This imbalance can lead to biased model predictions, where the model may prioritize accuracy on the majority class at the expense of correctly identifying fraudulent transactions.
2. Complex Fraud Patterns: Fraudsters continually evolve their tactics to bypass detection systems, leading to increasingly sophisticated fraud patterns. Traditional rule-based systems may struggle to capture these complex patterns, resulting in high false positive rates or missing fraudulent activities altogether.
3. Limited Interpretability: Some machine learning algorithms, especially complex models like neural networks, lack interpretability, making it challenging to understand how they arrive at their predictions. This lack of transparency can hinder trust and adoption of the fraud detection system by stakeholders such as bank managers or regulatory authorities.

Selected Issue:

One selected issue for further explanation is the Imbalanced Dataset. Class imbalance occurs when the number of instances in one class (e.g., legitimate transactions) is much higher than the other class (e.g., fraudulent transactions). In the context of fraud detection, this imbalance means that the model may be biased towards predicting the majority class (i.e., non-fraudulent transactions) while neglecting the minority class (i.e., fraudulent transactions).
This issue is particularly crucial because failing to detect fraudulent transactions can lead to substantial financial losses for both the bank and its customers. Moreover, false negatives (i.e., legitimate transactions incorrectly classified as fraudulent) can inconvenience customers and damage the bank's reputation.Therefore, addressing class imbalance is essential for building an effective fraud detection system.

Objectives:

1. Improve Model Performance: The primary objective is to enhance the performance of the fraud detection model by addressing class imbalance. This involves implementing techniques such as oversampling, undersampling, or generating synthetic samples to balance the distribution of the minority class and improve the model's ability to detect fraudulent transactions.
2. Reduce False Negatives: Another objective is to minimize false negatives, where fraudulent transactions are incorrectly classified as legitimate. By reducing false negatives, we aim to increase the sensitivity or recall of the model, ensuring that fraudulent activities are accurately detected and mitigated in a timely manner.
3. Maintain Precision: While improving sensitivity is crucial, it's also essential to maintain precision to avoid excessive false positives. Therefore, the objective is to balance sensitivity and precision by selecting appropriate evaluation metrics and fine-tuning the model parameters to achieve optimal performance in detecting fraudulent transactions while minimizing false alarms.

Python Packages Used Details

Name	Function Used	Explanation
Pandas pd()	DataFrame and read_csv	Pandas is a powerful data manipulation library in Python. It provides a DataFrame data structure for working with structured data, and read_csv function for reading data from CSV files into DataFrames.
NumPy (np)	numpy.array, numpy.zeros, and numpy.ones functions	NumPy is a fundamental package for scientific computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.
imbalanced-learn	SMOTE	imbalanced-learn is a Python library specifically designed for handling imbalanced datasets in machine learning. SMOTE (Synthetic Minority Over-sampling Technique) is one of its functions used for oversampling the minority class to balance the class distribution.
seaborn	set, sns.set(), sns.countplot(), sns.heatmap()	Seaborn is a data visualization library based on Matplotlib. It provides a high-level interface for creating attractive and informative statistical graphics. Functions like set are

		used for setting aesthetics, countplot for plotting count distributions, and heatmap for visualizing correlations.
matplotlib.pyplot as plt	plt.plot(), plt.xlabel(), plt.ylabel(), plt.title(), plt.legend(), plt.show()	Matplotlib is a plotting library for Python. The pyplot module provides a MATLAB-like interface for creating plots and visualizations. Functions like plot are used for creating line plots, xlabel and ylabel for labeling axes, title for setting the title of the plot, legend for adding a legend, and show for displaying the plot.
sklearn.model_selection	train_test_split	This module provides functions for splitting datasets into train and test sets for model evaluation. train_test_split is commonly used to split data into training and testing sets.
sklearn.metrics	confusion_matrix, classification_report, roc_curve, auc	This module provides various evaluation metrics for assessing the performance of machine learning models. Functions like confusion_matrix, classification_report are used for evaluating classification models, while roc_curve and auc are used for evaluating binary classifiers using ROC curve analysis.
xgboost as xgb	None mentioned explicitly, but commonly used functions include xgb.XGBClassifier,	XGBoost is an efficient and scalable implementation of gradient boosting

