

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.

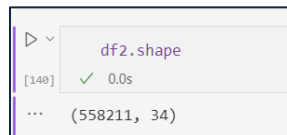
This dataset comprises of:

```
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 doesn'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.

Evaluation Metrics:

- **Precision** measures the proportion of True Positive predictions (Refers to the case where the model correctly predicts the positive class) out of all positive predictions being made by the model. **High Precision** means a high proportion of the instances **predicted** by the **model** are **indeed Fraudulent** (It minimizes the false alarms or False Positives). **Low Precision** means that a significant portion of the instances **predicted as Fraudulent** by the model **are not Actually Fraudulent**.
- **Accuracy** measures the proportion of correctly classified instances out of the total number of instances. It tells the number of correctly predicted instances to the total number of predicted instances. But it measures **the overall correctness of the model's predictions, regardless of the class**. High accuracy indicates that the model is **making correct predictions** for **both** fraudulent and non-fraudulent **transactions**. While **high** accuracy is desirable, it **can** be **misleading** if the dataset is **imbalanced**. Low accuracy may indicate that the model is **struggling** to correctly classify **both** fraudulent and non-fraudulent transactions.
- **Recall** or sensitivity measures the proportion of True Positive predictions out of all actual positive instances, or it can be said that it measures the proportion of actual Fraudulent transactions that are correctly identified by the model. **High** recall indicates that the model is **effective** at capturing a large portion of **actual fraud** cases. It means that the model correctly identifies most of the fraudulent transactions, minimizing the number of missed fraud cases. **Low** recall means that the model **fails** to capture **many actual fraud** cases, resulting in a higher number of missed fraud cases (false negatives).
- **Balance Between Recall and Precision:** A rise in recall could result in a fall in precision, and vice versa. For instance, to catch more real fraud cases (high recall), a model that flags a large number of transactions as fraudulent may also include more false positives, which would reduce precision. On the other hand, a model with a high precision focus might be

more cautious when classifying transactions as fraudulent, which could result in the missing of some actual fraud incidents (lower recall).

- **Identifying the Ideal Balance:** The ideal ratio of recall to precision relies on the goals and limitations of the fraud detection task. For instance, precision might be given priority to reduce false alarms in situations when the cost of false positives—erroneously reporting normal transactions as fraudulent—is significant. Recall, on the other hand, might be given priority in situations where it would be more expensive to overlook real fraud cases to guarantee that the majority of them are found.
- **F1 score** is a single metric that combines both precision and recall into a single value, providing a balanced measure of a classifier's performance. When the classifier obtains a **high F1 score**, it strikes a balance between high recall and high precision. This indicates that the classifier minimizes the number of false positives (low false positive rate) while successfully identifying a significant percentage of real illegal transactions (high recall). Stated differently, the classifier is deemed dependable for fraud detection due to its low false positive rate (precision) and low false negative rate (recall). On the other hand, a **low F1 score** indicates that your classifier can have poor recall, low precision, or both. This may suggest that the classifier has difficulty accurately identifying fraudulent transactions while preventing false alerts. Practically speaking, a low F1 score could indicate that the classifier mistakenly flags a sizable number of valid transactions as fraudulent (high false positive rate) or misses many true fraud incidents (high false negative rate).

Receiver Operating Characteristic (ROC) curve plots the true positive rate (recall) against the false positive rate for different threshold values. Area Under the ROC Curve (AUC) measures the area under the ROC curve, providing an aggregate measure of the model's performance across all possible threshold values.

- **ROC:** Plotting the true positive rate (also known as recall or sensitivity) versus the false positive rate (also known as specificity) for various classifier threshold values is what the ROC curve does. Every point on the ROC curve denotes a pair of parameters related to sensitivity and specificity that match a given threshold. The curve represents the trade-off that occurs when the decision threshold is changed between accurately recognizing genuine positive instances (fraudulent transactions) and mistakenly flagging false positive cases (non-fraudulent transactions).
- **Area Under the ROC Curve, or AUC for short:** is a statistical assessment of the model's performance across all potential threshold values. It does this by calculating the area under the ROC curve. $AUC = 1$ denotes a perfect classifier with a true positive rate of 1 (100% sensitivity) and a false positive rate of 0 (100% specificity). The AUC scales from 0 to 1. This suggests that the model is fully capable of differentiating between transactions that are fraudulent and those that are not. Random classifier with an AUC of 0.5 performs no better than chance. The diagonal line ($y = x$) is followed by the ROC curve, which shows that the model's predictions are no more accurate than a random guess. $AUC > 0.5$ indicates that the classifier outperforms arbitrary guesswork. The classifier's ability to discriminate

between fraudulent and non-fraudulent transactions across a range of threshold values improves with a higher AUC.

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 Modelling process:

Models:

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

1. **Naïve Bayes:** it is a simple probabilistic classifier based on Bayes Theorem with an assumption of Independence among the predictors. Based on the information provided by a transaction's features, Naive Bayes determines the likelihood that a transaction belongs to either the fraudulent or non-fraudulent class. Using the Bayes theorem, it estimates the posterior probability of each class given the features and computes the conditional probabilities of each feature given the class.

Below Code is used to Achieve this:

```
# Using Naive Bayes

# Create a Naive Bayes model
nb_model = GaussianNB()

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

# Predict on the testing data
y_pred = nb_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

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	

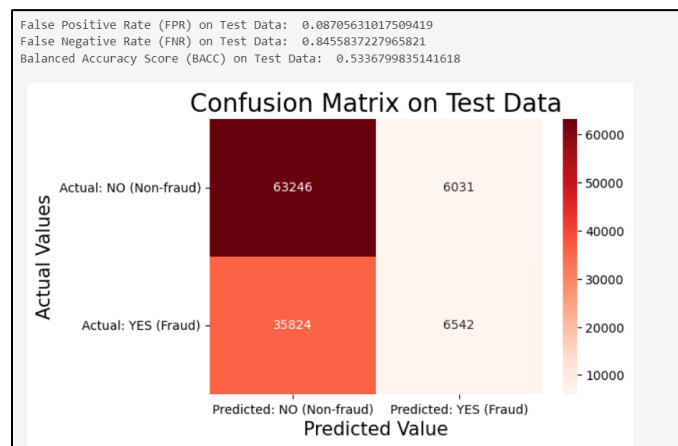
Accuracy of 0.625 which means 63% percentage of transactions the model classified correctly (both fraudulent and non-fraudulent) which is just above the random guessing baseline (50% for two classes).

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)

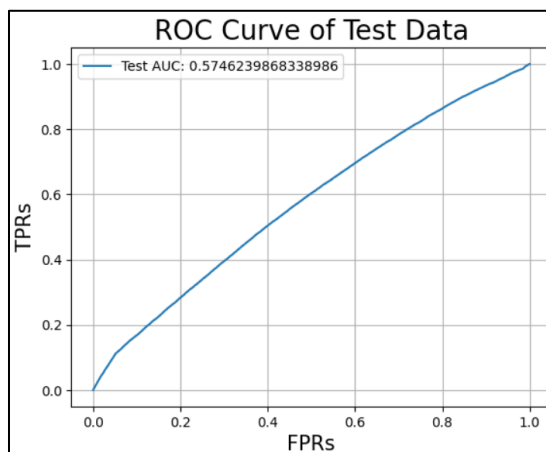
Recall for class 0 (non-fraudulent) is 91%. This means that the model identified 91% of the actual non-fraudulent transactions correctly. However, the recall for class 1 (fraudulent) is very low at 15%. This signifies that the model is missing a significant portion of fraudulent cases (false negatives).

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.

Confusion Matrix:



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.

So, proceeding with the next model for better performance

- Logistic Regression:** 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.

