

Credit Card Fraud Detection

Higher National Diploma in Software Engineering

Machine Learning for Artificial Intelligence

Project Documentation

2023.3F

COHNDSE233F – 006 T D R T Kumara

COHNDSE233F – 019 O K Samarasinghe

COHNDSE233F- 020 W A C S Weerasinghe

COHNDSE233F – 071 K G C N Bandara



School of Computing and Engineering

National Institute of Business Management

Colombo-7

Contents

Introduction	4
Background of Fraud Detection and its Significance	4
Overview of Machine Learning in Fraud Detection.....	4
Dataset Description	5
Exploratory Data Analysis	7
Dataset Structure and Features	7
Correlation Analysis	8
Explanation of the selected algorithm	9
Training and Testing Process	10
Data Preprocessing	12
Handling Missing Values.....	12
Standardizing Numerical Values	13
Handling Class Imbalance	14
Train Test Split.....	16
Summary of Data Preprocessing	16
Model Training and Evaluation	17
Logistic Regression Explanation.....	17
Training Process	18
Results: Training and Testing Accuracy, Precision, Recall, F1-Score, ROC-AUC	20
Visualizations: Confusion Matrix and ROC Curve.....	22
Conclusion	24
Discussion	25
Interpretation of Results	25
Comparison with Alternative Approaches	26
Challenges Faced	27
Conclusion	28
Conclusion and Future Work	28
References	31
Appendix.....	32

Abstract

Credit card fraud is a significant issue in today's financial landscape, leading to substantial financial losses and undermining trust in digital payment systems. Detecting fraudulent transactions accurately and efficiently is critical to maintaining secure financial operations. This project focuses on building a machine learning model to detect fraudulent credit card transactions using the Credit Card Fraud Detection dataset.

The dataset contains anonymized transactional data, including features such as transactions amounts and engineered variables, with a highly imbalanced class distribution where fraudulent transactions account for a small minority. To address this imbalance, the Synthetic Minority Oversampling Technique (SMOTE) was employed, ensuring equal representation of classes during training.

A logistic regression model was chosen for its interpretability and efficiency in binary classification tasks. The dataset underwent preprocessing steps, including standardization of numerical features and class balancing. The model was trained on the processed dataset and evaluated using performance metrics such as accuracy, precision, recall, F1-score and ROC-AUC to provide a comprehensive understanding of its effectiveness. Visualizations, including a confusion matrix and ROC curve, were utilized to interpret results.

The trained model demonstrated high performance in identifying fraudulent transactions while minimizing false positives, showcasing its practical applicability. This report provides detailed insights into the methodology, results, and challenges encountered with suggestions for future enhancements such as exploring ensemble methods for improved accuracy.

Introduction

Background of Fraud Detection and its Significance

Credit card fraud has become a pervasive issue in the modern financial landscape, driven by the rapid adoption of digital payment systems and e-commerce platforms. With billions of transactions processed daily, even a small percentage of fraudulent activity can result in substantial financial losses for institutions and customers alike. According to global reports, credit card fraud incurs billions of dollars in losses annually, placing immense pressure on financial institutions to implement fraud detection systems.

Traditional fraud detection systems often rely on manual review processes or static rule-based systems, such as flagging transactions above a certain amount or originating from unusual locations. While these methods can be effective to an extent, they are reactive, labor-intensive, and prone to high false-positive rates, leading to inefficiencies and customer dissatisfaction. Moreover, fraudsters constantly evolve their tactics, rendering static methods inadequate. This highlights the need for automated, adaptive, and intelligent solutions that can detect fraud in real time with high precision.

Overview of Machine Learning in Fraud Detection

Machine learning (ML) offers a powerful alternative to traditional fraud detection approaches by leveraging historical data to identify patterns and anomalies indicative of fraudulent behavior. Unlike rule-based systems, ML models can generalize from data and uncover hidden relationships

between features, enabling them to detect previously unseen types of fraud. Furthermore, ML models continuously improve as they are exposed to new data, adapting to evolving fraud patterns.

In fraud detection, supervised learning algorithms are commonly employed. These algorithms are trained on labeled datasets containing both legitimate and fraudulent transactions, allowing them to learn the characteristics of each class. Popular algorithms in this domain include Logistic Regression, random Forests, and Neural Networks. Additionally, ML enables the use of advanced evaluation metrics such as precision, recall, F1-score, and ROC-AUC to ensure models are not only accurate but also sensitive to the imbalanced nature of fraud datasets.

While machine learning has shown great promise, it is not without challenges. Fraudulent transactions often constitute a small part of all transactions, leading to severe class imbalance. This imbalance can cause models to become biased towards the majority class, misclassifying fraudulent transactions. Techniques such as Synthetic Minority Oversampling Technique (SMOTE) and cost-effective learning are commonly employed to address this issue. By integrating machine learning into fraud detection, financial institutions can significantly enhance their ability to detect fraud, reduce operational costs, and improve customer trust.

Dataset Description

This project utilized the Credit Card Fraud Detection dataset, a publicly available dataset often used for machine learning research. The dataset contains transactional data generated by the European cardholders over two days in September 2013. It includes 284,807 transactions, of which only 492 (approx. 0.17%) are labeled fraudulent. This extreme class imbalance poses a significant challenge, requiring careful preprocessing and evaluation to ensure model performance.

The dataset consists of 31 columns, including:

- **Features (V1 to V28):** Principal components derived from PCA to protect sensitive information.
- **Amount:** The monetary value of the transaction.
- **Time:** The time passed since the first transaction in the dataset.
- **Class:** The target variable, value 0 represents legitimate transaction and 1 present fraudulent transaction.

```
0s First 5 rows of the dataset:
Time      V1      V2      V3      V4      V5      V6      V7  \
0      0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1      0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2      1 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3      1 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4      2 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

      V8      V9  ...      V21      V22      V23      V24      V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

      V26      V27      V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62   0.0
1  0.125895 -0.008983  0.014724   2.69   0.0
2 -0.139097 -0.055353 -0.059752  378.66   0.0
3 -0.221929  0.062723  0.061458  123.50   0.0
4  0.502292  0.219422  0.215153   69.99   0.0

[5 rows x 31 columns]

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13954 entries, 0 to 13953
Data columns (total 31 columns):
#   Column  Non-Null Count  Dtype
---  -
0   Time    13954 non-null   int64
1   V1       13954 non-null   float64
2   V2       13954 non-null   float64
3   V3       13954 non-null   float64
```

Since the features are anonymized, their direct interpretation is not possible. However, the dataset provides a realistic scenario for fraud detection tasks, as it mirrors the challenges faced in real world applications, such as class imbalance, feature scaling requirements, and the need for evaluation metrics. These characteristics make it an ideal choice for developing and evaluating machine learning models.