	xgb.fit, and xgb.predict_proba	algorithms. It provides an implementation of the gradient boosting framework for classification and regression tasks. Functions like XGBClassifier are used to create XGBoost classifier models.
sklearn.neighbors	KNeighborsClassifier	This module provides implementations of various algorithms for nearest neighbor-based classification. KNeighborsClassifier is used for building K-Nearest Neighbors classifier models.
sklearn.ensemble	RandomForestClassifier, VotingClassifier	This module provides ensemble learning methods for building multiple models and combining their predictions. RandomForestClassifier is used to create random forest classifier models, while VotingClassifier is used to combine predictions from multiple classifiers using voting strategies.

Source Code:

1.Reading Datasets

```
import pandas as pd
data = pd.read_csv("miniproject.csv")
data.head(5)
```

2. Information Of The Data

```
data.info()
```

3. Create two dataframes with fraud and non-fraud data.

```
import seaborn as sns
import matplotlib.pyplot as plt
df_fraud = data.loc[data.fraud == 1]
df_non_fraud = data.loc[data.fraud == 0]
sns.countplot(x="fraud",data=data)
plt.title("Count of Fraudulent Payments")
plt.show()
print("Number of normal examples: ",df_non_fraud.fraud.count())
print("Number of fradulent examples: ",df_fraud.fraud.count())
```

4.Mean feature values per category

```
print("Mean feature values per category", data.groupby('category')[['amount', 'fraud']].mean())
```

5.GROUPBY

```
pd.concat([df_fraud.groupby('category')['amount'].mean(),df_non_fraud.groupby('category')['am
ount'].mean(),\data.groupby('category')['fraud'].mean()*100],keys=["Fraudulent","Non-
Fraudulent","Percent(%)"],axis=1,\sort=False).sort_values(by=['Non-Fraudulent'])
```

6.Plot boxplot of the amounts in fraud and non-fraud data

```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(30,10))
sns.boxplot(x=data.category,y=data.amount)
plt.title("Boxplot for the Amount spend in category")
plt.ylim(0,4000)
plt.legend()
plt.show()
```

7. Plot histograms of the amounts in fraud and non-fraud data

```
plt.hist(df_fraud.amount, alpha=0.5, label='fraud',bins=100)
plt.hist(df_non_fraud.amount, alpha=0.5, label='nonfraud',bins=100)
plt.title("Histogram for fraudulent and nonfraudulent payments")
plt.ylim(0,10000)
plt.xlim(0,1000)
plt.legend()
plt.show()
```

8. Group by age

```
print((data.groupby('age')['fraud'].mean()*100).reset_index().rename(columns={'age':'Age','frau
d' : 'Fraud Percent'}).sort_values(by='Fraud Percent'))
```

9. Dropping zipcodeori and zipMerchant since they have only one unique value

```
print("Unique zipCodeOri values: ",data.zipcodeOri.nunique())
print("Unique zipMerchant values: ",data.zipMerchant.nunique())
data_reduced = data.drop(['zipcodeOri','zipMerchant'],axis=1)
```

10.Checking the data after dropping.

```
data_reduced.columns
```

11.Turning object columns type to categorical for easing the transformation

```
col_categorical = data_reduced.select_dtypes(include= ['object']).columns
for col in col_categorical:
data_reduced[col] = data_reduced[col].astype('category')
data_reduced[col_categorical] = data_reduced[col_categorical].apply(lambda x: x.cat.codes)
data_reduced.head(5)
```

12. Define our independent variable (X) and dependant/target variable y

```
X = data_reduced.drop(['fraud'],axis=1)
y = data['fraud']
print(X.head(),"\n")
print(y.head())
```

