FRAUD DETECTION IN HEALTHCARE FOR MITIGATING MEDICARE SPENDING

Amila Viraj Mahinda Hewa Lunuwilage, Lohith Prasanna Teja Kakumanu, Monika Jangam Prabhudev

Background of the Problem:

The issue of provider fraud in Medicare is a serious problem because it has a significant impact on healthcare spending, increases insurance costs, and can affect the accessibility of quality healthcare services for people. The goal of the project is to tackle this problem by developing a system that can predict which healthcare providers might be engaging in fraudulent activities. Additionally, the project aims to understand the key factors that contribute to fraudulent behavior among providers and analyze patterns to better anticipate and prevent future fraudulent activities.Ultimately, the objective is to create a more secure and cost-effective healthcare system.

Problem Statement and significance:

This project's objective is to "predict the potentially fraudulent providers" based on the claims that these providers have submitted. In addition, we shall find significant factors that are useful in identifying the actions of possible fraud suppliers. To comprehend the future actions of providers, we will also examine patterns of deception in their claims.

The significance of this issue has serious consequences in two main ways. First, it makes healthcare more expensive for everyone by putting financial strain on the government and
insurance companies. Second, it damages the trust in our healthcare system because resources are being misused for fraudulent activities instead of helping patients. To ensure that Medicare works fairly and effectively, we need to tackle and reduce fraud among healthcare providers.

Potential of the problem:

The potential Fraud in Medicare related is critical as it impacts healthcare spending, increases insurance costs, and can affect the accessibility of quality healthcare services. In order to solve this problem, we came up with a solution to understand the data from multiple insurance providers and healthcare providers where a particular beneficiary has undergone through a particular treatment including how many days he was admitted what all diseases were diagnosed and lot more parameters to build a predictive model to identify whether the transaction made by the beneficiary in this sector is Fraudulent or not.

Objective:

The primary objective of this study is to develop an advanced predictive model for identifying and mitigating provider fraud within the Medicare system. Additionally, we aim to pinpoint key patterns and indicators in Medicare claims data to enhance fraud detection capabilities and contribute to the overall integrity and sustainability of healthcare insurance systems.

Data:

We have picked the data from Kaggle.
Source:https://www.kaggle.com/datasets/rohitrox/healthcare-provider-fraud-detection-analysis/data [1]
We're focusing on three main types of healthcare data for this project:

A) **Inpatient Data:**

This data gives us insights into the claims made for patients who are admitted to hospitals.
It includes additional details such as when they were admitted and discharged, as well as the diagnosis code for their admission.

B) **Outpatient Data:**

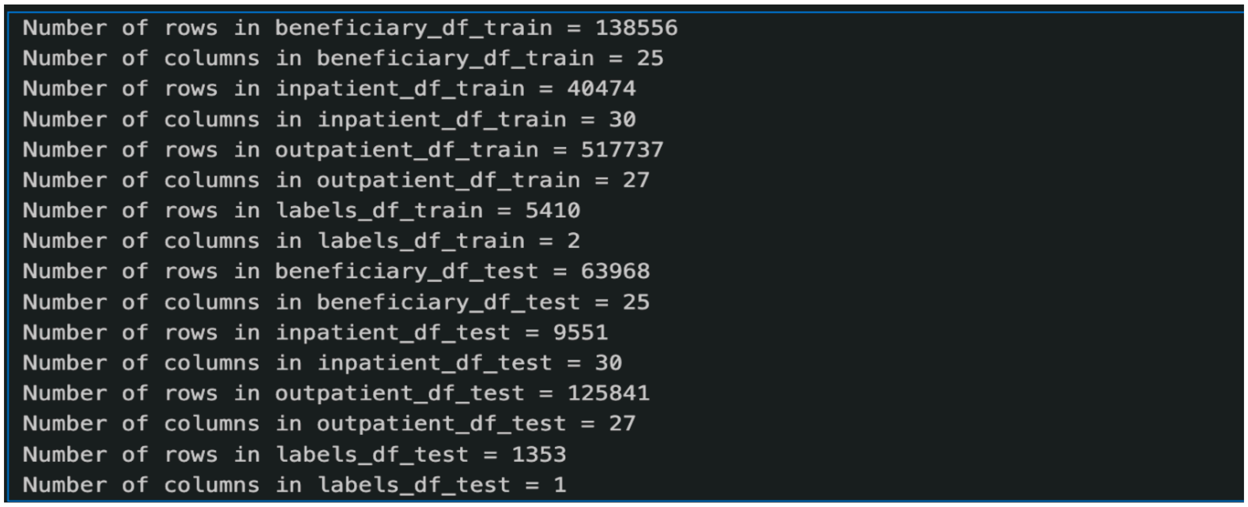
This data provides information about the claims made for patients who visit hospitals but are not admitted.

C) **Beneficiary Details Data:**

This data contains personal details about the beneficiaries, like their health conditions and the region they belong to.

Essentially, we're looking at the information related to patients admitted to hospitals, those who visit without admission, and the background details of the beneficiaries.

This dataset comprises of:

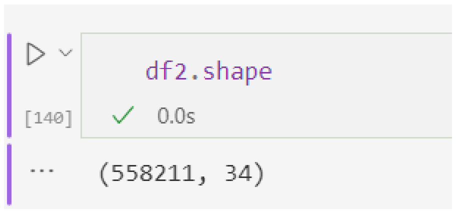


```
Number of rows in beneficiary_df_train = 138556
Number of columns in beneficiary_df_train = 25
Number of rows in inpatient_df_train = 40474
Number of columns in inpatient_df_train = 30
Number of rows in outpatient_df_train = 517737
Number of columns in outpatient_df_train = 27
Number of rows in labels_df_train = 5410
Number of columns in labels_df_train = 2
Number of rows in beneficiary_df_test = 63968
Number of columns in beneficiary_df_test = 25
Number of rows in inpatient_df_test = 9551
Number of columns in inpatient_df_test = 30
Number of rows in outpatient_df_test = 125841
Number of columns in outpatient_df_test = 27
Number of rows in labels_df_test = 1353
Number of columns in labels_df_test = 1
```

We performed various standards on our dataset to clean the entire information and make it
consistent, accurate ensuring the developers and stakeholders to have confidence in the insights and the decisions made by the model which include all the operations like handling missing values, dropping out the duplicate entries, formatting the data using different standardization techniques, dropping unwanted features or columns, removing the outliers, adding relevant features with Normalization and

Data Validation. After diving deep into the beneficiary or the patient data we segregated and merged, we tried to analyze them over different types including visual exploration to understand the patterns in the data, detect certain data points which don't follow the pattern significantly deviating from the rest of the data points available in the dataset.

After performing these all operations now our reliable **Data Frame** comprises of:



```
df2.shape
[140]  ✓ 0.0s
... (558211, 34)
```

With these steps completed, the data stands poised and refined, fully prepared for the modeling phase. When considering the classification tasks, we evaluate our Model using some of the Metrics like Precision, Recall, F1Score, Support.

Greater accuracy in classifying fraudulent transactions **while reducing false alarms**—transactions that are legitimate but are mistakenly categorized as fraudulent—is indicated by a higher AUC. As a result, it is preferable to maximize the AUC since it represents the classifier's overall effectiveness in differentiating between real and fraudulent transactions across various decision thresholds.

This is the Whole Purpose of Modelling process. Through careful evaluation and tuning of various machine learning algorithms, we strive to identify the model that best meets the requirements of our task. The **goal** is to achieve **accurate and reliable predictions**, particularly in critical applications such as **fraud detection**.

To Achieve this, we start our Modeling process:

Models:

We start our modeling process with Naïve Bayes and continue till the XGBoost

Model 1 : Naive Bayes

1.1 Justification for choosing:

The Naïve Bayes model was chosen for this project due to its suitability for analyzing the extensive and categorical healthcare claims data used for fraud detection. Given the complexity of healthcare transactions and the need for efficient processing, Naïve Bayes' simplicity and scalability make it an ideal candidate. Despite its assumption of feature independence, the model's performance can be effective, especially in identifying potentially fraudulent providers where factors may act independently. Its interpretability is crucial for explaining the reasons behind fraud predictions to healthcare stakeholders. Additionally, Naïve Bayes serves as a robust baseline model for evaluating the effectiveness of more complex algorithms in our fraud detection system.

1.2 Tuning the model:

The initial model achieves an accuracy of 62.5%, with a precision of 52% for fraudulent transactions, indicating misclassification of legitimate transactions. Additionally, the model's recall for fraudulent cases is only 15%, emphasizing the need for tuning strategies to improve its ability to detect fraudulent activity while minimizing false positives and false negatives.

1.3 Effectiveness:

The classification report provides more detailed metrics:

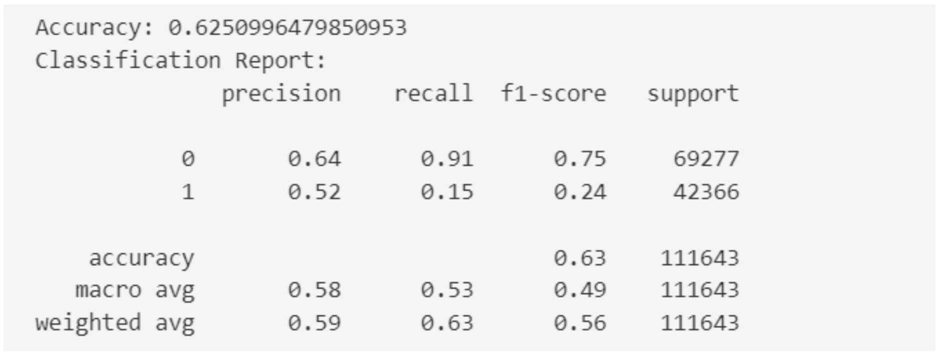
Accuracy of 0.625 which means 63% percentage of transactions the model classified correctly (both fraudulent and non-fraudulent) which is just above the random guessing baseline (50% for two classes).

Precision: Precision for class 0 (non-fraudulent) is 64%. This means that out of all transactions classified as non-fraudulent by the model, 64% were non-fraudulent. Conversely, precision for class 1 (fraudulent) is lower at 52%. This indicates that the model might be misclassifying some non-fraudulent transactions as fraudulent (false positives)

Recall for class 0 (non-fraudulent) is 91%. This means that the model identified 91% of the actual non-fraudulent transactions correctly. However, the recall for class 1 (fraudulent) is very low at 15%. This signifies that the model is missing a significant portion of fraudulent cases (false negatives).

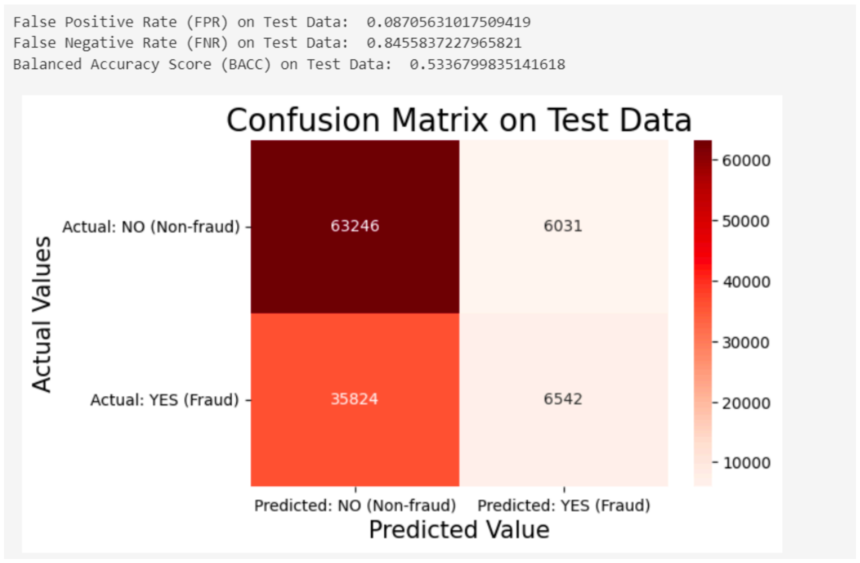
The **F1**-score for class 0 is decent (0.75), but the F1-score for class 1 (fraudulent) is very low (0.24), highlighting the model's struggle with identifying fraudulent cases.