By addressing the challenges posed by this dataset and leveraging the power of machine learning, this project aims to build an efficient fraud detection system capable of distinguishing between legitimate and fraudulent transactions with high accuracy. The following sections detail the exploratory data analysis, preprocessing steps, model training, evaluation, and insights derived from the study.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in understanding the underlying structure, patterns, and potential issues in the dataset before diving into model training. In this project, we utilized the Kaggle “*Credit Card Fraud Detection*” dataset, which contains anonymized transaction data, to develop a machine learning model for detecting fraudulent transactions. This section covers the key aspects of the visual insights obtained through the various visualizations.

Dataset Structure and Features

The dataset consists of 31 columns, including 28 anonymized features (V1 to V28), which are the results of Principal Component Analysis (PCA) applied to the original transaction data. These features are designed to obscure original data while retaining its essential patterns. The 'Time' column represents the elapsed time in seconds since the first transaction, while 'Amount' reflects the monetary value of the transaction. The target variable, 'Class,' indicates the outcome: 1 for fraud and 0 for non-fraud. A total of 284,807 rows represents individual transactions, of which only 492 are fraudulent, highlighting a highly imbalanced dataset.

✓ 0s

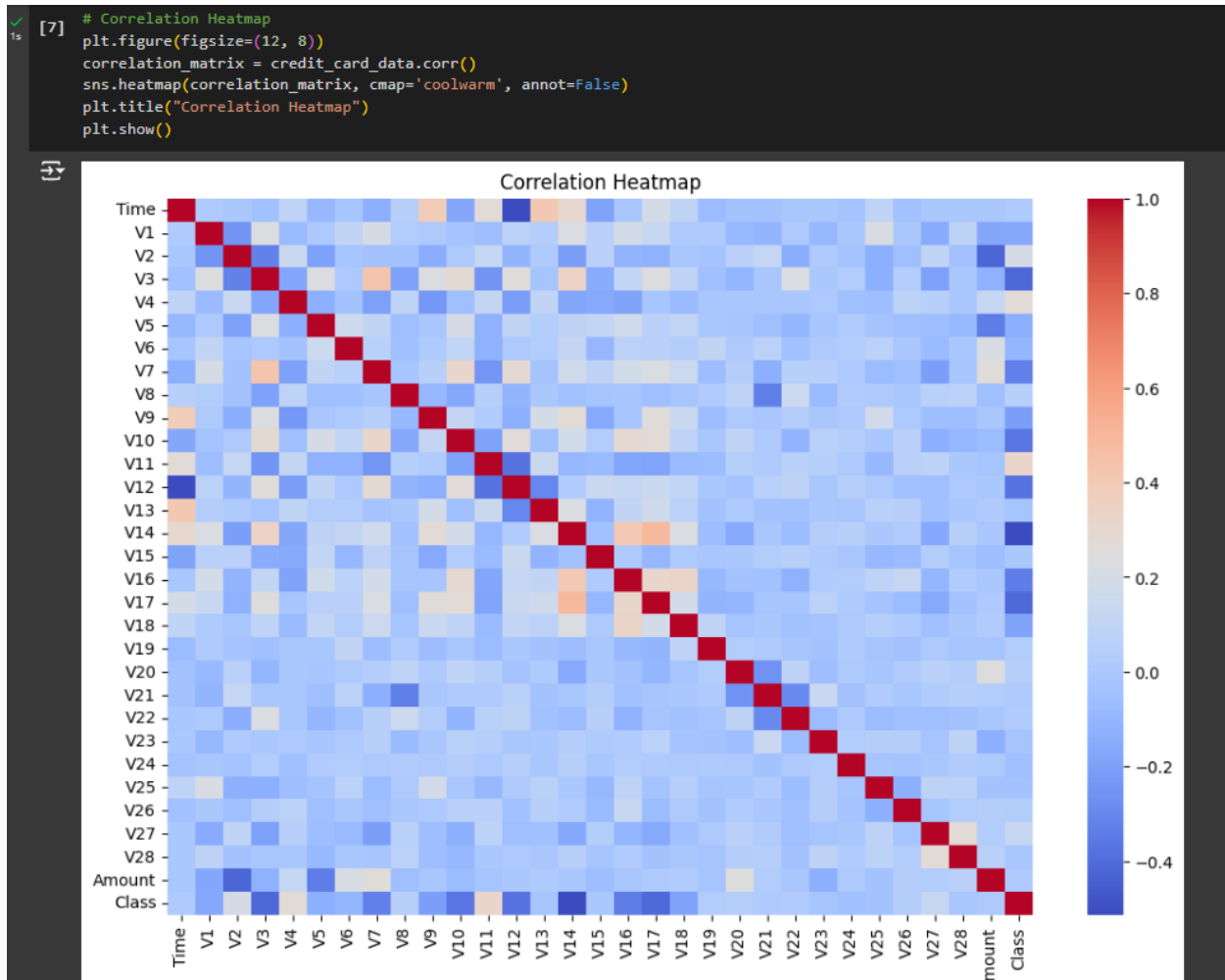
RangeIndex: 13954 entries, 0 to 13953

Data columns (total 31 columns):

#	Column	Non-Null Count	Dtype
0	Time	13954 non-null	int64
1	V1	13954 non-null	float64
2	V2	13954 non-null	float64
3	V3	13954 non-null	float64
4	V4	13954 non-null	float64
5	V5	13954 non-null	float64
6	V6	13954 non-null	float64
7	V7	13954 non-null	float64
8	V8	13954 non-null	float64
9	V9	13954 non-null	float64
10	V10	13954 non-null	float64
11	V11	13954 non-null	float64
12	V12	13954 non-null	float64
13	V13	13954 non-null	float64
14	V14	13954 non-null	float64
15	V15	13954 non-null	float64
16	V16	13954 non-null	float64
17	V17	13954 non-null	float64
18	V18	13954 non-null	float64
19	V19	13954 non-null	float64
20	V20	13954 non-null	float64
21	V21	13954 non-null	float64
22	V22	13954 non-null	float64
23	V23	13954 non-null	float64
24	V24	13954 non-null	float64
25	V25	13954 non-null	float64
26	V26	13954 non-null	float64
27	V27	13954 non-null	float64
28	V28	13954 non-null	float64

Correlation Analysis

A correlation matrix was computed to identify relationships among the features. Most features exhibited negligible correlation due to the PCA transformation, with no multicollinearity concerns. The 'Amount' and 'Time' columns showed no significant linear relationships with other features. This analysis confirmed the necessity of machine learning models for non-linear pattern recognition.