13.Oversampling with SMOTE

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)
y_res = pd.DataFrame(y_res)
print(y_res.value_counts())
```

14.Train Test Split for measuring the performance

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =
train_test_split(X_res,y_res,test_size=0.3,random_state=42,shuffle=True,stratify=y_res)
```

15.Function for plotting ROC_AUC curve

```
def plot_roc_auc(y_test, preds):
fpr, tpr, threshold = roc_curve(y_test, preds)
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

16.Accuracy score

```
print("Base accuracy score we must beat is: ",df_non_fraud.fraud.count()/
np.add(df_non_fraud.fraud.count(),df_fraud.fraud.count()) * 100)
```

17.KNN Neigbors

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report
knn = KNeighborsClassifier(n_neighbors=5,p=1)
knn.fit(X_train,y_train)
y_pred = knn.predict(X_test)
print("Classification Report for K-Nearest Neighbours: \n", classification_report(y_test, y_pred))
print("Confusion Matrix of K-Nearest Neigbours: \n", confusion_matrix(y_test,y_pred))
plot_roc_auc(y_test, knn.predict_proba(X_test)[:,1])
```

18.Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report
rf_clf =
RandomForestClassifier(n_estimators=100,max_depth=8,random_state=42,verbose=1,class_we
ight="balanced")
rf_clf.fit(X_train,y_train)
y_pred = rf_clf.predict(X_test)
```

```
print("Classification Report for Random Forest Classifier: \n", classification_report(y_test, y_pred))
print("Confusion Matrix of Random Forest Classifier: \n", confusion_matrix(y_test,y_pred))
plot_roc_auc(y_test, rf_clf.predict_proba(X_test)[:,1])
```

19.XGBOOST- Classifier

```
import xgboost as xgb
XGBoost_CLF = xgb.XGBClassifier(max_depth=6, learning_rate=0.05, n_estimators=400, objective="binary:hinge", booster='gbtree', n_jobs=-1, nthread=None, gamma=0, min_child_weight=1, max_delta_step=0, subsample=1, colsample_bytree=1, colsample_bylevel=1, reg_alpha=0, reg_lambda=1, scale_pos_weight=1, base_score=0.5, random_state=42, verbosity=1)

XGBoost_CLF.fit(X_train,y_train)
y_pred = XGBoost_CLF.predict(X_test)
print("Classification Report for XGBoost: \n", classification_report(y_test, y_pred))
print("Confusion Matrix of XGBoost: \n", confusion_matrix(y_test,y_pred))
plot_roc_auc(y_test, XGBoost_CLF.predict_proba(X_test)[:,1])
```

IMPLEMENTATION RESULT

The obtained results in the form of graphs, charts, etc., are to be provided here. The clear explanation of why the obtained results are so, are to be mentioned here.

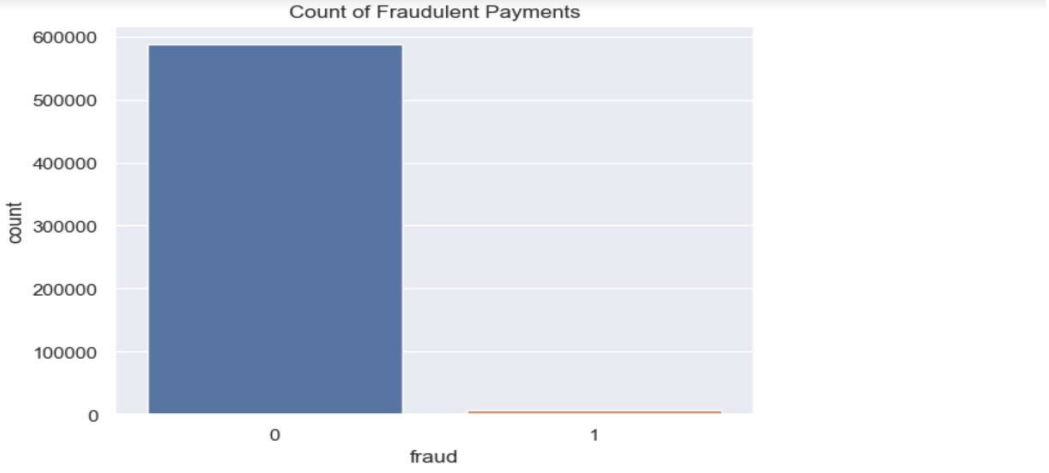
1.Reading Datasets

	step	customer	age	gender	zipcodeOri	merchant	zipMerchant	category	amount	fraud
0	0	'C1093826151'	'4'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	4.55	0
1	0	'C352968107'	'2'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	39.68	0
2	0	'C2054744914'	'4'	'F'	'28007'	'M1823072687'	'28007'	'es_transportation'	26.89	0
3	0	'C1760612790'	'3'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	17.25	0
4	0	'C757503768'	'5'	'M'	'28007'	'M348934600'	'28007'	'es_transportation'	35.72	0

2. Information Of The Data

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 594643 entries, 0 to 594642
Data columns (total 10 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   step         594643 non-null  int64
 1   customer     594643 non-null  object
 2   age          594643 non-null  object
 3   gender       594643 non-null  object
 4   zipcodeOri   594643 non-null  object
 5   merchant     594643 non-null  object
 6   zipMerchant  594643 non-null  object
 7   category     594643 non-null  object
 8   amount       594643 non-null  float64
 9   fraud        594643 non-null  int64
dtypes: float64(1), int64(2), object(7)
memory usage: 45.4+ MB
```