Below is the Classification Report or evaluation report for the same:



```
Accuracy: 0.6250996479850953
Classification Report:
              precision    recall  f1-score   support

           0       0.64      0.91      0.75     69277
           1       0.52      0.15      0.24     42366

    accuracy                           0.63    111643
   macro avg       0.58      0.53      0.49    111643
weighted avg       0.59      0.63      0.56    111643
```

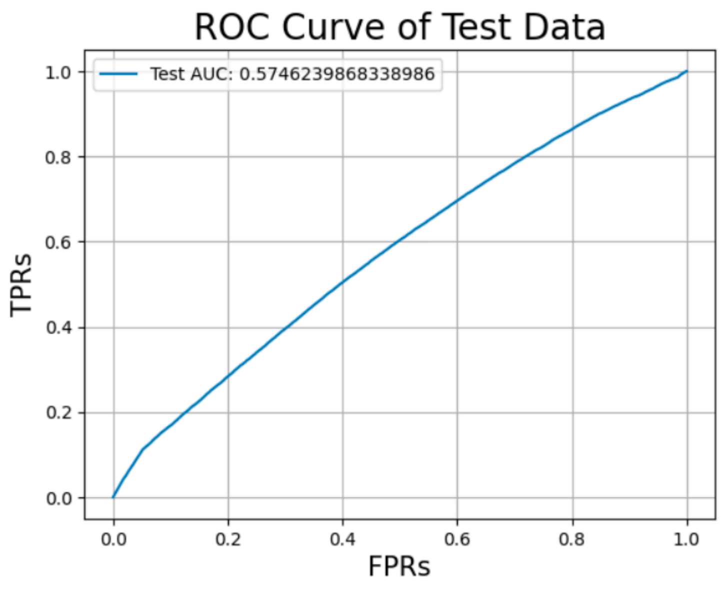
Confusion Matrix:



```
False Positive Rate (FPR) on Test Data:  0.08705631017509419
False Negative Rate (FNR) on Test Data:  0.8455837227965821
Balanced Accuracy Score (BACC) on Test Data:  0.5336799835141618
```

The above **Correlation matrix** tells us that overall, the model seems to be performing well at classifying non-fraudulent transactions (63,246 correctly classified) with a low number of false positives (6,031). However, there is a higher number of false negatives (35,824) which could be an area for improvement.

ROC:



ROC Curve tells us that this model has an **AUC** of 0.5746. An AUC of 1 represents a perfect classifier while an AUC of 0.5 represents a random classifier. So, this model's performance is slightly better than random chance.

1.4 Insights

Naïve Bayes demonstrates reasonable performance in classifying healthcare transactions for fraud detection. However, its struggle to accurately identify fraudulent cases, as evidenced by low recall and precision scores, suggests the need for careful tuning and optimization. While Naïve Bayes provides a solid foundation, further refinement through advanced tuning strategies and feature engineering could significantly enhance its effectiveness in detecting fraudulent activities within healthcare claims data.

Model 2 : Logistic Regression

2.1 Justification for choosing:

It is a fundamental Classification algorithm which is used when we need to Classify the given data into Yes/No or 1/0 or Fraudulent/Not in our case. And this is a good baseline for Fraud Detection tasks. Logistic Regression is chosen for its simplicity and interpretability, making it easy to understand the factors contributing to fraudulent behavior among healthcare providers.

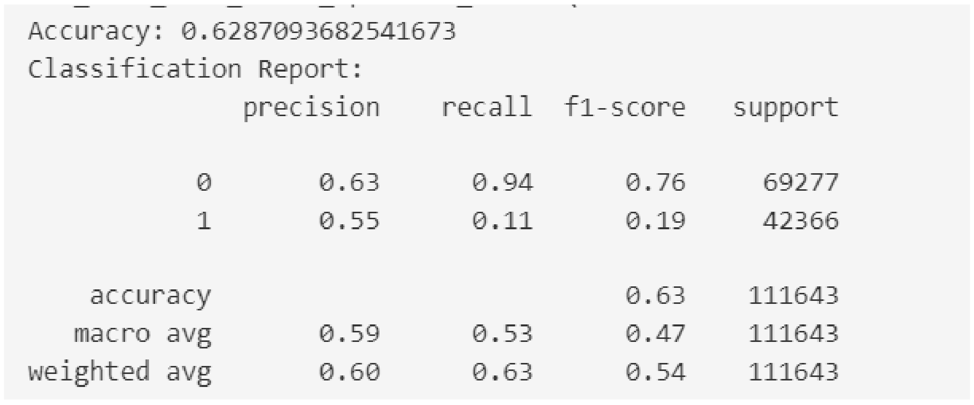
2.2 Tuning the model:

The Logistic Regression model achieves an accuracy of approximately 62.87%, with a precision of 55% for fraudulent transactions and 63% for non-fraudulent transactions. However, its low recall of 11% for fraudulent transactions highlights the need for tuning to enhance its ability to identify fraudulent cases while minimizing false negatives.

2.3 Effectiveness:

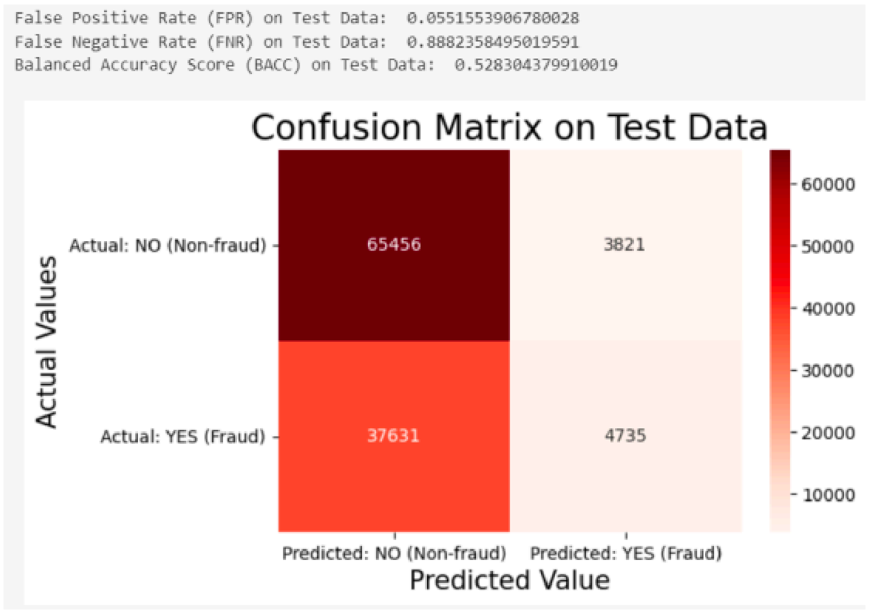
The classification report provides more detailed metrics:
Accuracy: Logistic Regression achieves approximately 62.87% accuracy on the testing data.
Precision: 55% precision for fraudulent transactions and 63% for non-fraudulent transactions.
Recall: Low recall of 11% for fraudulent transactions and high recall of 94% for non-fraudulent transactions.
F1 Score: 0.18 for fraudulent transactions and 0.75 for non-fraudulent transactions.

Below is the Evaluation report:



```
Accuracy: 0.6287093682541673
Classification Report:
              precision    recall  f1-score   support

           0       0.63      0.94      0.76     69277
           1       0.55      0.11      0.19     42366

    accuracy                           0.63    111643
   macro avg       0.59      0.53      0.47    111643
weighted avg       0.60      0.63      0.54    111643
```

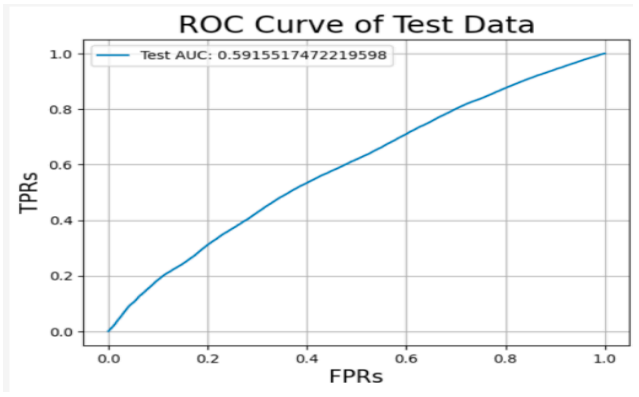
Confusion Matrix:



```
False Positive Rate (FPR) on Test Data:  0.0551553906780028
False Negative Rate (FNR) on Test Data:  0.8882358495019591
Balanced Accuracy Score (BACC) on Test Data:  0.528304379910019
```

The confusion matrix suggests that while the model is able to correctly identify a large number of non-fraudulent transactions (as indicated by the high TN count), it struggles to accurately detect fraudulent transactions, as evident from the relatively high number of false negatives (FN) compared to true positives (TP). This indicates a high false negative rate, implying that a significant portion of fraudulent transactions is being missed by the model.

ROC AUC: Test AUC of 0.59 indicates moderate performance.



While the Logistic Regression model demonstrates moderate accuracy and precision, its low recall for fraudulent transactions indicates a significant number of false negatives. This suggests that the model struggles to correctly identify fraudulent cases, which could have serious implications for fraud detection in healthcare claims data.

2.4 Insights

The Logistic Regression model demonstrates moderate accuracy in classifying transactions, correctly identifying over 62% of data points. However, its notably low recall for fraudulent transactions, at only 11%, indicates a significant challenge in effectively capturing instances of fraud within the dataset.So, this model's performance is slightly better than random chance and is almost similar to Naïve Bayes.

Model 3 : Decision Tree

3.1 Justification for choosing:

It can be used for classification as well as regression tasks, in our case works by partitioning the data based on features such as transaction amount, location, frequency of transactions, etc., to classify transactions as either fraudulent or non-fraudulent.

3.2 Tuning the model:

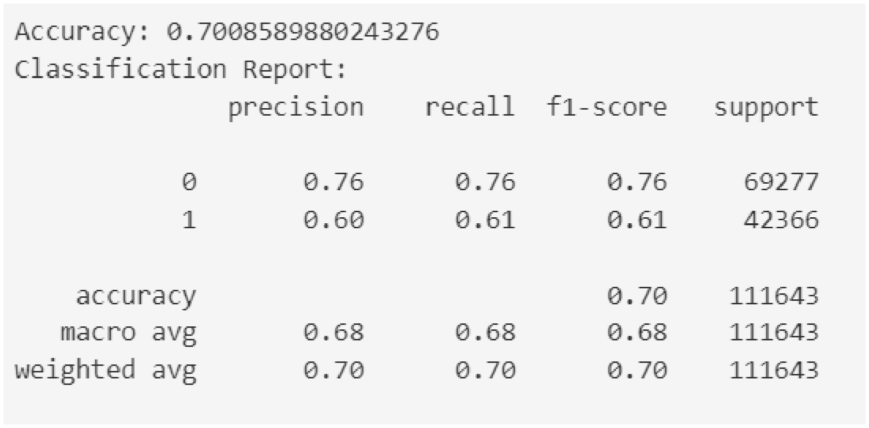
After training the Decision Tree model with the optimal hyperparameters, we evaluated its performance on the test set. The optimized model achieved an accuracy of 70%, which is better than the default Decision Tree model we trained

3.3 Effectiveness:

The classification report provides more detailed metrics:
The Decision Tree model achieves an **accuracy** of over 70%, demonstrating its capability to categorize transactions accurately.