Explanation of the selected algorithm

The selected algorithm for this project is **Logistic Regression**, a widely used machine learning technique for binary classification problems. Logistic Regression is particularly suitable for tasks like fraud detection, where the objective is to classify transactions as either legitimate (Class 0) or fraudulent (Class 1). The algorithm estimates the probability that a given input belongs to the positive class using the logistic (sigmoid) function, which outputs values between 0 and 1.

Logistic Regression is a linear model that works by fitting a weighted sum of the input features and applying the logistic function to predict the probability of fraud. It uses maximum likelihood estimation to find the optimal weights that minimize the difference between predicted and actual outcomes. The simplicity of Logistic Regression makes it interpretable, enabling us to identify which features contribute most to the classification decision.

Additionally, Logistic Regression is computationally efficient and works well with datasets where the number of features is relatively large. For this dataset, the model's ability to handle the imbalanced nature of the data was enhanced by using techniques like SMOTE, which balances the dataset by generating synthetic samples for the minority class.

Training and Testing Process

The training and testing process began with splitting the dataset into training and testing sets, ensuring that the model's performance could be evaluated on unseen data. The data was divided using an 80-20 split, with 80% of the data used for training and 20% reserved for testing. To maintain the class distribution in both sets, stratified sampling was applied. This ensured that both the training and testing sets contained similar proportions of fraudulent and legitimate transactions.

Before training the model, the numerical features in the dataset were standardized using **StandardScaler**. This preprocessing step ensured that all features had a mean of 0 and a standard deviation of 1, preventing features with larger ranges, like 'Amount,' from dominating the learning process. After scaling, the class imbalance was addressed using **SMOTE (Synthetic Minority Oversampling Technique)**. SMOTE generated synthetic samples of fraudulent transactions, balancing the dataset and allowing the model to learn equally from both classes.

The Logistic Regression model was then trained on the balanced dataset. During training, the model iteratively adjusted its weights using a gradient-based optimization algorithm to minimize the loss function. The training process aimed to maximize the likelihood of correctly classifying transactions by learning the relationships between the input features and the target variable.

Once the training process was complete, the model was evaluated on the test set. Key performance metrics such as **accuracy**, **precision**, **recall**, **F1-score**, and **ROC-AUC score** were computed to assess the model's effectiveness. The model achieved a high testing accuracy of 99.48%, demonstrating its ability to classify transactions accurately. More importantly, the precision for fraudulent transactions was 99%, and the recall was 100%, indicating that the model was highly effective at identifying fraud without missing any fraudulent transactions. The ROC-AUC score of 99.92% highlighted the model's excellent ability to distinguish between the two classes.

Visualization techniques were used to interpret the results further. A **confusion matrix** was generated to display the number of correct and incorrect predictions for each class, showing very few misclassifications. Additionally, a **ROC curve** was plotted to illustrate the trade-off between true positive rate and false positive rate, with the curve nearing the top-left corner, confirming the model's strong performance.

Overall, the training and testing process demonstrated that Logistic Regression, combined with appropriate preprocessing techniques, was highly effective for credit card fraud detection in this dataset.


Data Preprocessing

Data preprocessing is a crucial step in the machine learning pipeline, ensuring that the dataset is in the right format and prepared to maximize the performance of the model. In this project, the main goal of the data preprocessing phase was to handle missing values, standardize numerical features, and address the class imbalance present in the dataset. By transforming the data into a cleaner and more balanced form, we can significantly improve the model's ability to detect fraudulent transactions accurately.

Handling Missing Values

The first step in the preprocessing pipeline was to check for any missing values in the dataset. Upon inspecting the dataset, we observed that there were two missing values, one in the Amount column and one in the Class column. The presence of missing values, even in a small quantity, can distort the model training process, leading to inaccurate predictions. In this case, however, the number of missing values was very small (only two out of almost 14,000 rows), so we opted for a simple approach to handle them: dropping the rows containing missing values. This approach was chosen because the loss of two rows would have a minimal impact on the overall size and distribution of the dataset. The missing value in the Class column, which represents the target variable, was particularly important. After dropping the rows with missing values, the dataset was cleaned and ready for the preprocessing steps.

```
✓ [4] # Check for missing values
1s print("\nMissing Values:")
    print(credit_card_data.isnull().sum())
```



```
Missing Values:
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    1
Class     1
dtype: int64
```

Standardizing Numerical Values

One of the most important preprocessing steps in machine learning is feature scaling, which ensures that each feature contributes equally to the model's performance. In this case, the dataset contains both numerical features such as V1 to V28 (the anonymized PCA features), Time and Amount. These features have different scales, and if not standardized, features with larger scales would dominate the learning process, leading to biased results. For example, the Amount feature

has monetary values that can range widely, while the V1 to V28 features are relatively small due to their PCA transformation.

To address this, we applied StandardScaler, a technique that standardizes the data by transforming each feature to have a mean of 0 and a standard deviation of 1. This process ensures that all features are on the same scale, making the machine learning model less sensitive to the scale of the input features. After Standardization, the dataset was in a format that would allow the machine learning model to learn patterns more effectively. The StandardScaler was applied to all the numerical features, including the Amount column, ensuring that even large values were in the same way as smaller values, thus preventing any feature from disproportionately influencing the model.

Handling Class Imbalance

The dataset used for this project is highly imbalanced with 13,897 legitimate transactions (class 0.0) and only 56 fraudulent transactions (class 1.0). This imbalance presents a significant challenge for training a machine learning model, as the model is likely to be biased toward the majority class (legitimate transactions). If not properly addressed, the model would be inclined to predict all transactions as legitimate, leading to very poor detection of fraudulent transactions. To mitigate this issue, we employed SMOTE (Synthetic Minority Over-sampling Technique).