3. Create two dataframes with fraud and non-fraud data.



```
Number of normal examples:  587443
Number of fradulent examples:  7200
```

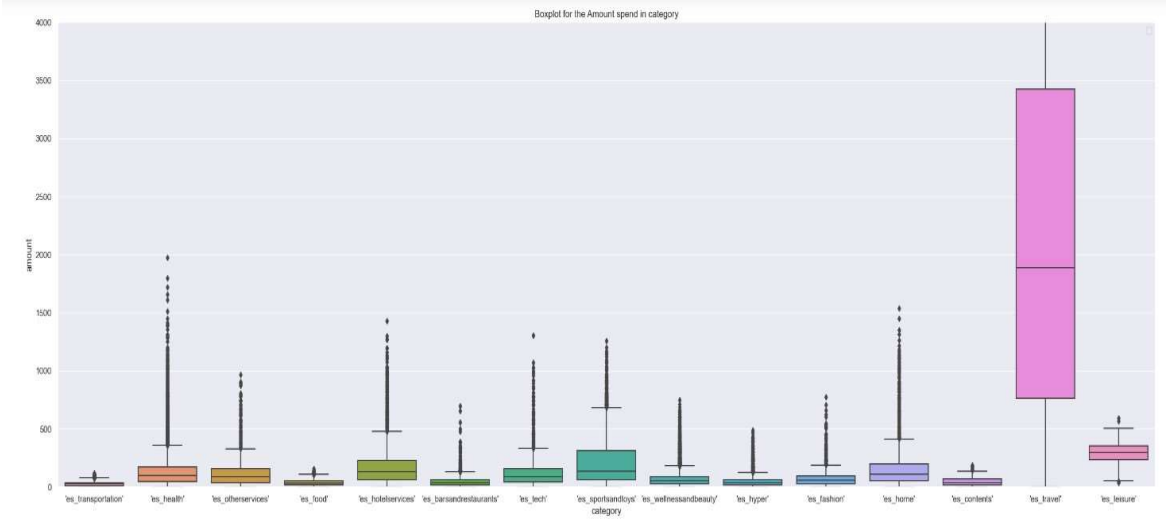
4.Mean feature values per category

```
Mean feature values per category                              amount     fraud
category
'es_barsandrestaurants'    43.461014  0.018829
'es_contents'              44.547571  0.000000
'es_fashion'               65.666642  0.017973
'es_food'                  37.070405  0.000000
'es_health'               135.621367  0.105126
'es_home'                 165.670846  0.152064
'es_hotelservices'        205.614249  0.314220
'es_hyper'                 45.970421  0.045917
'es_leisure'              288.911303  0.949900
'es_otherservices'        135.881524  0.250000
'es_sportsandtoys'        215.715280  0.495252
'es_tech'                 120.947937  0.066667
'es_transportation'        26.958187  0.000000
'es_travel'              2250.409190  0.793956
'es_wellnessandbeauty'     65.511221  0.047594
```