Below code is used to achieve this:

```
# Using Logistic Regression

# Initialize the Logistic Regression model
logistic_model = LogisticRegression(random_state=42)

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

# Predict on the testing data
y_pred = logistic_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Results:

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

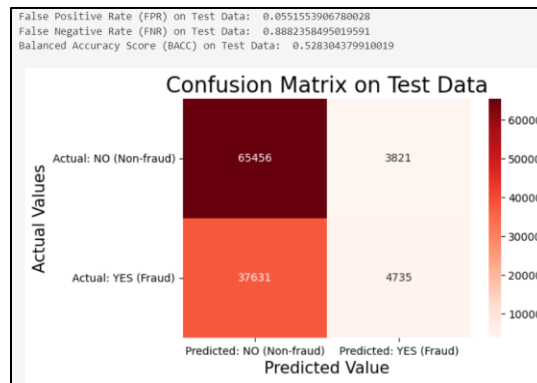
     0       0.63       0.94       0.76       69277
     1       0.55       0.11       0.19       42366

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

This Algorithm Achieved an accuracy of approximately 62.87% on the testing data. It means it correctly classified slightly over 62% of the data points. For non-fraudulent transactions (class 0), the precision is 63%. This means that when the model predicts a transaction as non-fraudulent, it is correct about 63% of the time. For fraudulent transactions (class 1), the precision is 55%. This means that when the model predicts a transaction as fraudulent, it is correct about 55% of the time. For non-fraudulent transactions, the recall is 94%. This means that the model correctly identifies about 94% of all actual non-fraudulent transactions. However, for fraudulent transactions, the recall is only 11%. This means that the model captures only about 11% of all actual fraudulent transactions.

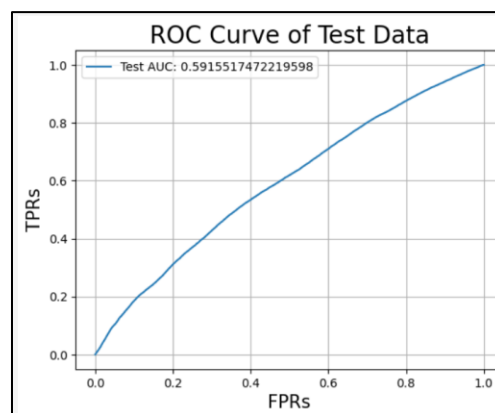
This suggests that the model may need further improvement, particularly in capturing more fraudulent cases.

Confusion Matrix:



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: The Test AUC of 0.59 indicates the performance is moderate.



So, this model's performance is slightly better than random chance and is almost similar to Naïve Bayes.

3. **Decision Tree:** 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.

Below is the code for implementing it:

```
# Using Decision Tree

# Initialize the Decision Tree model
decision_tree_model = DecisionTreeClassifier(random_state=42)

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

# Predict on the testing data
y_pred = decision_tree_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Classification Report:

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

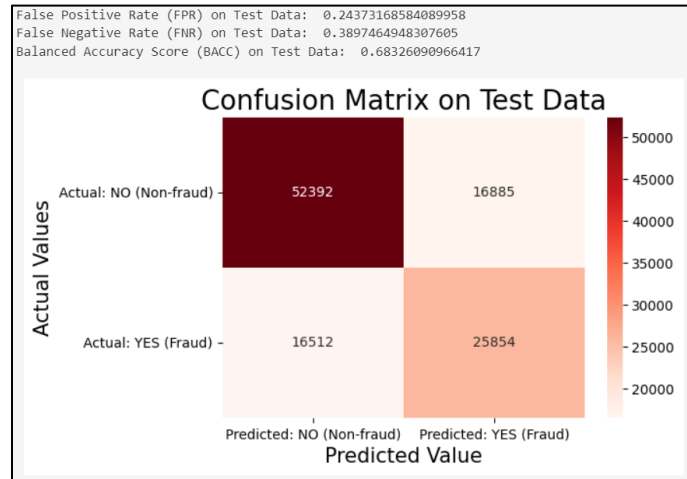
     0       0.76       0.76       0.76     69277
     1       0.60       0.61       0.61     42366

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

The **accuracy** of the model is **70%** which mean the overall prediction is better when compared to the previous model but how reliable it is can be verified from the precision, recall and other parameters.