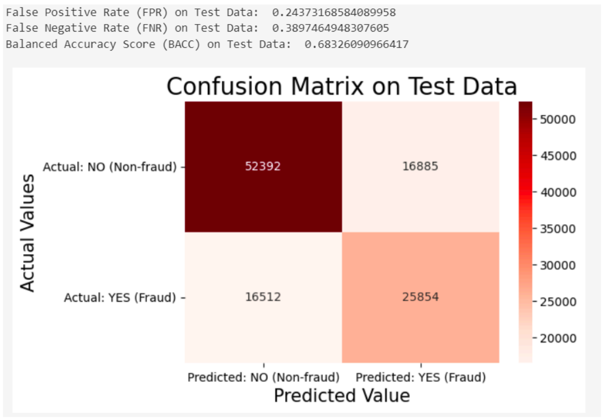
Precision and recall for non-fraudulent transactions (class 0) are both approximately 76%, indicating strong performance in detecting legitimate transactions.
Precision and recall for fraudulent transactions (class 1) are roughly 60% and 61% respectively, suggesting room for improvement in accurately identifying fraudulent cases.

Below is the Classification Report:



```
Accuracy: 0.7008589880243276
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.76      0.76     69277
           1       0.60      0.61      0.61     42366

    accuracy                           0.70    111643
   macro avg       0.68      0.68      0.68    111643
weighted avg       0.70      0.70      0.70    111643
```

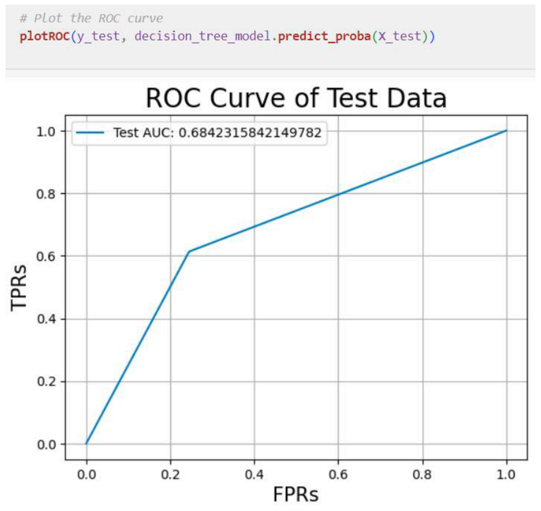
Confusion Matrix:



```
False Positive Rate (FPR) on Test Data:  0.24373168584089958
False Negative Rate (FNR) on Test Data:  0.3897464948307605
Balanced Accuracy Score (BACC) on Test Data:  0.68326090966417
```

The model's **AUC score** of 0.684 shows moderate performance in differentiating between fraudulent and non-fraudulent transactions, indicating potential for further enhancement to increase its predictive capacity.

ROC Curve:



```
# Plot the ROC curve
plotROC(y_test, decision_tree_model.predict_proba(X_test))
```

It implies that while the model's accuracy in ranking cases outperforms random guesswork, further development is necessary to increase its prediction capacity.

3.4 Insights

The Decision Tree model demonstrates promising performance, boasting an accuracy exceeding 70% and exhibiting robust precision and recall metrics for non-fraudulent transactions. However, there is room for improvement in accurately identifying fraudulent cases, as reflected by slightly lower precision and recall scores for classifying fraudulent transactions. Despite this, the model's moderate AUC score of 0.684 suggests its ability to effectively differentiate between fraudulent and non-fraudulent transactions, laying a solid foundation for further development and refinement. Overall, while the model shows considerable potential, continued optimization efforts are essential to enhance its predictive capacity and reliability in accurately detecting fraudulent activity within the dataset.

Model 4 : Random Forest

4.1 Justification for choosing:

For fraud detection, Random Forest is preferred because of its ensemble learning, which fortifies predictions by combining multiple decision trees. It delivers important insights for precisely identifying fraudulent conduct, captures non-linear correlations, and handles high-dimensional data with effectiveness. It is also suited for real-world applications due to its scalability and resistance to overfitting.

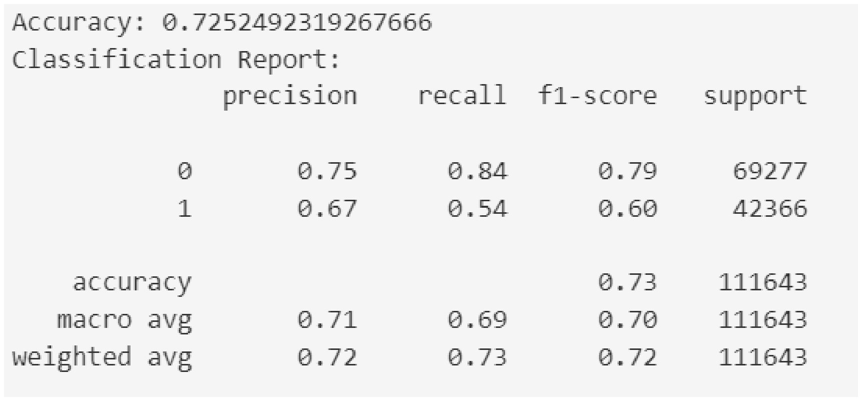
4.2 Tuning the model:

Precision and recall for fraudulent transactions (Class 1) were 67% and 54% respectively, highlighting the need for improvement in detecting fraudulent activity. Through iterative adjustments of hyperparameters such as the number of trees and maximum tree depth, We aimed to optimize the model's ability to accurately identify fraudulent cases while maintaining high accuracy overall.

4.3 Effectiveness:

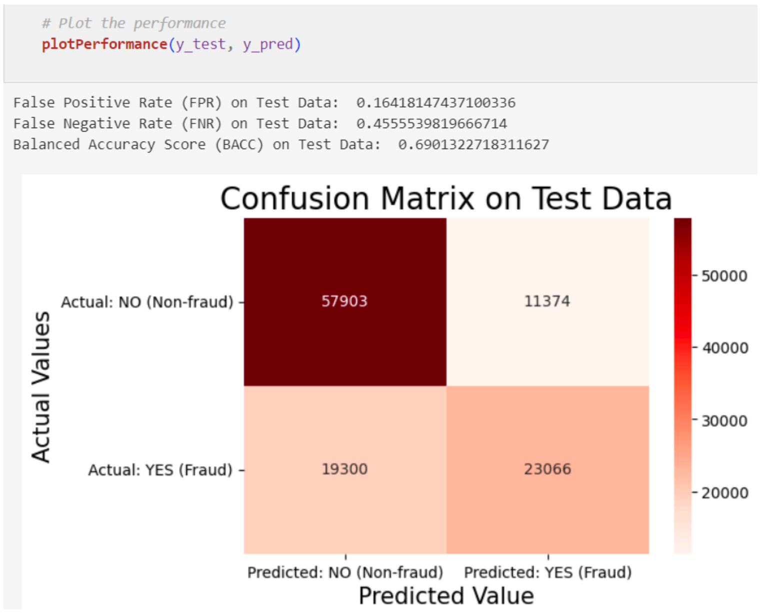
The classification report provides more detailed metrics:
Accuracy: The model achieved an accuracy of approximately 72.5%, indicating its overall effectiveness in correctly classifying transactions.
Precision and Recall for Non-Fraudulent Transactions (Class 0): With a precision of 75% and recall of 84%, the Random Forest model demonstrates strong performance in accurately identifying the majority of non-fraudulent events while minimizing false positives.
Precision and Recall for Fraudulent Transactions (Class 1): Precision and recall values for fraudulent transactions are 67% and 54% respectively. This indicates that while the model can detect some fraudulent cases, there is still room for improvement to reduce false positives and increase true positives.

Classification Report for the same:



```
Accuracy: 0.7252492319267666
Classification Report:
              precision    recall  f1-score   support

           0       0.75      0.84      0.79     69277
           1       0.67      0.54      0.60     42366

    accuracy                           0.73    111643
   macro avg       0.71      0.69      0.70    111643
weighted avg       0.72      0.73      0.72    111643
```

Confusion Matrix :



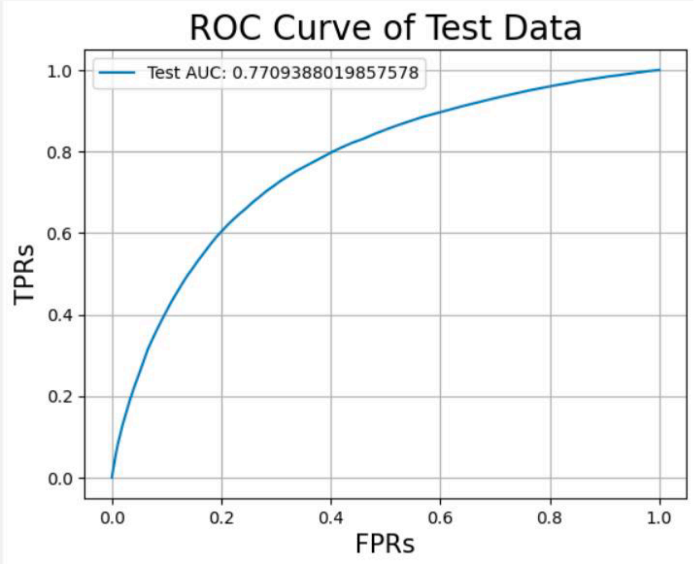
```
# Plot the performance
plotPerformance(y_test, y_pred)
```

```
False Positive Rate (FPR) on Test Data:  0.16418147437100336
False Negative Rate (FNR) on Test Data:  0.4555539819666714
Balanced Accuracy Score (BACC) on Test Data:  0.6901322718311627
```

The confusion matrix shows that 11,374 cases of fraud were incorrectly classified as false positives by the model, whereas 57,903 true positive instances of fraud were correctly identified by the algorithm. Additionally, it accurately identified 23,066 cases as true negatives but missed 19,300 cases of actual fraud, often known as false negatives. All things considered, the model shows a high recall rate (about 75%) for identifying fraud, but it also shows a significant number of false positives, indicating possible areas that might be further improved for increased precision.

ROC: Below is the ROC curve for the test data

```python
# Plot the ROC curve
plotROC(y_test, random_forest_model.predict_proba(X_test))
```



When tested on test data that has not yet been observed, the Random Forest model does a good job at differentiating between positive and negative cases, as seen by its test AUC (Area Under the Curve) of 0.77. Higher values indicate greater performance, suggesting that the model can distinguish between fraudulent and non-fraudulent cases with reasonable accuracy.

4.4 Insights

The Random Forest model demonstrates promising performance in accurately classifying both fraudulent and non-fraudulent transactions, with an overall accuracy of approximately 72.5%. While precision and recall for non-fraudulent transactions are high at 75% and 84% respectively, indicating effective identification of legitimate events, there is room for improvement in detecting fraudulent cases, as reflected by lower precision (67%) and recall (54%). However, the model's test AUC of 0.77 suggests that it effectively distinguishes between fraudulent and non-fraudulent cases with reasonable accuracy, highlighting its potential for real-world application.

Model 5 : XGBoost

5.1 Justification for choosing:

Extreme Gradient Boosting, or XGBoost, is a potent machine learning technique that performs exceptionally well in categorization jobs such as fraud detection. It provides reliable management of big datasets, effective processing, and excellent predictive accuracy. XGBoost is especially useful for capturing intricate patterns in fraud data while limiting overfitting because of its capacity to progressively assemble an ensemble of weak learners while optimizing for both bias and variance. Its versatility in addressing missing values and outliers, along with its built-in regularization approaches, further enhance its appropriateness for fraud detection jobs.