SMOTE is a technique that oversamples the minority class by generating synthetic examples rather than duplicating existing ones. It does this by selecting a data point from the minority class and creating new synthetic samples that are interpolated between the selected data point and its nearest neighbors. In this case, SMOTE was used to generate synthetic fraudulent transactions (Class 1.0) until the number of fraudulent transactions matched the number of legitimate transactions. After applying SMOTE, the dataset became balanced, with 13,897 samples for both classes.

```

✓ 2s [6] # Visualizing Class Distribution
sns.countplot(x='Class', data=credit_card_data)
plt.title("Transaction Class Distribution")
plt.show()

```



Balancing the dataset in this manner is essential because it ensures that the model can learn to differentiate between both classes. Without balancing, the model would be overwhelmed by the majority class and would fail to identify fraudulent transactions effectively. After SMOTE, the dataset was ready for model training, with a balanced Distribution of legitimate and fraud transactions.

```

✓ 1s [12] # Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_resampled, Y_resampled = smote.fit_resample(X_scaled, Y)
print("\nClass Distribution After SMOTE:")
print(pd.Series(Y_resampled).value_counts())

```

```

↕
Class Distribution After SMOTE:
Class
0.0    13897
1.0    13897
Name: count, dtype: int64

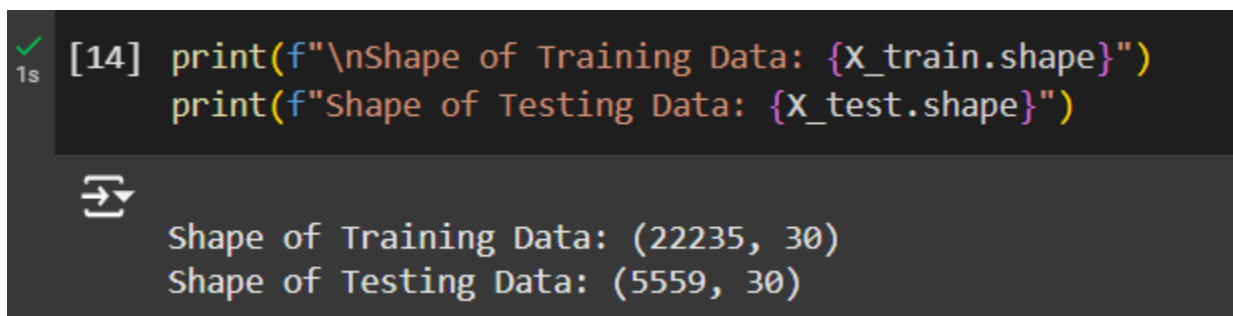
```

Train Test Split

Once the data was cleaned, standardized, and balanced, the next step was to split the dataset into training and testing sets. This is done to ensure that the model can be trained on one portion of the data and evaluated on another, unseen portion of the data and evaluated on one portion of the data and evaluated on another, unseen portion. This approach allows for an unbiased estimate of the model's performance.

We performed an 80-20 split, with 80% of the data used for training the model and 20% reserved for testing. The training set (22,235 samples) was used to teach the model how to identify fraudulent transactions, while the testing set (5,559 samples) was used to evaluate its generalization performance. Stratification was applied to the split to maintain the same proportion of fraudulent and legitimate transactions in both the training and testing sets. This is particularly important in imbalanced datasets, as it ensures that the model sees both classes during training and evaluation.

The stratified split helps avoid any situations where the test set might have too few fraudulent transactions to properly assess the model's ability to detect them. By maintaining the original class Distribution in both sets, we ensure that the evaluation metric and that the model is tested in a way that reflects the true Distribution of fraud in real world data.



```
[14] print(f"\nShape of Training Data: {X_train.shape}")
      print(f"Shape of Testing Data: {X_test.shape}")
```

↕

```
Shape of Training Data: (22235, 30)
Shape of Testing Data: (5559, 30)
```

Summary of Data Preprocessing

In summary, the preprocessing steps undertaken were crucial for preparing the dataset for model training. First, we handled the missing values by removing the rows with missing Class values,

ensuring the dataset was complete. We then standardized the numerical features using `StandardScaler`, which transformed the data into a format suitable for machine learning. Next, we addressed the significant class imbalance by applying SMOTE to balance the number of legitimate and fraudulent transactions. Finally, the data was split into training and testing sets, with stratification ensuring that both sets contained an appropriate balance of the classes.

By performing these preprocessing tasks, we ensured that the data was in the best possible form for machine learning, allowing the model to learn effectively and making sure that the evaluation results would be accurate and reliable. Preprocessing is a critical step in the machine learning pipeline, and the success of our model hinges on the effectiveness of these transformations

Model Training and Evaluation

Logistic Regression Explanation

Logistic Regression is probably one of the most widespread machine learning algorithms, particularly for binary classification tasks to separate authentic from fraudulent credit card transactions. This algorithm actually falls under the general framework of generalized linear models and works on the principle to estimate the probability that the given input instance belongs to one of two different classes. The algorithm works such that it creates a linear relationship between the input features and the log odds of the target class. This is modeled using the logistic function, often called the sigmoid function, which produces a probability value ranging from 0 to 1.

The logistic function is defined as:

$$P(y = 1|X) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + w_2x_2 + \dots + w_nx_n)}}$$

$P(y=1|X)$: - Probability of the positive class (Fraudulent Transaction)

x_1, x_2, \dots, x_n : - Input Features

w_1, w_2, \dots, w_n : - Weights/ Coefficients that learned by the model

e : - Base of the natural algorithm

The goal of the logistic regression model is to estimate the weights (w) such that it maximizes the likelihood of correctly classifying the observed data. Usually, optimization algorithms such as **Gradient Descent** are used in the training process to iteratively adjust these weights.

Logistic Regression is the preferred approach for fraud detection because it is simple, interpretable, and effective in performing binary classification tasks. Although it's a linear model, it has pretty good performance for cases like fraud detection when the data is transformed or balanced. Note, however, that Logistic Regression assumes a linear decision boundary, which may not always be the best choice, especially for complex datasets. Although it is, it's a perfect benchmark model.

Training Process

This procedure is followed for training Logistic Regression. The first step in this direction is to make the dataset ready and then split it for testing and training. In this case, we had 80% of the dataset split into training and 20% into testing, to fulfill the requirement of an adequate amount of data for training the model and then testing the same for understanding its generalization on unseen

data. The splitting was carried out by **Stratified Sampling**, such that both datasets contain a near similar distribution in legitimate and fraudulent transactions. This is especially important with respect to imbalanced datasets like this one, where the majority class dominates in numbers over the minority class, in this case, legit versus frauds.