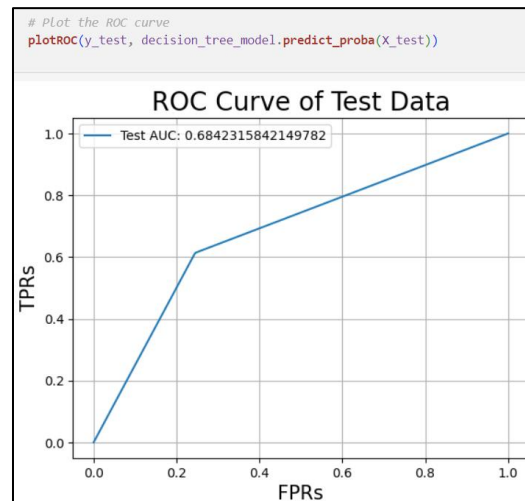
With an accuracy of over 70%, the decision tree model demonstrated its capacity to accurately categorize transactions that were fraudulent as well as those that weren't. Class 0 (non-fraudulent) precision and recall are both approximately 76%, indicating that the model is good at detecting non-fraudulent transactions but marginally less so for fraudulent ones. **Precision** and **recall** for class 1 (fraudulent) are roughly 60% and 61%, respectively, suggesting a moderate ability to detect fraudulent transactions. There is potential for improvement in the decision tree model's overall performance, particularly in accurately identifying fraudulent cases, when it comes to differentiating between fraudulent and non-fraudulent transactions.

Below is the Confusion Matrix:



The model successfully classified 25,854 occurrences as non-fraudulent (True Negatives) and correctly recognized 52,392 instances of fraudulent transactions (True Positives), according to the confusion matrix. False Positives were generated when 16,885 legitimate transactions were mistakenly categorized as fraudulent, and False Negatives were produced when 16,512 fraudulent transactions were overlooked. **There is a need for further refining to improve the model's dependability in detecting fraudulent activity**, even while it performs well in correctly recognizing non-fraudulent transactions. The model struggles with both false positives and false negatives.

ROC Curve:



The decision tree model's AUC (Area Under the Curve) score of 0.684 shows that it performs moderately well at differentiating between fraudulent and non-fraudulent transactions. It **implies that while the model's accuracy in ranking cases outperforms random guesswork**, further development is necessary to increase its prediction capacity.

4. **Random Forest:** For fraud detection, Random Forest is preferred because of its ensemble learning, which fortifies predictions by combining multiple decision trees. It delivers feature 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.