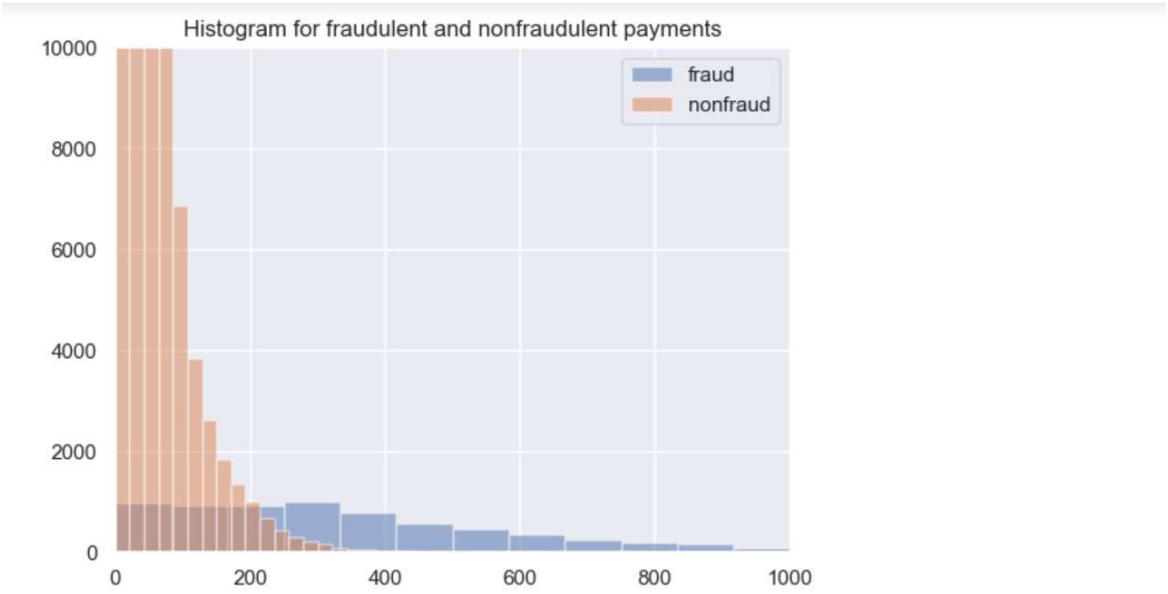
5.Group By Category

category	Fraudulent	Non-Fraudulent	Percent(%)
'es_transportation'	NaN	26.958187	0.000000
'es_food'	NaN	37.070405	0.000000
'es_hyper'	169.255429	40.037145	4.591669
'es_barsandrestaurants'	164.092667	41.145997	1.882944
'es_contents'	NaN	44.547571	0.000000
'es_wellnessandbeauty'	229.422535	57.320219	4.759380
'es_fashion'	247.008190	62.347674	1.797335
'es_leisure'	300.286878	73.230400	94.989980
'es_otherservices'	316.469605	75.685497	25.000000
'es_sportsandtoys'	345.366811	88.502738	49.525237
'es_tech'	415.274114	99.924638	6.666667
'es_health'	407.031338	103.737228	10.512614
'es_hotelservices'	421.823339	106.548545	31.422018
'es_home'	457.484834	113.338409	15.206445
'es_travel'	2660.802872	669.025533	79.395604