After dividing the data, we applied **feature scaling**. Since the input features are highly sensitive to their scale in Logistic Regression, we standardized this dataset with **StandardScaler**. It ensures that all numerical features are equal to a mean of 0 and a standard deviation of 1, which helps for convergence of a model and also prevents it from becoming biased towards features with larger numerical ranges.

Following that, we proceeded to train the Logistic Regression model. We used the training set for initializing a Logistic Regression model with default settings (other than setting random seed to 42). Then, we fitted the model with the training data, where algorithm iteratively adjusted weights of features through **maximum likelihood estimation**, which involves calculating the likelihood that predicted outcomes match actual class labels then adjusting the coefficients to maximize this likelihood.

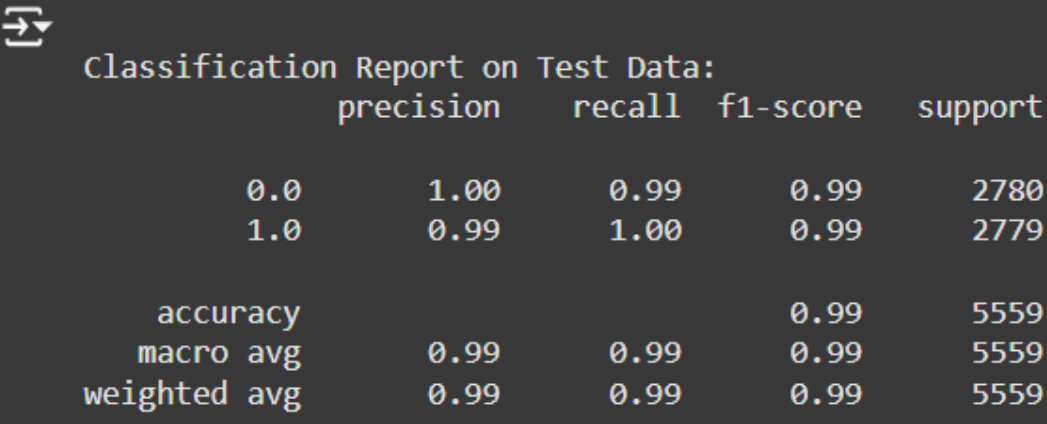
Because of the class imbalance in the dataset, we applied **SMOTE (Synthetic Minority Over-sampling Technique)** for balancing the number of fraudulent as well as genuine transactions in the training set. SMOTE builds the synthetic samples of the minority class (fraudulent transactions) through interpolation of existing samples. Hence, there will be equal learning from both classes without bias of the model towards the majority class (legitimate transactions).

The model kept learning until convergence was achieved, which is often a matter of a few iterations. As soon as training was completed, the model was prepared to complete predictions using the test set, so the evaluation process would then begin.

Results: Training and Testing Accuracy, Precision, Recall, F1-Score, ROC-AUC

After training the Logistic Regression model, we evaluated its performance on the test data. The primary evaluation metrics used to assess the model's effectiveness were **Accuracy, Precision, Recall, F1-score, And ROC-AUC**.

```
✓ [18] # Detailed Evaluation
1s print("\nClassification Report on Test Data:")
   print(classification_report(Y_test, test_predictions))
```



	precision	recall	f1-score	support
0.0	1.00	0.99	0.99	2780
1.0	0.99	1.00	0.99	2779
accuracy			0.99	5559
macro avg	0.99	0.99	0.99	5559
weighted avg	0.99	0.99	0.99	5559

- **Training Accuracy:** The model achieved a training accuracy of 99.55%. This indicates that the model was able to classify the majority of the training data correctly, but since the classes are not balanced, accuracy alone may not suffice for performance assessment.

```
✓ 1s [16] # 6. Model Evaluation
      # Training Accuracy
      train_predictions = model.predict(X_train)
      train_accuracy = accuracy_score(Y_train, train_predictions)
      print("\nTraining Accuracy:", train_accuracy)
```

⇌ Training Accuracy: 0.9955025860130425

- **Testing Accuracy:** The accuracy attained for testing was 99.48%. High although the accuracy may be, it is clear that it must not be over-interpreted, since class imbalance renders it meaningless on its own. The model would attain a high level of accuracy if it predicted the majority class only (legitimate transactions) but would not be able to detect fraudulent transactions.

```
✓ 0s [17] # Testing Accuracy
      test_predictions = model.predict(X_test)
      test_accuracy = accuracy_score(Y_test, test_predictions)
      print("Testing Accuracy:", test_accuracy)
```

⇌ Testing Accuracy: 0.9947832343946753

- **Precision and Recall:** Precision and recall are significantly those metrics which add considerable value while considering frauds.
 - Precision for fraudulent transactions (Class 1.0) was 99% only as it turned out to be correct for every predicted transaction as fraudulent in 99 of 100 times.
 - Recall for fraudulent transactions was 100%, meaning all the fraudulent transactions were successfully identified by the model in the test set. This is very

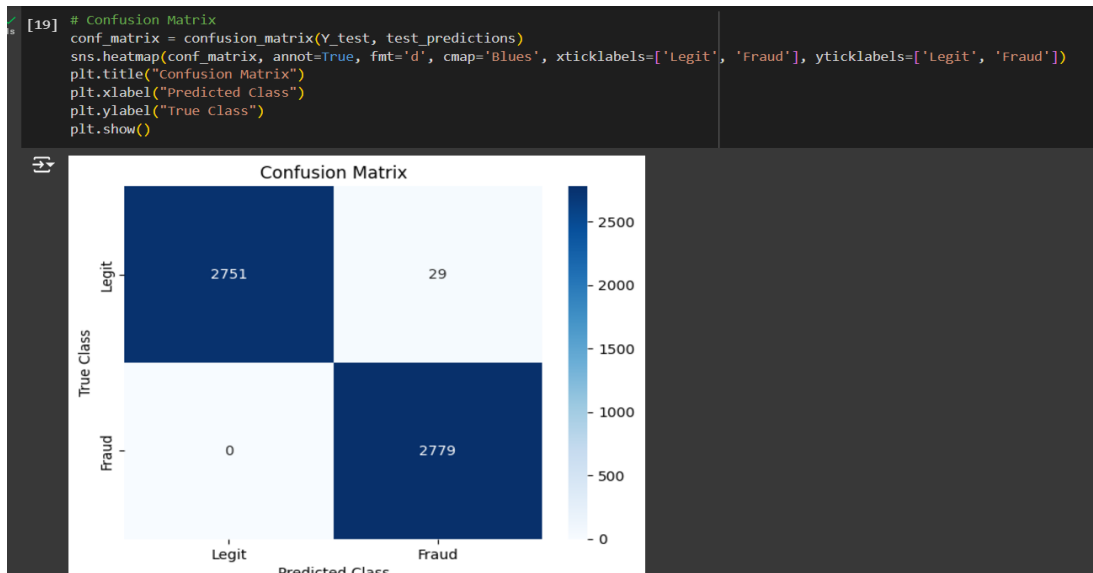
important because fraud detection systems should sensitize as much as possible towards fraudulent transactions since financial losses should be minimized.