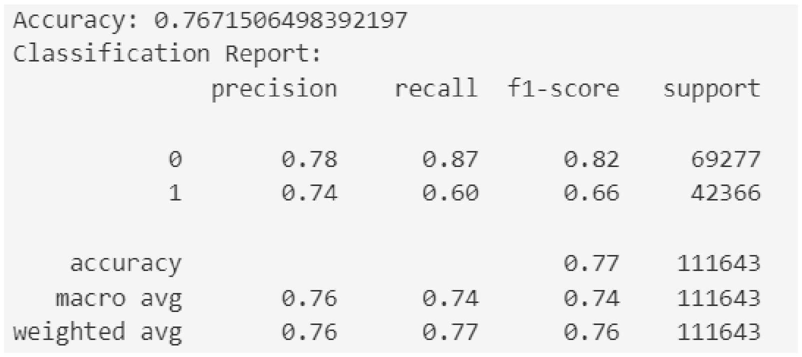
5.2 Tuning the model:

Precision and recall for fraudulent transactions (Class 1) were 67% and 54% respectively, highlighting the need for improvement in detecting fraudulent activity. Through iterative adjustments of hyperparameters such as the number of trees and maximum tree depth, We aimed to optimize the model's ability to accurately identify fraudulent cases while maintaining high accuracy overall.

5.3 Effectiveness:

The classification report provides more detailed metrics:
Accuracy: The XGBoost model achieves an accuracy of 0.77, indicating its proficiency in accurately categorizing examples into the appropriate classes.
Precision: With a precision of 0.78 for class 0 (non-fraudulent) and 0.74 for class 1 (fraudulent), the model correctly identifies 78% and 74% of instances respectively.
Recall: Class 0 recall is 0.87, indicating that 87% of all non-fraudulent cases are properly identified, while class 1 recall is 0.60, suggesting that 60% of all fraudulent cases are accurately identified by the model.
F1 Score: The F1-score, balancing between recall and precision, ensures a comprehensive evaluation of the model's performance in identifying fraudulent and non-fraudulent cases.

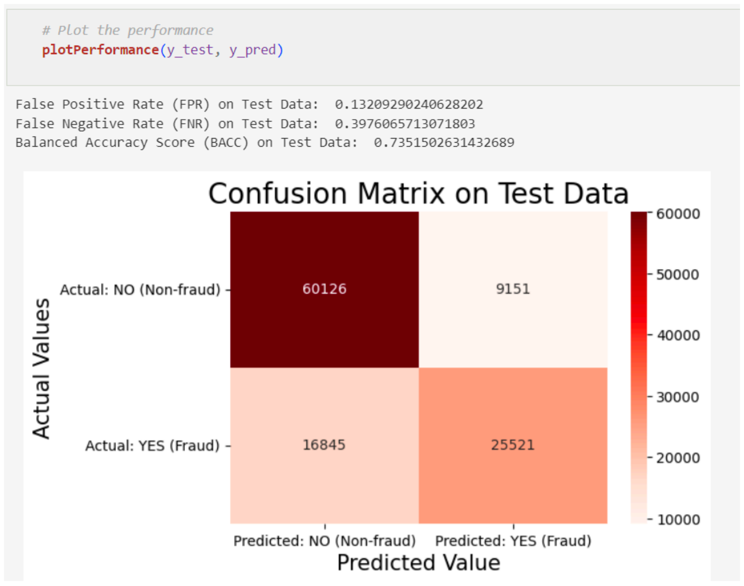
Classification Report:



```
Accuracy: 0.7671506498392197
Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.87      0.82     69277
           1       0.74      0.60      0.66     42366

    accuracy                           0.77    111643
   macro avg       0.76      0.74      0.74    111643
weighted avg       0.76      0.77      0.76    111643
```

All things considered, the model performs well in identifying fraudulent and non-fraudulent cases, with slightly superior results in categorizing non-fraudulent situations.

Confusion Matrix:



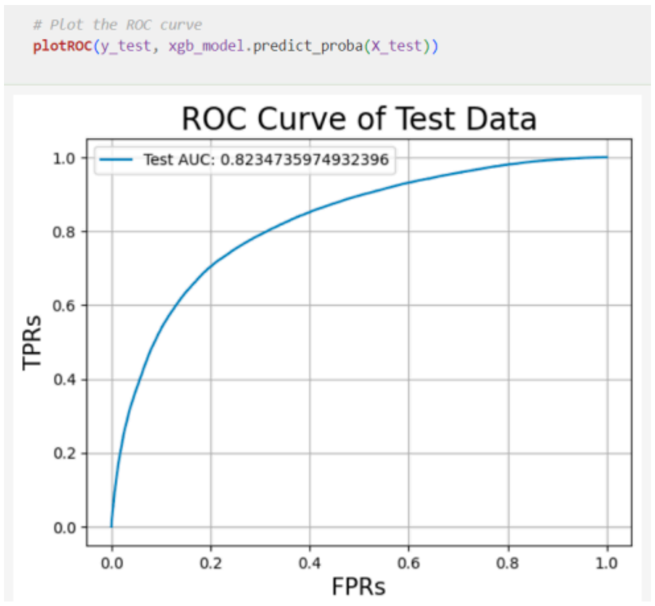
```
# Plot the performance
plotPerformance(y_test, y_pred)
```

```
False Positive Rate (FPR) on Test Data:  0.13209290240628202
False Negative Rate (FNR) on Test Data:  0.3976065713071803
Balanced Accuracy Score (BACC) on Test Data:  0.7351502631432689
```

According to the confusion matrix, 6,026 true positives (fraudulent claims correctly identified), 9,151 false positives (fraudulent claims incorrectly identified as non-fraudulent), 16,845 false negatives (fraudulent claims missed), and 25,521 true negatives (fraudulent claims correctly identified) were found out of 111,643 samples. This breakdown shows the regions where the model may have misclassified occurrences as well as how well it can distinguish between claims that are fraudulent and those that are not.

ROC AUC: With an AUC of 82.34%, the model effectively discriminates between true positive rates (TPR) and false positive rates (FPR), underlining its capacity to distinguish between fraudulent and non-fraudulent cases with good discrimination ability.

ROC:



```
# Plot the ROC curve
plotROC(y_test, xgb_model.predict_proba(X_test))
```

Overall, the model provides a valuable tool for identifying potential fraud, but additional measures may be necessary to address the false negative cases and further enhance its performance.

5.4 Insights

The classification report provides detailed metrics for the XGBoost model, including a precision of 0.78 for class 0 (non-fraudulent) and 0.74 for class 1 (fraudulent), as well as recall values of 0.87 and 0.60 respectively. These metrics offer insights into the model's performance in correctly identifying both non-fraudulent and fraudulent cases.

Overall, the model provides a valuable tool for identifying potential fraud, but additional measures may be necessary to address the false negative cases and further enhance its performance.

Model 6 : Light GBM

6.1 Justification for choosing:

Light GBM is a gradient boosting framework, can handle huge datasets effectively and can deal with imbalanced data—a prevalent feature in fraud detection tasks—it is a good fit for fraudulent classification. Furthermore, Light GBM is ideally suited for real-time fraud detection applications due to its better accuracy and speed. Its tree-based methodology makes it possible to capture intricate relationships in the data, which improves the model's prediction ability to correctly identify fraudulent activity.

6.2 Tuning the model:

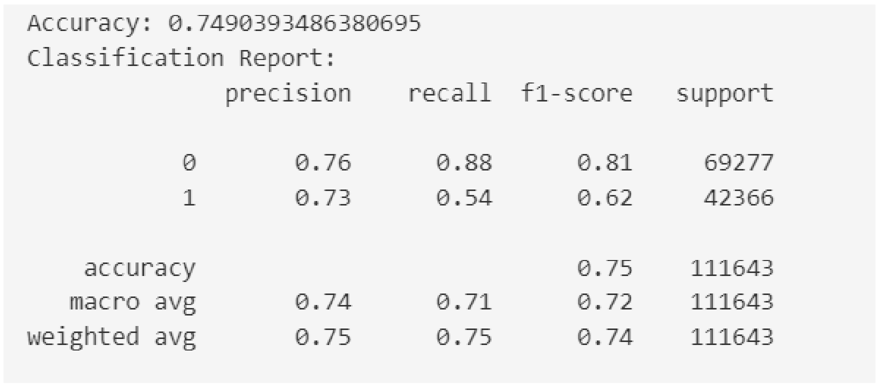
In parameter tuning for the Light GBM model, which achieved an accuracy of 74.90%, I focused on optimizing its performance metrics. With a recall of 54%, the model effectively detects over half of all real fraudulent claims, identifying 60,849 instances of fraudulent claims. However, the confusion matrix indicates 19,590 false negatives and 8,428 false positives, suggesting areas for improvement in correctly identifying fraudulent and non-fraudulent claims.

6.3 Effectiveness:

The classification report provides more detailed metrics:
Accuracy: Achieving an accuracy of 74.90%, the model demonstrates proficiency in categorizing claims as fraudulent or not, with 73% accuracy in identifying fraudulent claims.
Recall: The model's recall of 54% indicates its ability to detect over half of all real fraudulent claims, identifying 60,849 instances of fraudulent claims.

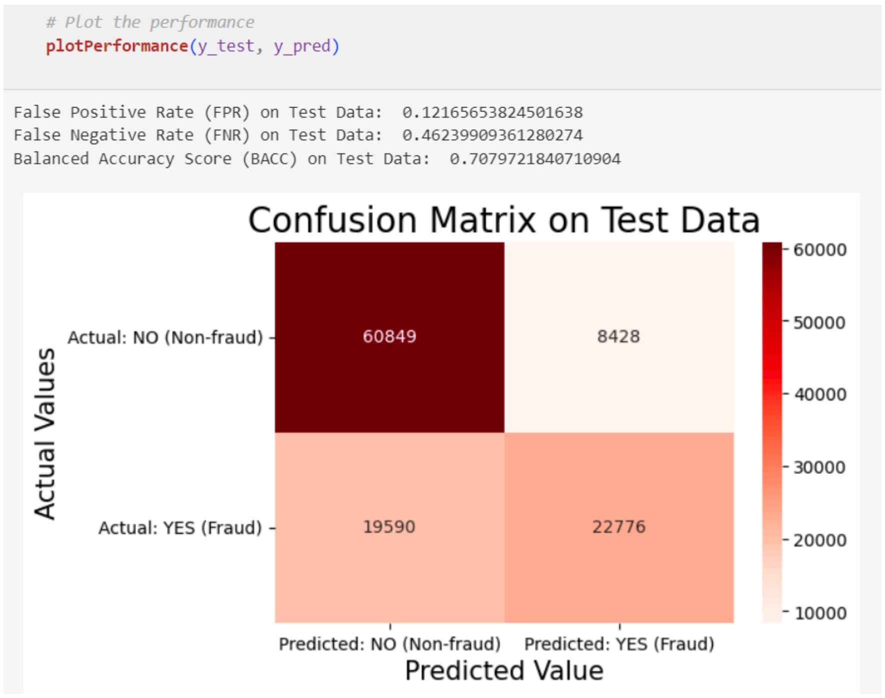
Precision: With a precision of 73% in identifying fraudulent claims, the model accurately labels a significant portion of flagged claims as fraudulent, while also identifying 22,776 non-fraudulent claims accurately.
Confusion Matrix Analysis: The model's confusion matrix reveals 19,590 false negatives and 8,428 false positives, highlighting areas for improvement in correctly identifying fraudulent and non-fraudulent claims.