6.Plot boxplot of the amounts in fraud and non-fraud data



7. Plot histograms of the amounts in fraud and non-fraud data



8.Group by age

	Age	Fraud Percent
7	'U'	0.594228
6	'6'	0.974826
5	'5'	1.095112
1	'1'	1.185254
3	'3'	1.192815
2	'2'	1.251401
4	'4'	1.293281
0	'0'	1.957586

9.Dropping zipcodeori and zipMerchant since they have only one unique value.

```
Unique zipCodeOri values:  1
Unique zipMerchant values:  1
```

10.Checking the data after dropping.

```
Index(['step', 'customer', 'age', 'gender', 'merchant', 'category', 'amount',
       'fraud'],
      dtype='object')
```

11.Turning object columns type to categorical for easing the transformation

	step	customer	age	gender	merchant	category	amount	fraud
0	0	210	4	2	30	12	4.55	0
1	0	2753	2	2	30	12	39.68	0
2	0	2285	4	1	18	12	26.89	0
3	0	1650	3	2	30	12	17.25	0
4	0	3585	5	2	30	12	35.72	0

12.Define our independent variable (X) and dependant/target variable y

```
   step  customer  age  gender  merchant  category  amount
0     0       210    4       2        30        12    4.55
1     0      2753    2       2        30        12   39.68
2     0      2285    4       1        18        12   26.89
3     0      1650    3       2        30        12   17.25
4     0      3585    5       2        30        12   35.72

0    0
1    0
2    0
3    0
4    0
Name: fraud, dtype: int64
```

13.Counting Fraud

```
fraud
0       587443
1       587443
Name: count, dtype: int64
```

14. Accuracy score.

```
Base accuracy score we must beat is:  98.7891894800746
```

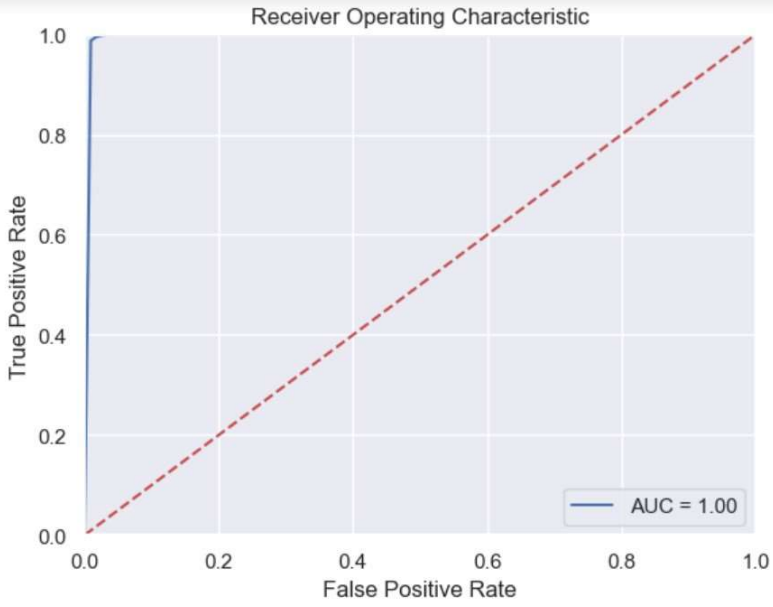
15.KNN Neigbors

```
Classification Report for K-Nearest Neighbours:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99    176233
           1       0.98      1.00      0.99    176233

    accuracy                           0.99    352466
   macro avg       0.99      0.99      0.99    352466
weighted avg       0.99      0.99      0.99    352466

Confusion Matrix of K-Nearest Neigbours:
 [[171999   4234]
 [   362 175871]]
```