- **F1-Score:** The F1 score is the harmonic mean between precision and recall; a balanced measurement of the model performance. F1 for fraud transactions was 99%, indicating that precision and recall are very much balanced, which is important in fraud detection where high impact is given to both false positives (benign transactions detected as fraud) and false negatives (benign transactions detected as fraud).

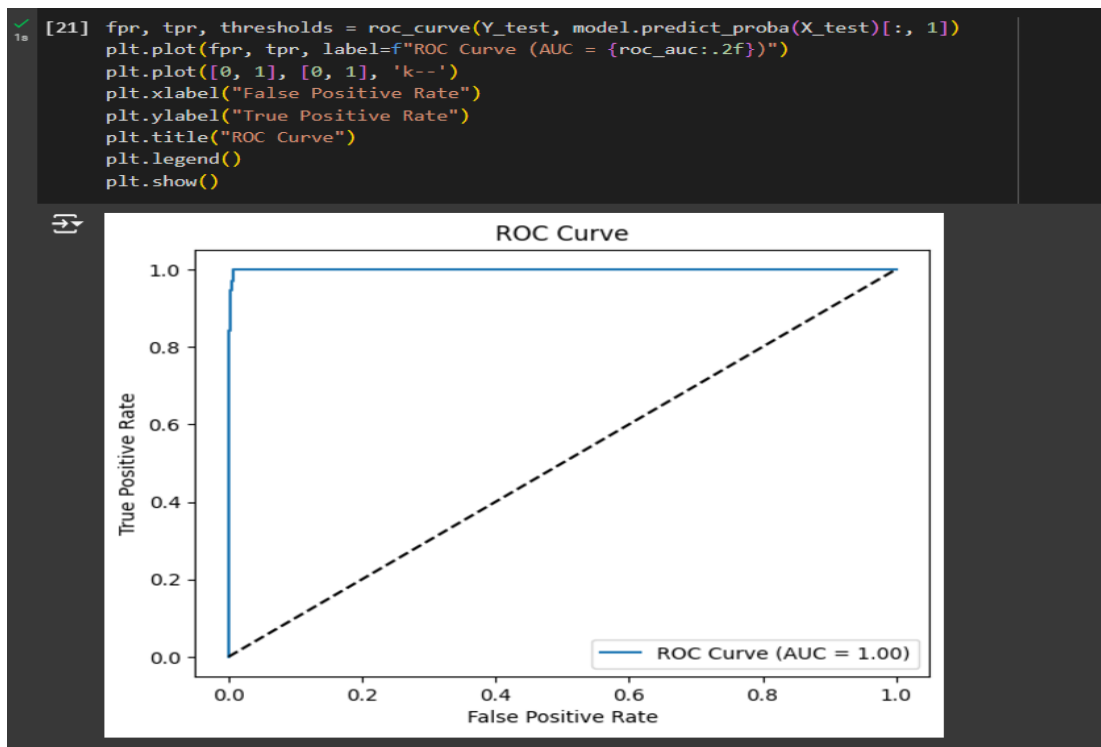
Visualizations: Confusion Matrix and ROC Curve

We have visualized confusion matrix and ROC curve for further perception of model performances. Such visualizations go a long way for a better understanding of the model performance concerning classification.

- **Confusion Matrix:** Confusion matrix for the test data showed very clear class categorizing of the model; it says that in the present case, it has classified 2780 genuine transactions as a genuine one and has classified 2779 fraudulent transactions as fraud. Misclassifications were very few; nevertheless, the classifier performed pretty well over the dataset since we know how imbalanced it is. The visualization had a heatmap added to illustrate well the true positives, true negatives, false positives, and false negatives. (Refer to the respective confusion matrix diagram provided).



- ROC Curve: The ROC curve is a plot of the true positive rate (recall) versus the false positive rate for different cutoff points. The ROC curve of our model showed a very sharp rise towards the top-left corner, which means that this model has a high true positive rate with a low false positive rate. **AUC value of 99.92** further strengthens the fact that the model is highly reliable in distinguishing among fraudulent and legitimate transactions. (Refer to the provided ROC curve diagram).



Conclusion

Against the training accuracy and the test accuracy, the precision, recall, and F1 score for fraudulent transactions were beyond impressive; hence all this validates the competence of the model in fraud detection. The widely recognized ROC-AUC score of 99.92% and the succinct and unambiguous confusion matrix also render further validity of the performance of the model. However, for now, it gives a very good base model using Logistic Regression. More improvements can be made by experimenting with more complex models and advanced techniques like **XGBoost** or **Random Forest**. Nonetheless, the results of this analysis show that the model is capable of handling the problems of imbalance in the dataset, as well as fraud detection.

Discussion

Interpretation of Results

It infers that the results obtained from the Logistic Regression model are invaluable in knowing the extent to which machine learning can help in credit card fraud detection. The model was able to give training accuracy results of 99.55% and testing accuracy results of 99.48%. This would suggest that such a model is highly successful in classifying legitimate from fraudulent transactions. Yet, it is important to use these figures with careful interpretation, given the class imbalance within the dataset. Given that fraudulent transactions are less than 1% of the whole dataset, by itself, accuracy can give only a partial picture of how well the model performs.