Below is the **code** for the Random Forest implementation:

```
# Using Random Forest

# Initialize the Random Forest model
random_forest_model = RandomForestClassifier(random_state=42)

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

# Predict on the testing data
y_pred = random_forest_model.predict(X_test)

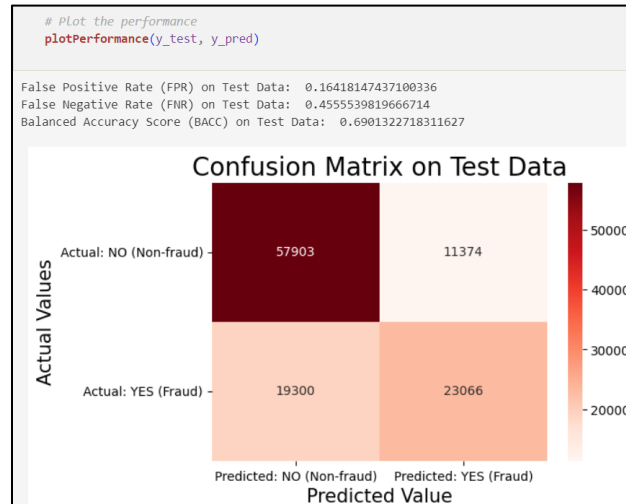
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Classification Report for the same:

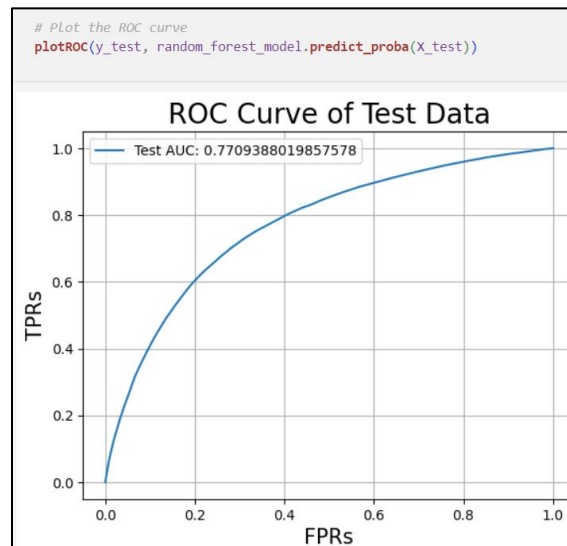
Accuracy: 0.7252492319267666					
Classification Report:					
	precision	recall	f1-score	support	
0	0.75	0.84	0.79	69277	
1	0.67	0.54	0.60	42366	
accuracy			0.73	111643	
macro avg	0.71	0.69	0.70	111643	
weighted avg	0.72	0.73	0.72	111643	

The Random Forest model achieved an accuracy of approximately 72.5%, indicating its ability to correctly classify both fraudulent and non-fraudulent transactions. For non-fraudulent transactions (class 0), the Random Forest model shows good precision (75%) and recall (84%) demonstrating its ability to correctly identify the majority of non-fraudulent events while minimizing false positives. The **precision** (67%) and **recall** (54%) values for fraudulent transactions (class 1), however, indicate that although the model can detect some fraudulent cases, there is still opportunity for development to lower false positives and increase the number of true positives. Overall, **the model's performance highlights how well it minimizes misclassifying cases that aren't fraudulent**, but improvements are still needed to more effectively identify fraudulent activity.

Confusion Matrix:

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



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.

5. **XGBoost:** 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.

Below is the **code**:

```
# Using XGBoost

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

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

# Predict on the testing data
y_pred = xgb_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

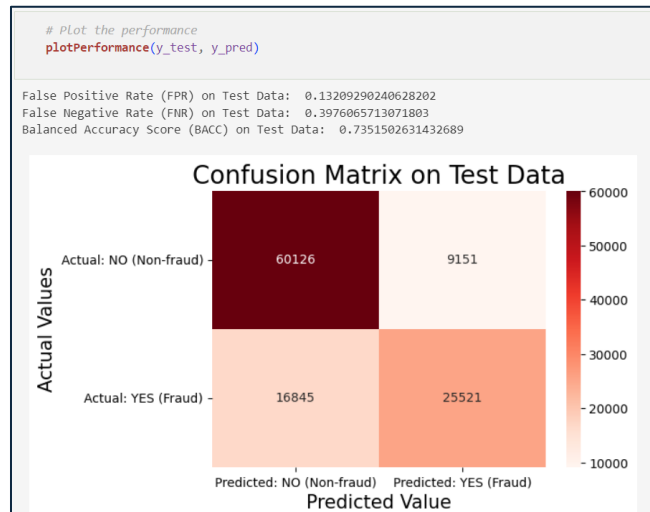
Classification Report:

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	

With an accuracy of 0.77, the XGBoost model demonstrated its ability to accurately categorize examples into the appropriate classes. With a precision of 0.78 for class 0 (non-fraudulent), 78% of all cases predicted as non-fraudulent are indeed non-fraudulent. Class 0 recall is 0.87, which indicates that 87% of all non-fraudulent cases are properly identified by the model. With a precision of 0.74 for class 1 (fraudulent), 74% of all instances predicted to be fraudulent are indeed fraudulent. Class 1 recall is 0.60, which indicates that 60% of all fraudulent cases are accurately identified by the model. For both classes, the F1-score offers a balance between recall and precision. All things considered, the model

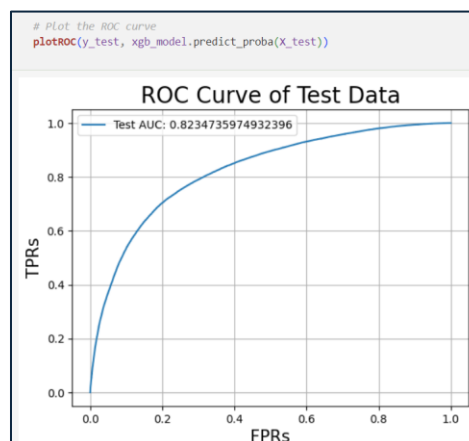
performs well in identifying fraudulent and non-fraudulent cases, with slightly superior results in categorizing non-fraudulent situations.

Confusion Matrix:



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:



With an **AUC of 82.34%**, the model demonstrates good discrimination ability between true positive rates (TPR) and false positive rates (FPR), indicating its effectiveness in

distinguishing between fraudulent and non-fraudulent cases. However, the relatively higher false negative rate suggests that the model may miss some fraudulent cases, potentially leading to overlooked instances of fraud. While the true positive rate indicates the proportion of actual fraudulent cases correctly identified, the false positive rate highlights the instances incorrectly classified as fraudulent. This suggests the model's reliability in correctly identifying fraudulent cases while minimizing false alarms, but it may still miss some fraudulent activities.

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.

- Light GBM:** We are approaching with this model because 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.

Code for implementation:

```
# Using LightGBM

# Initialize the LightGBM model
lgbm_model = LGBMClassifier(random_state=42)

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

# Predict on the testing data
y_pred = lgbm_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Generate a classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

Classification Report:

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

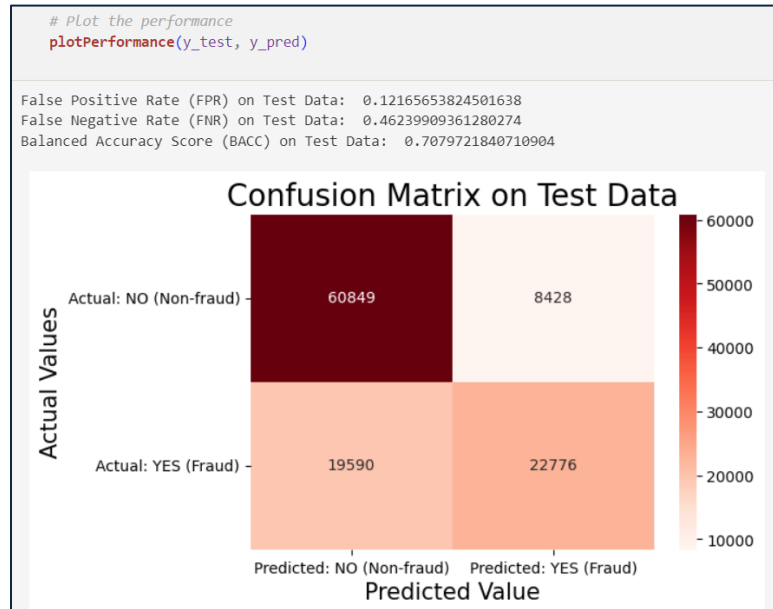
     0       0.76       0.88       0.81       69277
     1       0.73       0.54       0.62       42366

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

When categorized claims as fake or not, the Light GBM model had an accuracy of 74.90%. The accuracy of 73% in identifying fraudulent claims means that 73% of the claims that

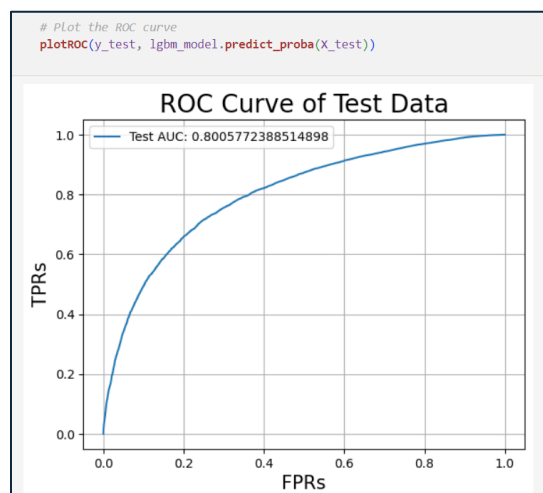
were flagged as fraudulent were actually fraudulent. The fraud detection model's recall, or true positive rate, was 54%, meaning that 54% of all real fraudulent claims were detected by the model. The model performs well overall in terms of recall and precision, which makes it a good option for tasks involving fraud detection.

Confusion Matrix:



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: Below is the ROC Curve of the Test Data for the light Gradient Boosting model



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).

Now, In the case of Fraudulent detection **we aim for a model with high precision to minimize false positives (i.e., identifying non-fraudulent cases as fraudulent)**, while also achieving a high recall to capture as many fraudulent cases as possible. This balance ensures effective fraud identification without overwhelming investigators with false alarms. **We will be picking the best model by comparing all these values.**

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