Classification Report:



```
Accuracy: 0.7490393486380695
Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.88      0.81     69277
           1       0.73      0.54      0.62     42366

    accuracy                           0.75    111643
   macro avg       0.74      0.71      0.72    111643
weighted avg       0.75      0.75      0.74    111643
```

Confusion Matrix:

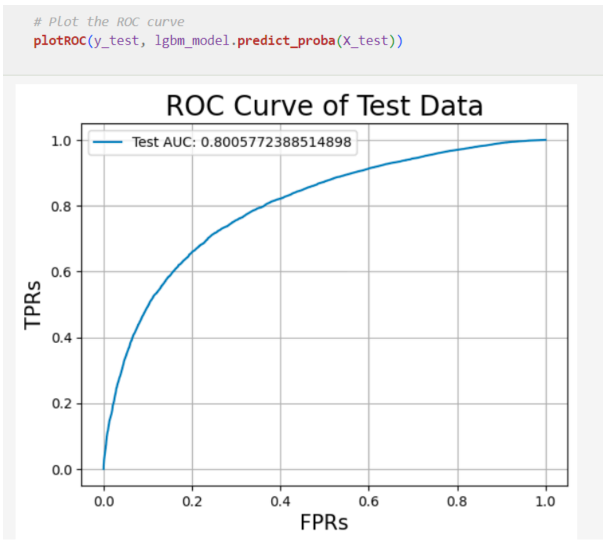


```python
# Plot the performance
plotPerformance(y_test, y_pred)
```

```
False Positive Rate (FPR) on Test Data:  0.12165653824501638
False Negative Rate (FNR) on Test Data:  0.46239909361280274
Balanced Accuracy Score (BACC) on Test Data:  0.7079721840710904
```

The Light GBM model classified 19,590 fraudulent claims as non-fraudulent (false negatives) while properly identifying 60,849 instances of fraudulent claims (true positives) in the confusion matrix. Furthermore, it labeled 8,428 non-fraudulent claims as fraudulent (false positives) despite accurately identifying 22,776 non-fraudulent claims (true negatives). Overall, the model performs well in identifying erroneous claims, as seen by a greater true positive count than false negative count.

ROC:



```
# Plot the ROC curve
plotROC(y_test, lgbm_model.predict_proba(X_test))
```

The **Light GBM** model does a good job of differentiating between fraudulent and non-fraudulent claims, as evidenced by its **AUC of 80.05%.** This statistic indicates that the model is useful for fraud detection tasks because it strikes a fair balance between the true positive rate (sensitivity) and the false positive rate (1-specificity).

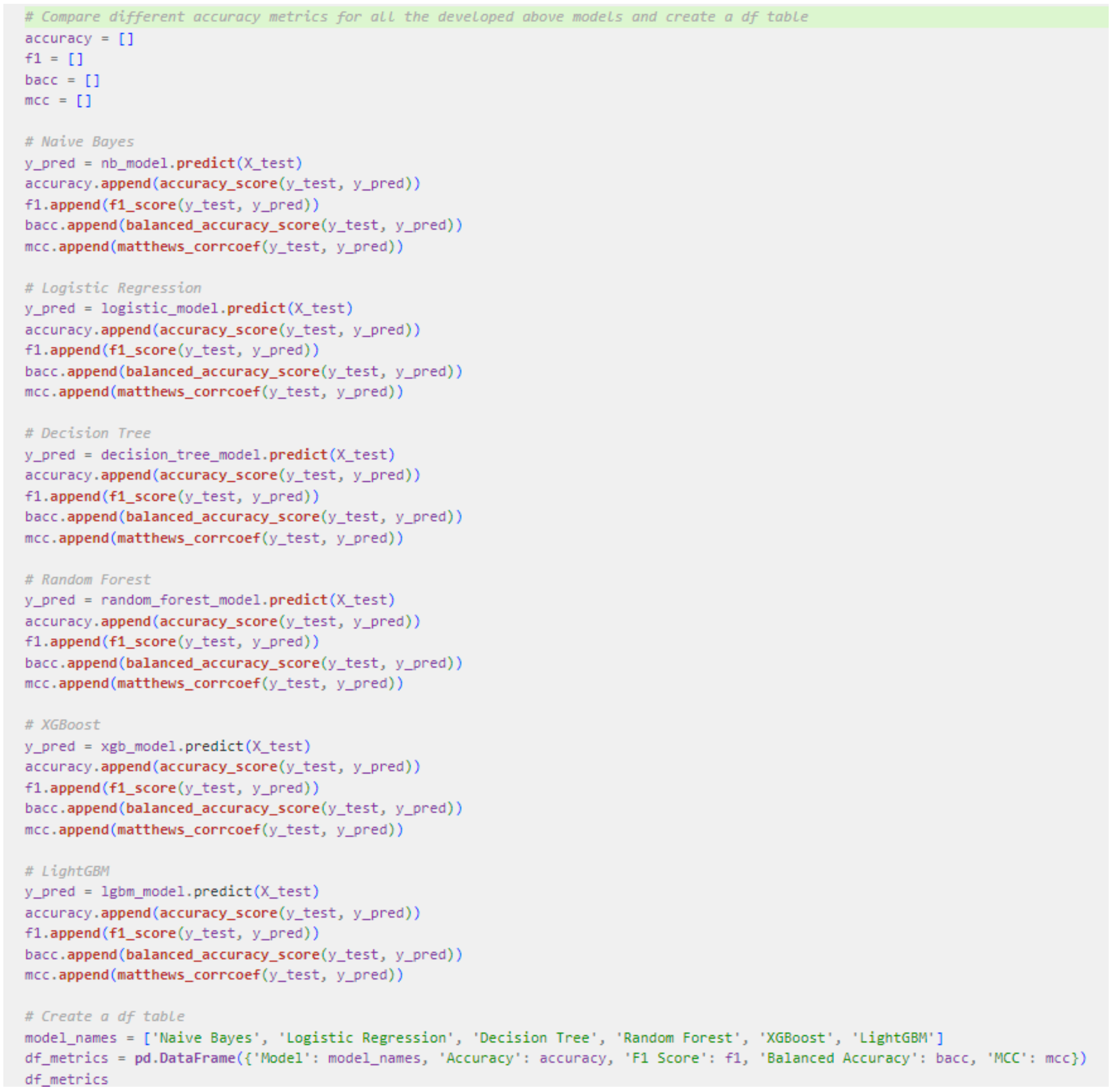
6.4 Insights

The model achieves an accuracy of 74.90% and a recall of 54%, effectively identifying over half of all real fraudulent claims. With a precision of 73%, it accurately labels flagged claims as fraudulent while minimizing false positives. However, the confusion matrix reveals 19,590 false negatives and 8,428 false positives, indicating areas for improvement in accurately identifying fraudulent and non-fraudulent claims.

We will be picking the best model by comparing all these values.

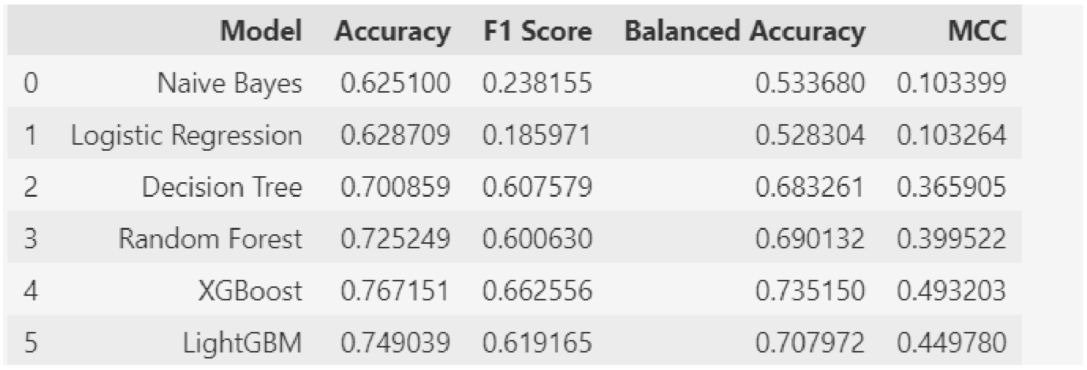
Algorithm Selection:

Below is the code to **compare** the **models:**



```python
# Compare different accuracy metrics for all the developed above models and create a df table
accuracy = []
f1 = []
bacc = []
mcc = []

# Naive Bayes
y_pred = nb_model.predict(X_test)
accuracy.append(accuracy_score(y_test, y_pred))
f1.append(f1_score(y_test, y_pred))
bacc.append(balanced_accuracy_score(y_test, y_pred))
mcc.append(matthews_corrcoef(y_test, y_pred))

# Logistic Regression
y_pred = logistic_model.predict(X_test)
accuracy.append(accuracy_score(y_test, y_pred))
f1.append(f1_score(y_test, y_pred))
bacc.append(balanced_accuracy_score(y_test, y_pred))
mcc.append(matthews_corrcoef(y_test, y_pred))

# Decision Tree
y_pred = decision_tree_model.predict(X_test)
accuracy.append(accuracy_score(y_test, y_pred))
f1.append(f1_score(y_test, y_pred))
bacc.append(balanced_accuracy_score(y_test, y_pred))
mcc.append(matthews_corrcoef(y_test, y_pred))

# Random Forest
y_pred = random_forest_model.predict(X_test)
accuracy.append(accuracy_score(y_test, y_pred))
f1.append(f1_score(y_test, y_pred))
bacc.append(balanced_accuracy_score(y_test, y_pred))
mcc.append(matthews_corrcoef(y_test, y_pred))

# XGBoost
y_pred = xgb_model.predict(X_test)
accuracy.append(accuracy_score(y_test, y_pred))
f1.append(f1_score(y_test, y_pred))
bacc.append(balanced_accuracy_score(y_test, y_pred))
mcc.append(matthews_corrcoef(y_test, y_pred))

# LightGBM
y_pred = lgbm_model.predict(X_test)
accuracy.append(accuracy_score(y_test, y_pred))
f1.append(f1_score(y_test, y_pred))
bacc.append(balanced_accuracy_score(y_test, y_pred))
mcc.append(matthews_corrcoef(y_test, y_pred))

# Create a df table
model_names = ['Naive Bayes', 'Logistic Regression', 'Decision Tree', 'Random Forest', 'XGBoost', 'LightGBM']
df_metrics = pd.DataFrame({'Model': model_names, 'Accuracy': accuracy, 'F1 Score': f1, 'Balanced Accuracy': bacc, 'MCC': mcc})
df_metrics
```

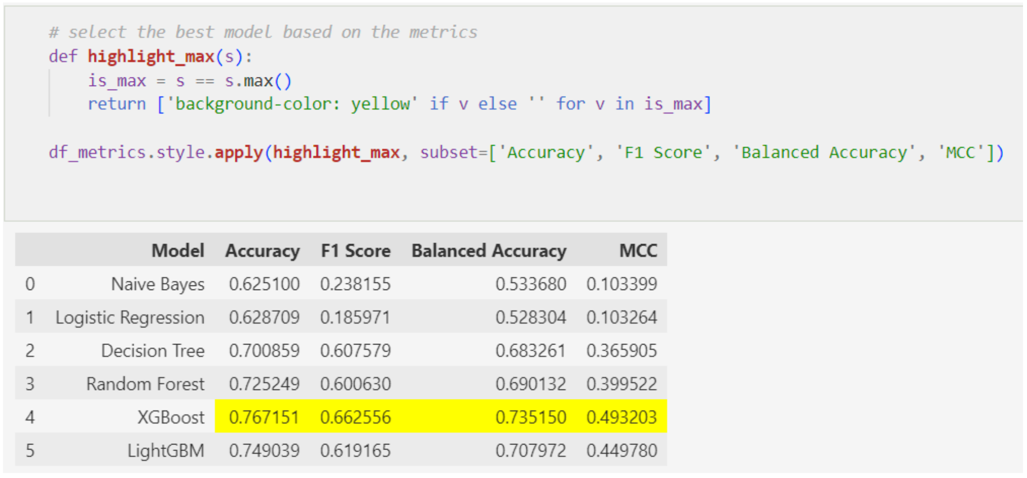
Output:



	Model	Accuracy	F1 Score	Balanced Accuracy	MCC
0	Naive Bayes	0.625100	0.238155	0.533680	0.103399
1	Logistic Regression	0.628709	0.185971	0.528304	0.103264
2	Decision Tree	0.700859	0.607579	0.683261	0.365905
3	Random Forest	0.725249	0.600630	0.690132	0.399522
4	XGBoost	0.767151	0.662556	0.735150	0.493203
5	LightGBM	0.749039	0.619165	0.707972	0.449780

XGBoost outperforms the other models taken into consideration with the highest accuracy, F1 score, balanced accuracy, and Matthew's correlation coefficient (MCC) according to these criteria. Nevertheless, alternative models like Random Forest and LightGBM exhibit commendable performance and could be considered based on additional variables like interpretability and computational cost.