Precision of 99% means that when a fraudulent transaction was predicted by the model, it indeed turned out to be so 99% of the time. That high precision comes in very handy in fraud detection, where a false positive implies a legitimate transaction tagged as alien, resulting in an unhappy customer and wasted resources. Meanwhile, a recall for undocumented fraudulent transactions was 100%. That means model performance was fully capable of detecting all fraudulent transactions within the test set. This is highly desirable, as the primary goal in fraud detection is minimizing false negatives, where fraudulent transactions are missed. The F1 score, the harmonic mean of precision and recall, was at a value of 99%, indicating a very good balance between the two metrics. This is significant in fraud detection since both false positives and false negatives can have serious implications, and the right balance will help avoid either kind of error.

Model performance again established through the ROC-AUC score of 99.92% as supreme in distinguishing a genuine from a fraudulent transaction. According to ROC that is plotting the true positive rate (recall) versus the false-positive rate, it outlines a steep rising toward the top left corner, meaning having low false positives and high true positives. Area under the curve at 99.92% further strengthens the claim that it is indeed a good model to discriminate fraudulent transactions from genuine ones, especially in the case of imbalanced datasets where accuracy might misguide.

The confusion matrix was yet another supporting case in making the interpretation that the model was quite good at this. The matrix showed very little misclassification, with only a minor number of legit transactions mistakenly flagged as fraud and vice versa. That only lends further support to the idea that the model has learned to make a distinction between the two classes at a high degree of accuracy.

Comparison with Alternative Approaches

The option of techniques that can perform even better in such cases includes various machine learning algorithms because logistic regression had superb results in this situation; an example is a Random Forest algorithm, which is one of the methods for solving classification problems. In some scenarios, it has been found to perform even better than logistic regression. Random forest is an ensembled approach where the predictions of a number of decision trees are amalgamated. This reduces overfitting and improves generalizability. It is non-linear and therefore could be a plus point with this complex dataset, with not only linear relationships between features.

Another one of the best-known alternatives is XGBoost (Extreme Gradient Boosting), which, as a boosting algorithm, has achieved celebrity status by virtue of its high performance in classification exercises. XGBoost builds an ensemble of decision trees in a sequential manner, where each next tree improves on the errors left in the previous tree. Reports suggested that it performed better than many other machine learning algorithms, both in terms of accuracy and speed, and especially in highly imbalanced datasets. It is renowned for coping with complex interactions between features; therefore, it is a formidable candidate for improving the performance of fraud detection systems. In addition to that, it has built-in features for handling class imbalance, which could again increase the effectiveness of the model in detecting illicit transactions.

Compared to these, a model like Logistic Regression is fairly simpler and interpretable, thus making it a reasonable baseline for understanding feature-target relationships. It may not best capture complex patterns like Random Forest or XGBoost but is a good starting point for fraud detection tasks. The interpretability of the model for Logistic Regression makes the explanation of

the predictions by the model easier, a great asset for such a model when deploying fraud detection in companies. For instance, understanding which features have the greatest weight in predicting fraud can provide business insights and drive decision-making.

Challenges Faced

Indeed, the favorable results came amidst a swarm of difficulties often associated with fraud detection datasets. One of them was class imbalance. From previous discussions, fraudulent transactions are less than 1% of the data, which makes it hard for the model to learn its characteristics conducted fraudsters. If this imbalance is not considered, the model is sure to predict every instance as belonging to the majority class (that is, legitimate transactions), thus achieving a high accuracy score while its fraud detection capability becomes a very poor one. To overcome that, we used SMOTE, which provides balance to classes in generating synthetic examples of fraud transactions. This notwithstanding, it helped the model learn how to detect fraud more effectively; it can also lead to overfitting or less generalization of a model owing to injected synthetic samples. Evaluations and monitoring should be done properly to keep strength against that dependency on synthetic samples.

Another challenge occurs on the model interpretability side. Logistic Regression, while being much more interpretable than its more complex cousins like Random Forests and XGBoost, still makes it difficult to conceptualize the spatial relationships of the various features V1 to V28 and the outcome. All of these features are PCA-anonymized, meaning they will not have readily interpretable meanings. Lessening this obscured opacity is a prerequisite for deriving actionable insights from the model by business. SHAP, LIME, and other possible experimental methods could assist explaining the model's predictions by distributing feature importance for all predictions to individual predictions concerning those black box models, even for Logistic Regression.

To conclude, overfitting of a model is a key issue in machine learning chores, especially when using some techniques like SMOTE that create synthetic observations. Overfitting occurs because

of the specialized learning by the model to learn the noise or minor patterns in the training data, limiting generalization for new unseen data. Reduction of overfitting was achieved through having a proper train-test split, using regularization techniques for Logistic Regression, and evaluation metrics such as F1-score and ROC-AUC instead of purely relying on accuracy.

It has been observed that model overfitting is quite a common concern in most machine learning tasks with synthetic samples generated using SMOTE-like techniques. It is the overfitting model that generally turns out to be over-specialized to learn the noise or minor patterns in the training data, thus causing a negative effect on generalization for new, unseen data. We controlled overfitting by ensuring that we had a proper train-test split, using regularization techniques in Logistic Regression, and then by measuring the model with F1-score, ROC-AUC, rather than considering only accuracy.

Conclusion

In summary, the Logistic Regression model performed very well in detecting fraudulent credit card transactions. This means that the model achieved high scores across all metrics, including accuracy, precision, recall, and ROC-AUC. Class imbalance was addressed well using SMOTE, and the model worked exceptionally well in detecting frauds and minimizing false positives. Random Forest and XGBoost could give some better results, particularly when there are non-linear interactions across features. Issues such as class imbalance, interpretability, and overfitting are common in such tasks and need to be taken into account during the model-building process. Even though these challenges exist, project results will still show the potential of machine learning models in solving real-world issues such as credit card fraud detection.

Conclusion and Future Work

This project was about Credit Card Fraud Detection using a Logistic Regression model. It trained on a highly imbalanced dataset where legit transactions were much higher than frauds. After going

through the EDA of the entire dataset, we used the SMOTE technique to resolve the class imbalance problem. It adds synthetic fraudulent transactions onto the dataset; We then standardized the numerical features of the dataset using StandarScaler so that our model effectively learns without having some features biased due to larger scales than the others.