```
# 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:

```
# We have the highest accuracy with XGBoost 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
```

```
RandomizedSearchCV
  estimator: XGBClassifier
    XGBClassifier
```

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**:

```
> ~
# 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:

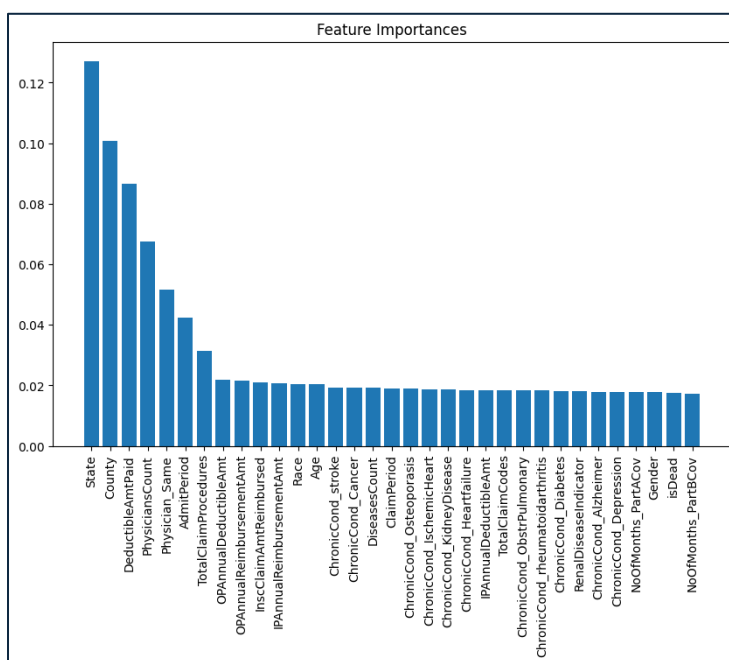
Extracting and visualizing feature importance is crucial for both data scientists and stakeholders. It provides insights into which features have the most significant impact on the model's predictions. For data scientists, understanding feature importance helps in feature selection, model refinement, 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**:

```
# 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.

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 possibly 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.**

Citations:

[1]: Dataset source: <https://www.kaggle.com/datasets/rohitrox/healthcare-provider-fraud-detection-analysis/data>

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. Random Forest: 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