Code for **Selecting** the **best model**:



```
# select the best model based on the metrics
def highlight_max(s):
    is_max = s == s.max()
    return ['background-color: yellow' if v else '' for v in is_max]

df_metrics.style.apply(highlight_max, subset=['Accuracy', 'F1 Score', 'Balanced Accuracy', 'MCC'])
```

	Model	Accuracy	F1 Score	Balanced Accuracy	MCC
0	Naive Bayes	0.625100	0.238155	0.533680	0.103399
1	Logistic Regression	0.628709	0.185971	0.528304	0.103264
2	Decision Tree	0.700859	0.607579	0.683261	0.365905
3	Random Forest	0.725249	0.600630	0.690132	0.399522
4	XGBoost	0.767151	0.662556	0.735150	0.493203
5	LightGBM	0.749039	0.619165	0.707972	0.449780

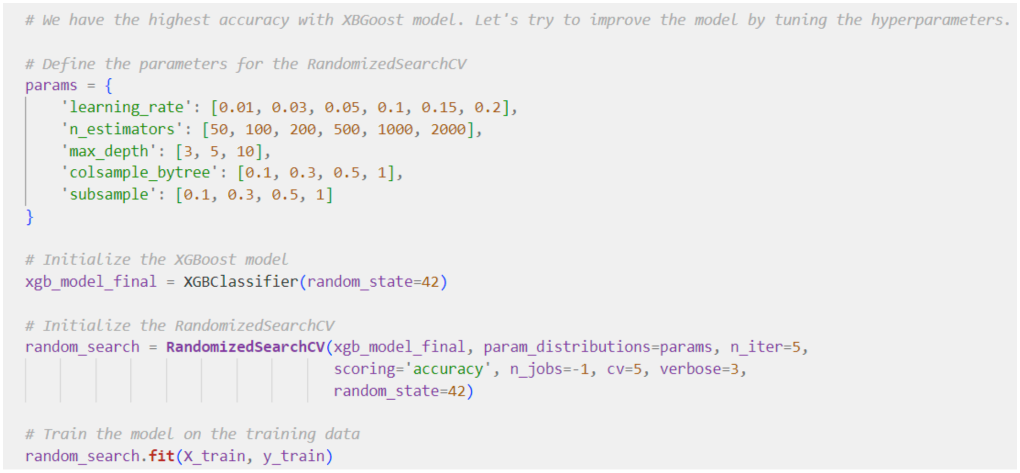
Why did we consider XG Boost Model as the best performing model?

Because XGBoost performs exceptionally well across a wide range of evaluation measures, we have selected it as our main algorithm for fraud detection. Of all the models taken into consideration, **XGBoost** has the best accuracy (**76.7%**), guaranteeing that a sizable percentage of fraudulent cases are accurately identified.

Furthermore, at **0.663**, its **F1** score—which measures memory and precision—is noticeably high, suggesting a strong capacity to maximize true positives and reduce false negatives. Furthermore, the model's performance is further validated by the fact that its balanced accuracy, which measures its ability to differentiate across classes, is 73.5%. At 0.493, the Matthews correlation coefficient (MCC), **which accounts for genuine and spurious positives as well as negatives**, is likewise better. This illustrates how XGBoost can **reduce false positives** while increasing **true positives**, giving **fraud detection professionals** a dependable tool **that reduces false accusations** and monetary losses.

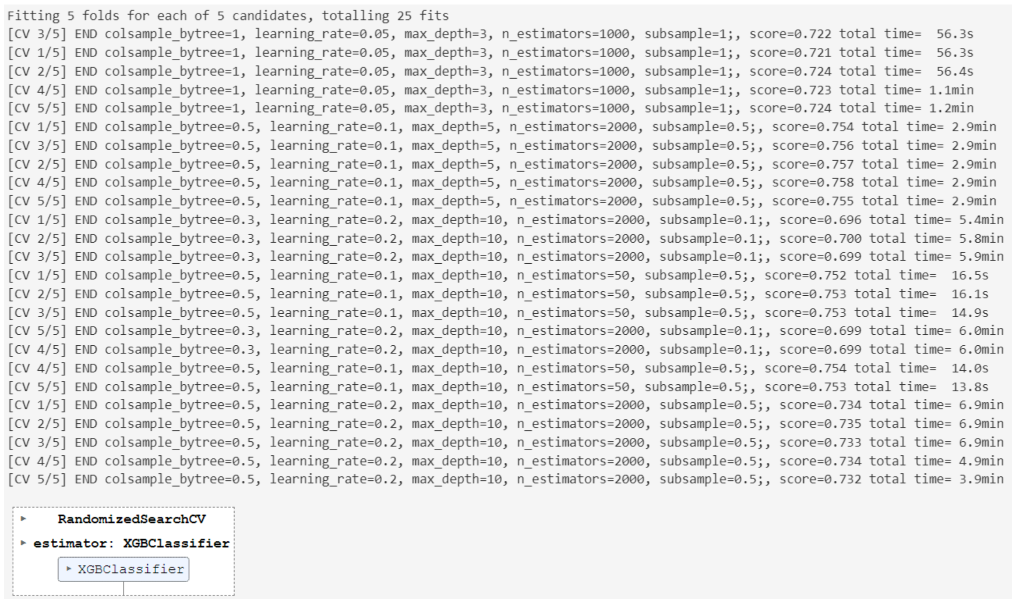
Now We try to tune the model's hyperparameters to improve the performance using Randomized Search Cross Validation.

Code for HYPER PARAMETER **Tuning**:



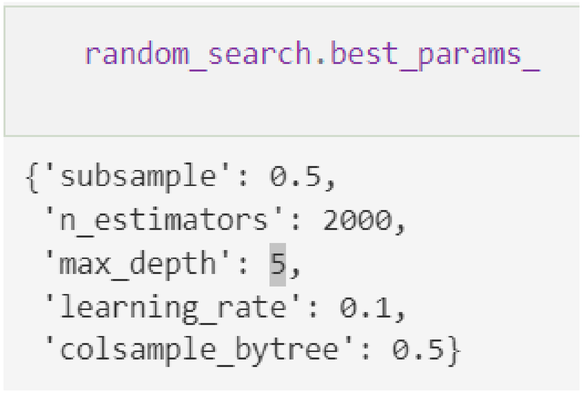
```python
# We have the highest accuracy with XBGoost model. Let's try to improve the model by tuning the hyperparameters.

# Define the parameters for the RandomizedSearchCV
params = {
    'learning_rate': [0.01, 0.03, 0.05, 0.1, 0.15, 0.2],
    'n_estimators': [50, 100, 200, 500, 1000, 2000],
    'max_depth': [3, 5, 10],
    'colsample_bytree': [0.1, 0.3, 0.5, 1],
    'subsample': [0.1, 0.3, 0.5, 1]
}

# Initialize the XGBoost model
xgb_model_final = XGBClassifier(random_state=42)

# Initialize the RandomizedSearchCV
random_search = RandomizedSearchCV(xgb_model_final, param_distributions=params, n_iter=5,
                                   scoring='accuracy', n_jobs=-1, cv=5, verbose=3,
                                   random_state=42)

# Train the model on the training data
random_search.fit(X_train, y_train)
```

Output:



```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV 3/5] END colsample_bytree=1, learning_rate=0.05, max_depth=3, n_estimators=1000, subsample=1;, score=0.722 total time=  56.3s
[CV 1/5] END colsample_bytree=1, learning_rate=0.05, max_depth=3, n_estimators=1000, subsample=1;, score=0.721 total time=  56.3s
[CV 2/5] END colsample_bytree=1, learning_rate=0.05, max_depth=3, n_estimators=1000, subsample=1;, score=0.724 total time=  56.4s
[CV 4/5] END colsample_bytree=1, learning_rate=0.05, max_depth=3, n_estimators=1000, subsample=1;, score=0.723 total time= 1.1min
[CV 5/5] END colsample_bytree=1, learning_rate=0.05, max_depth=3, n_estimators=1000, subsample=1;, score=0.724 total time= 1.2min
[CV 1/5] END colsample_bytree=0.5, learning_rate=0.1, max_depth=5, n_estimators=2000, subsample=0.5;, score=0.754 total time= 2.9min
[CV 3/5] END colsample_bytree=0.5, learning_rate=0.1, max_depth=5, n_estimators=2000, subsample=0.5;, score=0.756 total time= 2.9min
[CV 2/5] END colsample_bytree=0.5, learning_rate=0.1, max_depth=5, n_estimators=2000, subsample=0.5;, score=0.757 total time= 2.9min
[CV 4/5] END colsample_bytree=0.5, learning_rate=0.1, max_depth=5, n_estimators=2000, subsample=0.5;, score=0.758 total time= 2.9min
[CV 5/5] END colsample_bytree=0.5, learning_rate=0.1, max_depth=5, n_estimators=2000, subsample=0.5;, score=0.755 total time= 2.9min
[CV 1/5] END colsample_bytree=0.3, learning_rate=0.2, max_depth=10, n_estimators=2000, subsample=0.1;, score=0.696 total time= 5.4min
[CV 2/5] END colsample_bytree=0.3, learning_rate=0.2, max_depth=10, n_estimators=2000, subsample=0.1;, score=0.700 total time= 5.8min
[CV 3/5] END colsample_bytree=0.3, learning_rate=0.2, max_depth=10, n_estimators=2000, subsample=0.1;, score=0.699 total time= 5.9min
[CV 1/5] END colsample_bytree=0.5, learning_rate=0.1, max_depth=10, n_estimators=50, subsample=0.5;, score=0.752 total time=  16.5s
[CV 2/5] END colsample_bytree=0.5, learning_rate=0.1, max_depth=10, n_estimators=50, subsample=0.5;, score=0.753 total time=  16.1s
[CV 3/5] END colsample_bytree=0.5, learning_rate=0.1, max_depth=10, n_estimators=50, subsample=0.5;, score=0.753 total time=  14.9s
[CV 5/5] END colsample_bytree=0.3, learning_rate=0.2, max_depth=10, n_estimators=2000, subsample=0.1;, score=0.699 total time= 6.0min
[CV 4/5] END colsample_bytree=0.3, learning_rate=0.2, max_depth=10, n_estimators=2000, subsample=0.1;, score=0.699 total time= 6.0min
[CV 4/5] END colsample_bytree=0.5, learning_rate=0.1, max_depth=10, n_estimators=50, subsample=0.5;, score=0.754 total time=  14.0s
[CV 5/5] END colsample_bytree=0.5, learning_rate=0.1, max_depth=10, n_estimators=50, subsample=0.5;, score=0.753 total time=  13.8s
[CV 1/5] END colsample_bytree=0.5, learning_rate=0.2, max_depth=10, n_estimators=2000, subsample=0.5;, score=0.734 total time= 6.9min
[CV 2/5] END colsample_bytree=0.5, learning_rate=0.2, max_depth=10, n_estimators=2000, subsample=0.5;, score=0.735 total time= 6.9min
[CV 3/5] END colsample_bytree=0.5, learning_rate=0.2, max_depth=10, n_estimators=2000, subsample=0.5;, score=0.733 total time= 6.9min
[CV 4/5] END colsample_bytree=0.5, learning_rate=0.2, max_depth=10, n_estimators=2000, subsample=0.5;, score=0.734 total time= 4.9min
[CV 5/5] END colsample_bytree=0.5, learning_rate=0.2, max_depth=10, n_estimators=2000, subsample=0.5;, score=0.732 total time= 3.9min
```