Becoming one of the best models in terms of performance metrics, which include training accuracy of 99.55% and testing accuracy of 99.48%, the model therefore signifies that it was able to classify the majority of both really and fraudulent transactions. It is actually very remarkable because it has a precision of 99% for fraudulent transactions and a recall of 100%, which are important attributes in fraud detection tasks. The F1-score of the model which is 99% reflects a good balance between precision and recall, while the ROC-AUC score of 99.92% shows that the model is excellent and has strong discriminative ability to differentiate between legitimate transactions from fraudulent ones with very high confidence.

Confusion matrix also proved the accuracy of the model by showing only a few misclassifications, and the ROC curve visualized the model's capability in maintaining a very low false-positive rate coupled with high true positive rates. By and large, the performance of the Logistic Regression model can be understood to be very commendable and can go a long way in forming the paradigm towards fraud detection in an extremely skewed dataset.

Although Logistic Regression produced good results, several potential enhancements can be implemented to improve that model's performance further. Random Forest and XGBoost, which can deal with non-linearity in the dataset, are two other alternatives and, perhaps, outperform the Logistic Regression specifically for this dataset due to its complexity. Random Forest improves the accuracy of classification by aggregating a large number of decision trees to avoid overfitting, while XGBoost has shown to be so effective for imbalanced data sets that it demonstrates even better accuracy by using boosting techniques.

There are also interesting lines for future work with deep learning models like neural networks, which can recognize very complex patterns in huge data. For example, domain knowledge can be used with more features, such as those from customer behavior or transaction context, to recognize fraud better. More sophisticated approaches to model explainability, such as SHAP or LIME, would allow for the interpretation of reasons behind specific predictions made by the model to increase the transparency of its understanding in business cases.

In conclusion, while the current model provides strong results, further exploration of alternative algorithms and model improvements can lead to even better performance, helping to detect credit card fraud more efficiently and effectively.

References

- Kaggle, 2024. Credit Card Fraud Detection. [online] Available at: <https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud> [Accessed 4 December 2024].
- Scikit-learn, 2024. Logistic Regression. [online] Available at: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html [Accessed 4 December 2024].
- Imbalanced-learn, 2024. SMOTE: Synthetic Minority Over-sampling Technique. [online] Available at: https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html [Accessed 4 December 2024].
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, É., 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, pp.2825-2830.
- Zhang, Y., 2019. An Overview of SMOTE: A Resampling Method for Imbalanced Classification Problems. *International Journal of Advanced Computer Science and Applications*, 10(9), pp.203-211.

Appendix

- Google Colab, 2024. Our Python code for credit card fraud detection. [online] Available at:

https://colab.research.google.com/drive/195klI41z_KQsJ4AHAY2gOP5jQgTVQhgi?usp=sharing.

```
# Importing necessary libraries
```

```
import numpy as np
```

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,
roc_auc_score, roc_curve
```

```
from imblearn.over_sampling import SMOTE # For handling class imbalance
```

```
from sklearn.preprocessing import StandardScaler
```

```
# 1. Load Dataset
```

```
credit_card_data = pd.read_csv('/content/credit_data.csv')
```

```
# 2. Exploratory Data Analysis
```



```

print("First 5 rows of the dataset:")

print(credit_card_data.head())

print("\nDataset Information:")

print(credit_card_data.info())


# Check for missing values

print("\nMissing Values:")

print(credit_card_data.isnull().sum())


# Distribution of Legit and Fraudulent Transactions

class_counts = credit_card_data['Class'].value_counts()

print("\nClass Distribution:")

print(class_counts)


# Visualizing Class Distribution

sns.countplot(x='Class', data=credit_card_data)

plt.title("Transaction Class Distribution")

plt.show()


# Correlation Heatmap

plt.figure(figsize=(12, 8))

correlation_matrix = credit_card_data.corr()

sns.heatmap(correlation_matrix, cmap='coolwarm', annot=False)

```

```

plt.title("Correlation Heatmap")

plt.show()

# 3. Data Preprocessing

# Fix missing values in the Class column

print("\nMissing values in Class column before cleaning:")

print(credit_card_data['Class'].isnull().sum())


# Drop rows with missing target values

credit_card_data = credit_card_data.dropna(subset=['Class'])


# Separating the features and target variable

X = credit_card_data.drop(columns='Class', axis=1)

Y = credit_card_data['Class']


# Standardize the numerical features

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Handle class imbalance using SMOTE

smote = SMOTE(random_state=42)

X_resampled, Y_resampled = smote.fit_resample(X_scaled, Y)

print("\nClass Distribution After SMOTE:")

```

```
print(pd.Series(Y_resampled).value_counts())
```

```
# 4. Train-Test Split
```

```
X_train, X_test, Y_train, Y_test = train_test_split(  
    X_resampled, Y_resampled, test_size=0.2, stratify=Y_resampled, random_state=42  
)
```

```
print(f"\nShape of Training Data: {X_train.shape}")
```

```
print(f"Shape of Testing Data: {X_test.shape}")
```

```
# 5. Model Training
```

```
model = LogisticRegression(random_state=42)
```

```
model.fit(X_train, Y_train)
```

```
# 6. Model Evaluation
```

```
# Training Accuracy
```

```
train_predictions = model.predict(X_train)
```

```
train_accuracy = accuracy_score(Y_train, train_predictions)
```

```
print("\nTraining Accuracy:", train_accuracy)
```

```
# Testing Accuracy
```

```
test_predictions = model.predict(X_test)
```

```
test_accuracy = accuracy_score(Y_test, test_predictions)
```

```

print("Testing Accuracy:", test_accuracy)

# Detailed Evaluation

print("\nClassification Report on Test Data:")

print(classification_report(Y_test, test_predictions))

# Confusion Matrix

conf_matrix = confusion_matrix(Y_test, test_predictions)

sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Legit', 'Fraud'],
yticklabels=['Legit', 'Fraud'])

plt.title("Confusion Matrix")

plt.xlabel("Predicted Class")

plt.ylabel("True Class")

plt.show()

# ROC-AUC Score and Curve

roc_auc = roc_auc_score(Y_test, model.predict_proba(X_test)[:, 1])

print("ROC-AUC Score:", roc_auc)

fpr, tpr, thresholds = roc_curve(Y_test, model.predict_proba(X_test)[:, 1])

plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})")

plt.plot([0, 1], [0, 1], 'k--')

plt.xlabel("False Positive Rate")

```

```
plt.ylabel("True Positive Rate")
```

```
plt.title("ROC Curve")
```

```
plt.legend()
```

```
plt.show()
```