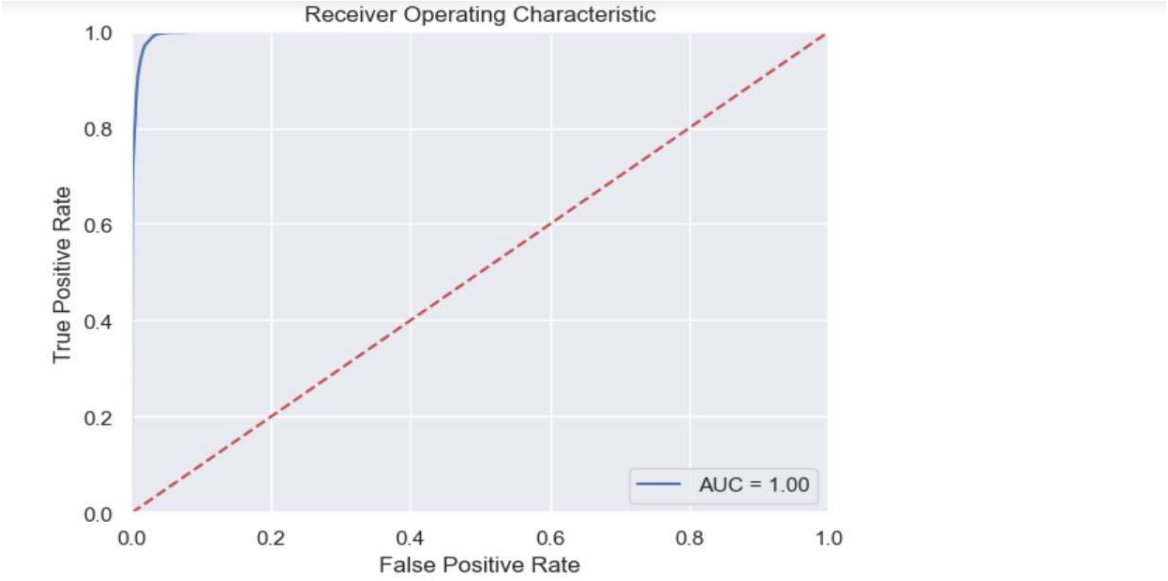
16.Random Forest Classifier

```
Classification Report for Random Forest Classifier:
              precision    recall  f1-score   support

           0       0.99      0.97      0.98    176233
           1       0.97      0.99      0.98    176233

    accuracy                           0.98    352466
   macro avg       0.98      0.98      0.98    352466
weighted avg       0.98      0.98      0.98    352466

Confusion Matrix of Random Forest Classifier:
 [[170106   6127]
 [  1079 175154]]
```



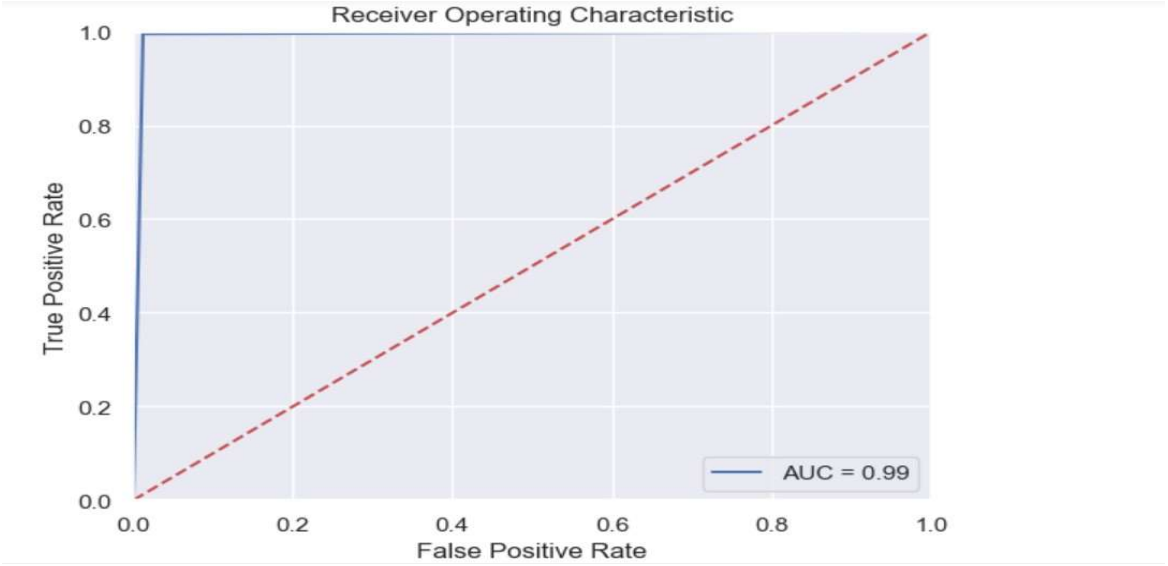
17.XGBoost Classifier

```
Classification Report for XGBoost:
               precision    recall  f1-score   support

           0       1.00      0.99      0.99    176233
           1       0.99      1.00      0.99    176233

    accuracy                           0.99    352466
   macro avg       0.99      0.99      0.99    352466
weighted avg       0.99      0.99      0.99    352466

Confusion Matrix of XGBoost:
 [[174100   2133]
 [   787 175446]]
```