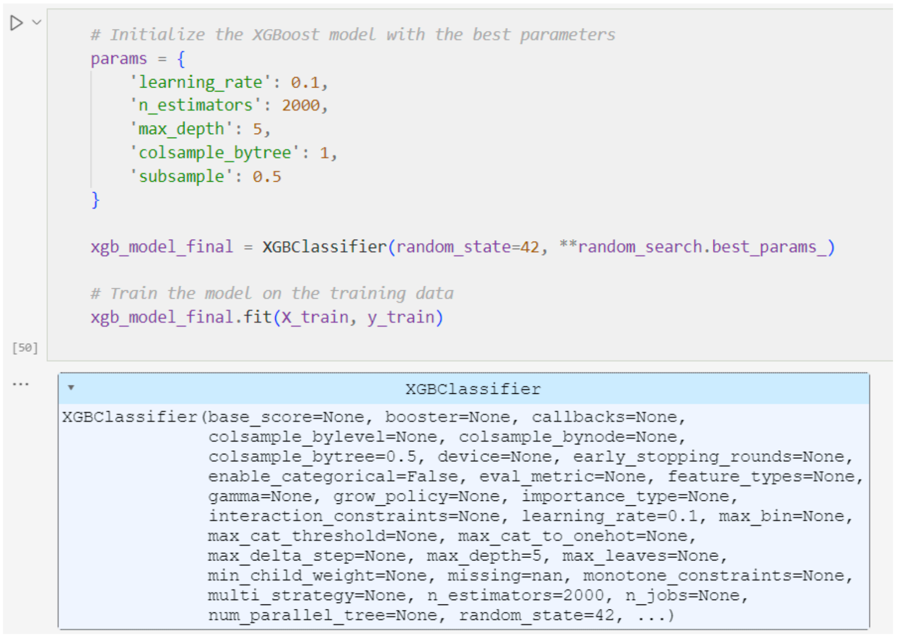
Getting the best hyperparameters for the model:



```
random_search.best_params_
```

```
{'subsample': 0.5,
 'n_estimators': 2000,
 'max_depth': 5,
 'learning_rate': 0.1,
 'colsample_bytree': 0.5}
```

Initializing the **best** hyperparameters to the **selected** model **XGB**:



```python
# Initialize the XGBoost model with the best parameters
params = {
    'learning_rate': 0.1,
    'n_estimators': 2000,
    'max_depth': 5,
    'colsample_bytree': 1,
    'subsample': 0.5
}

xgb_model_final = XGBClassifier(random_state=42, **random_search.best_params_)

# Train the model on the training data
xgb_model_final.fit(X_train, y_train)
```

[50]

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.5, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, 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=5, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=2000, n_jobs=None,
              num_parallel_tree=None, random_state=42, ...)
```

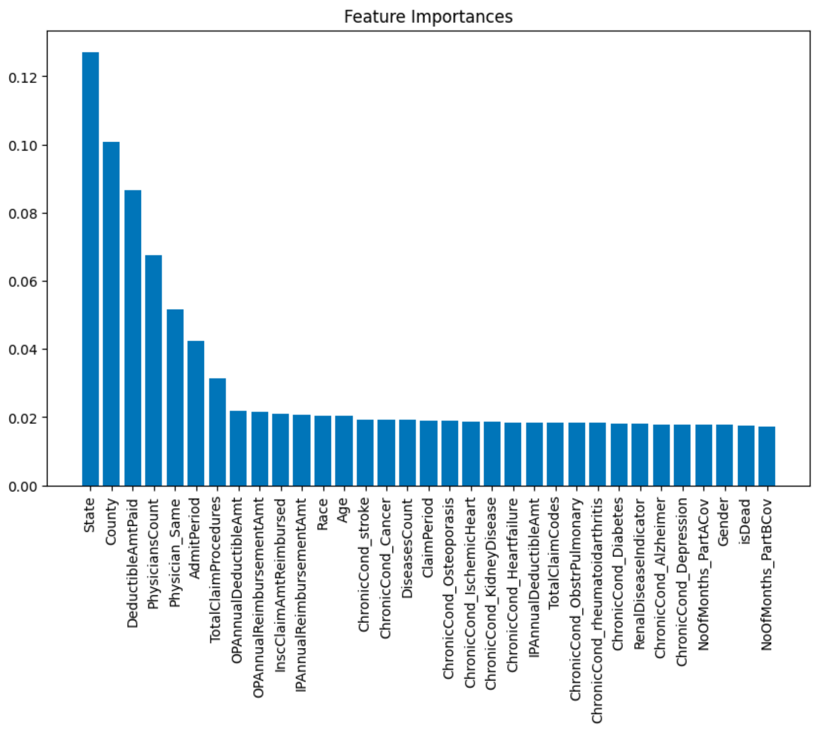
To understand the **Transparency** of the **model** we **analyzing** the Feature Importance:

Extracting and visualizing feature importance is crucial for both data scientists and stakeholders. It provides insights into which features have the most significant impact on the model's predictions. For data scientists, understanding feature importance helps in feature selection, model refinement, and identifying potential areas for further investigation. Stakeholders benefit from this by gaining transparency into the model's decision-making process and understanding which factors drive its predictions. This transparency builds trust and confidence in the model's reliability and allows stakeholders to make informed decisions based on the identified influential features.

Below is the **Code** for getting **feature importance:**

```python
# get the most important features
features = X_train.columns
importances = xgb_model_final.feature_importances_
indices = np.argsort(importances)[::-1]

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.title("Feature Importances")
plt.bar(range(X_train.shape[1]), importances[indices])
plt.xticks(range(X_train.shape[1]), [features[i] for i in indices], rotation=90)
plt.show()
```

Output:

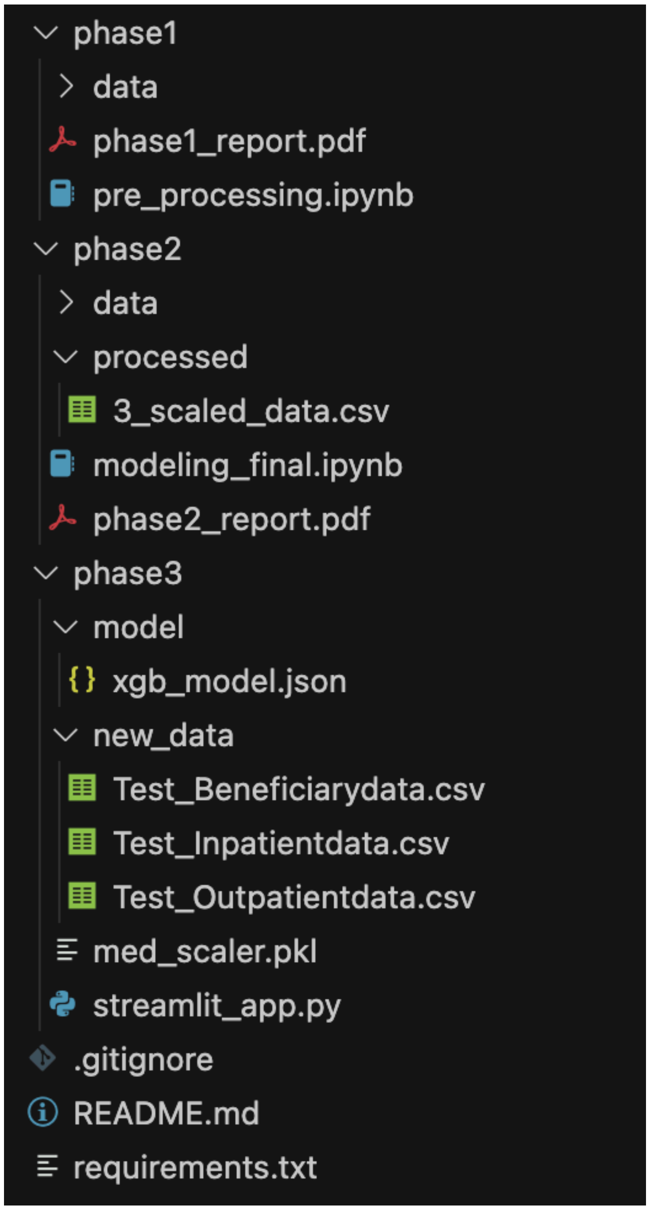


Based on the **feature importance study**, it seems that **geographic location** like **State**, **Country** followed by **Deductible Amount Paid** play a big role in spotting possible fraud. This suggests that in order to take advantage of holes in the system, fraudsters **can modify deductible payments** or **target particular areas.** Furthermore, the conduct of healthcare providers, including the number of treatments they undertake, may also be a sign of fraud. By identifying these trends, we can improve our procedures for spotting fraudulent activity in the healthcare system and put policies in place to lessen the likelihood of it happening.

Project Set up and Booting:

We have provided **README.md** which is fully documented code and working instructions to demo/use our finished product.

Folder Structure:



```
phase1
  data
  phase1_report.pdf
  pre_processing.ipynb
phase2
  data
  processed
    3_scaled_data.csv
  modeling_final.ipynb
  phase2_report.pdf
phase3
  model
    xgb_model.json
  new_data
    Test_Beneficiarydata.csv
    Test_Inpatientdata.csv
    Test_Outpatientdata.csv
  med_scaler.pkl
  streamlit_app.py
.gitignore
README.md
requirements.txt
```

- The src folder had 3 folders phase1,Phase2 and phase3.
- **Phase 1**: Contains code for phase 1 EDA of our project.
  - phase1_report.pdf - Phase1 report.

    - pre_processing.ipynb - The EDA code of phase 1.
    - data - contains the data files (.csv files)
- **Phase 2:** Contains code for phase 2 Data modeling of our project.
    - data - contains the data files (.csv files)
    - Processed - contains the processed and ready data file.
    - modeling_final.ipynb - The modeling code of phase 2.
    - phase2_report.pdf - Phase2 report.
- **Phase 3:** Contains code for phase 3 of our project.
    - model - contains json file.
    - new_datamed_scaler.pkl
    - new_data - contains the data file.
    - streamlit_app.py - The code of phase 3 web application.
- **Report** - present outside src folder is the final project report.
- **README.md** - contains the instructions to set up and run our project code.

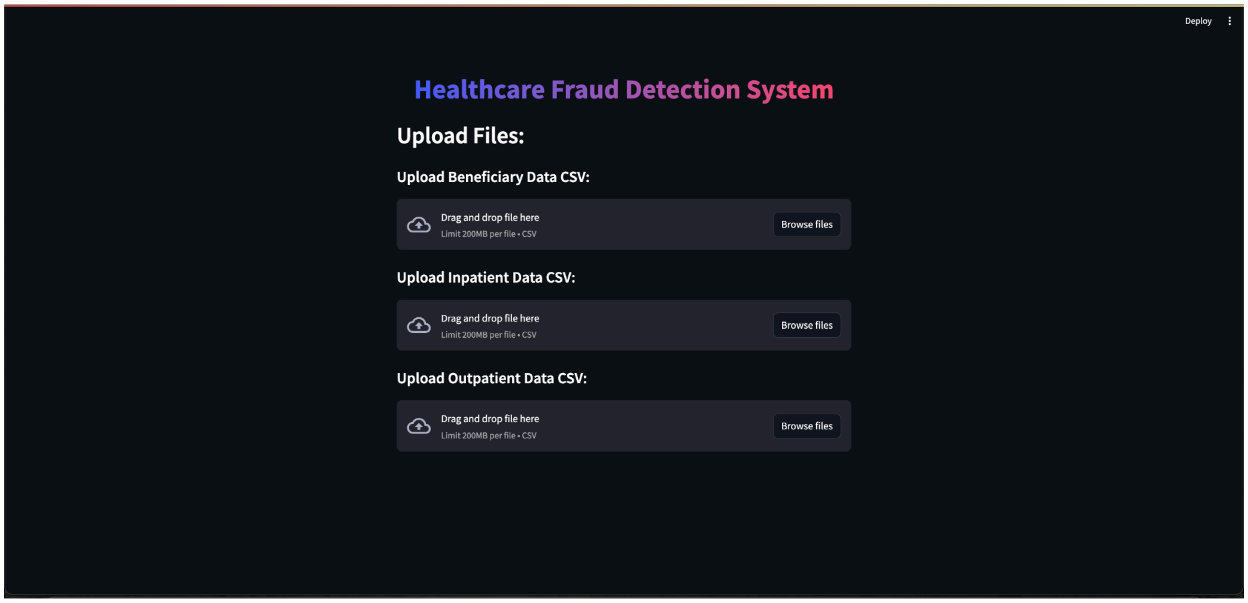
Github link- https://github.com/amilaub/healthcare-medical-fraud-detection

Web Application Implementation:

The core functionality of our web application is to accept CSV files containing inpatient claims, outpatient claims, and beneficiary details as input. Upon uploading the data, the application provides a preview of the dataset and generates predictions regarding the likelihood of fraudulent activity. Users can visualize the results and access detailed insights into the factors contributing to the prediction.

Application Walkthrough:

Data Input page: The web page where it accepts user input files for inpatient claims, outpatient claims, and beneficiary details.



Healthcare Fraud Detection System

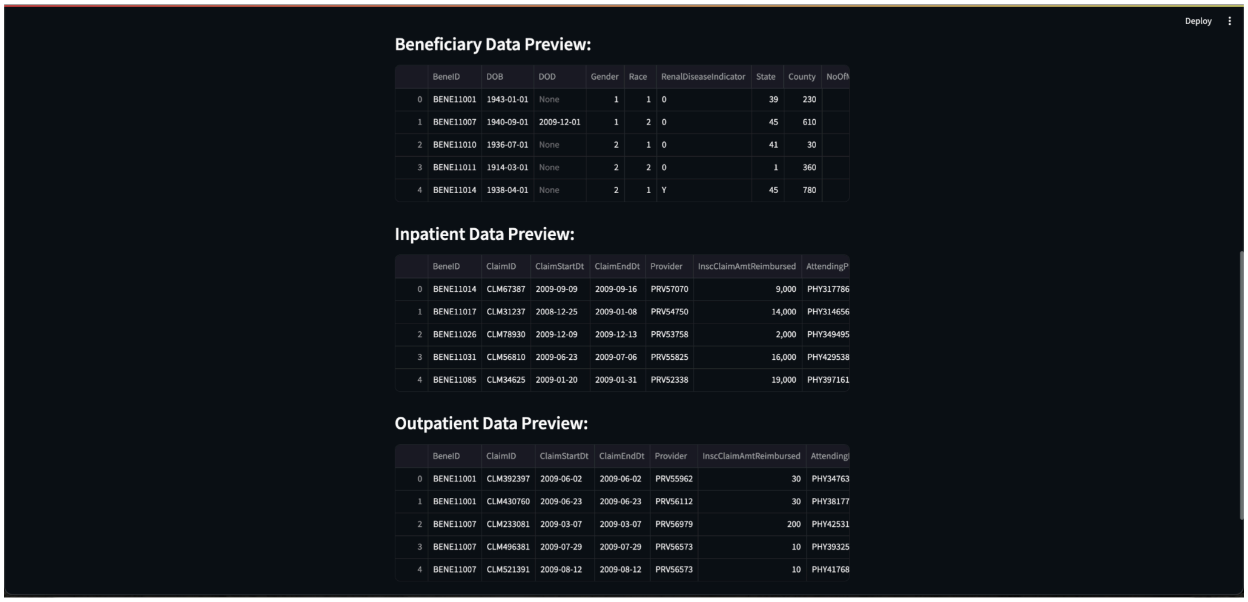
Upload Files:

Upload Beneficiary Data CSV:

Upload Inpatient Data CSV:

Upload Outpatient Data CSV:

Data preview Page: The web portal displaying the preview of the data uploaded by the user.



Beneficiary Data Preview:

	BeneID	DOB	DOD	Gender	Race	RenalDiseaseIndicator	State	County	NoOf
0	BENE11001	1943-01-01	None	1	1	0	39	230	
1	BENE11007	1940-09-01	2009-12-01	1	2	0	45	610	
2	BENE11010	1936-07-01	None	2	1	0	41	30	
3	BENE11011	1914-03-01	None	2	2	0	1	360	
4	BENE11014	1938-04-01	None	2	1	Y	45	780	

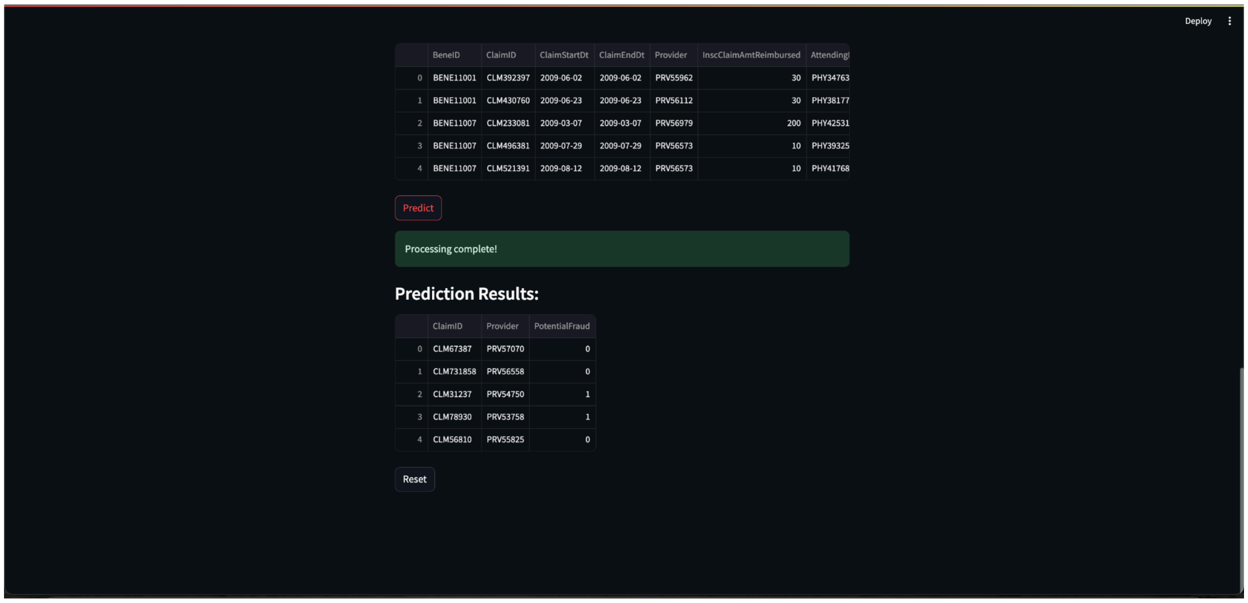
Inpatient Data Preview:

	BeneID	ClaimID	ClaimStartDt	ClaimEndDt	Provider	InscClaimAmtReimbursed	AttendingP
0	BENE11014	CLM67387	2009-09-09	2009-09-16	PRV57070	9,000	PHY317786
1	BENE11017	CLM31237	2008-12-25	2009-01-08	PRV54750	14,000	PHY314656
2	BENE11026	CLM78930	2009-12-09	2009-12-13	PRV53758	2,000	PHY349495
3	BENE11031	CLM56810	2009-06-23	2009-07-06	PRV55825	16,000	PHY429538
4	BENE11085	CLM34625	2009-01-20	2009-01-31	PRV52338	19,000	PHY397161

Outpatient Data Preview:

	BeneID	ClaimID	ClaimStartDt	ClaimEndDt	Provider	InscClaimAmtReimbursed	Attending
0	BENE11001	CLM392397	2009-06-02	2009-06-02	PRV55962	30	PHY34763
1	BENE11001	CLM430760	2009-06-23	2009-06-23	PRV56112	30	PHY38177
2	BENE11007	CLM233081	2009-03-07	2009-03-07	PRV56979	200	PHY42531
3	BENE11007	CLM496381	2009-07-29	2009-07-29	PRV56573	10	PHY39325
4	BENE11007	CLM521391	2009-08-12	2009-08-12	PRV56573	10	PHY41768

Output Page: Users can visualize the results and access detailed insights into the factors contributing to the prediction.



	BeneID	ClaimID	ClaimStartDt	ClaimEndDt	Provider	InscClaimAmtReimbursed	AttendingI
0	BENE11001	CLM392397	2009-06-02	2009-06-02	PRV55962	30	PHY34763
1	BENE11001	CLM430760	2009-06-23	2009-06-23	PRV56112	30	PHY38177
2	BENE11007	CLM233081	2009-03-07	2009-03-07	PRV56979	200	PHY42531
3	BENE11007	CLM496381	2009-07-29	2009-07-29	PRV56573	10	PHY39325
4	BENE11007	CLM521391	2009-08-12	2009-08-12	PRV56573	10	PHY41768

Processing complete!

Prediction Results:

	ClaimID	Provider	PotentialFraud
0	CLM67387	PRV57070	0
1	CLM731858	PRV56558	0
2	CLM31237	PRV54750	1
3	CLM78930	PRV53758	1
4	CLM56810	PRV55825	0

Conclusion:

The **Journey** of **our analysis** started with the gathering of data from healthcare insurance claims, which was then examined, cleansed, and preprocessed to extract useful information. Missing values were addressed throughout the cleaning process, and pertinent characteristics were chosen for additional examination. A deeper comprehension of the dataset was made possible via exploratory data analysis (EDA), which highlighted trends, distributions, and possible correlations between variables.

Identifying potential fraud indications including geographic location, deductible amounts, and provider conduct are among the analysis's key results. Predictive models were created to identify fictitious insurance claims by utilizing machine learning algorithms including Random Forest, XGBoost, and LightGBM. The accuracy, precision, and recall of these models varied, with XGBoost showing the highest levels of performance.

Overall, this analysis offers healthcare stakeholders—policymakers, insurers, and providers—actionable insights to enhance fraud detection systems, enhance patient care outcomes, and maximize resource use. Through the use of cutting-edge machine learning techniques and the analysis of Medicare claims data, we have created prediction models that effectively detect possible fraudulent activity. These models establish the foundation for ongoing monitoring and development and highlight the value of data-driven strategies in tackling issues facing the healthcare industry. We can promote openness, confidence, and effectiveness in the provision of healthcare by taking proactive steps to reduce fraud, which will benefit both patients and healthcare professionals. We want to build a more affordable and secure healthcare system by working together and making well-informed decisions. **This will eventually improve the availability and caliber of healthcare services for all beneficiaries.**

References:

1. XGBoost: A Scalable Tree Boosting System- https://arxiv.org/abs/1603.02754
2. Random Forests and Decision Trees: https://ijcsi.org/issues.php
3.RandomForest:https://www.researchgate.net/publication/259235118_Random_Forests_and_Decision_Trees
4.LightGBM:https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf
5. CSE 587 - Data Intensive Computing: Lecture Slides
6. Streamlit: https://docs.streamlit.io/