CONCLUSION

In this project, we addressed the challenge of fraud detection in bank payments using machine learning techniques. We leveraged the Banksim dataset, a synthetically generated dataset containing payment transactions, to develop and evaluate our fraud detection models. Through a combination of exploratory data analysis (EDA), data preprocessing, and advanced machine learning algorithms, we aimed to build accurate and efficient fraud detection systems.

Findings:

1. Data Imbalance: We identified that the dataset suffers from class imbalance, where the number of fraudulent transactions is significantly lower than legitimate transactions. This imbalance poses a challenge for building effective fraud detection models as they may prioritize accuracy on the majority class, leading to the underrepresentation of fraudulent activities.

2. Complex Fraud Patterns: We observed that fraud patterns in bank payments can be complex and evolve over time. Traditional rule-based systems may struggle to capture these intricate patterns, resulting in high false positive rates or missing fraudulent activities altogether. Machine learning algorithms offer a promising solution to address this issue by analyzing historical transaction data and identifying subtle anomalies indicative of fraudulent behavior.

3. Model Performance: Through experimentation with various machine learning algorithms including K-Nearest Neighbors, Random Forest, and XGBoost classifiers, we found that ensemble methods like XGBoost generally outperformed individual classifiers in terms of fraud detection accuracy and robustness.

Methods Used:

1. Exploratory Data Analysis (EDA): We performed EDA to gain insights into the distribution of features, identify correlations, and visualize patterns in the data. EDA helped us understand the characteristics of fraudulent transactions and guide our preprocessing steps.

2. Data Preprocessing: We cleaned the data, handled missing values, encoded categorical variables, and scaled numerical features to prepare the dataset for model training. Additionally, we addressed class imbalance by oversampling the minority class using SMOTE (Synthetic Minority Over-sampling Technique).

3. Model Training and Evaluation: We trained several machine learning models on the preprocessed data and evaluated their performance using metrics such as confusion matrix, classification report, and ROC curve analysis. We fine-tuned model parameters and selected the best-performing model based on its ability to detect fraudulent transactions while minimizing false positives and false negatives.

Future Directions:

Moving forward, there are several avenues for further improvement and exploration:

1. **Feature Engineering**: Experiment with additional feature engineering techniques to capture more nuanced patterns in the data and improve model performance.

2. **Ensemble Learning:** Explore more sophisticated ensemble methods and model stacking techniques to further enhance the robustness and generalization capabilities of the fraud detection system.

3. Real-time Monitoring: Implement the fraud detection system in a real-time environment, allowing for continuous monitoring and adaptation to emerging fraud patterns.

In conclusion, our project demonstrates the effectiveness of machine learning in detecting fraudulent activities in bank payments. By combining advanced analytics with domain knowledge, we can build more resilient and accurate fraud detection systems, ultimately safeguarding the financial integrity of institutions and protecting customers' assets.

References

Kaggle