TABLE OF CONTENTS

1. INTRODUCTION

1.1 Overview

Credit card fraud has become a major concern in the financial world, with a growing number of online transactions leading to increased vulnerability. Detecting fraudulent activities is essential to prevent financial losses and maintain customer trust. However, due to the massive volume of transactions and the subtle nature of fraud patterns, traditional detection methods often fall short.

To address this, machine learning has emerged as a powerful tool, capable of learning patterns in transaction data and identifying anomalies that may indicate fraud. One of the major challenges in building a fraud detection system is dealing with highly imbalanced data—fraudulent transactions represent only a tiny fraction of all transactions.

This project utilizes a real-world credit card dataset to build a machine learning-based fraud detection system. Various sampling techniques, such as Random Oversampling, Random Undersampling, and SMOTE, are applied to handle the imbalance in the dataset. The processed data is then used to train models that can classify transactions as either legitimate or fraudulent. Performance is evaluated using common metrics, and results are compared to understand the impact of data balancing techniques on fraud detection effectiveness.

1.2 Problem Statement

With the growing reliance on digital payments, credit card fraud has become a critical concern for financial institutions worldwide. Detecting fraudulent transactions is a complex task due to the subtle and deceptive nature of fraudulent behaviour. The problem is further intensified by the severe class imbalance, where fraudulent transactions make up only a small portion of the overall data. This imbalance leads to challenges in training machine learning models, as they tend to favour the majority class, resulting in poor detection of actual fraud. There is a need for a reliable and efficient fraud detection system that can accurately identify fraudulent transactions while minimizing false positives, ensuring security without compromising the user experience.

1.3 Motivation

With the rapid growth of e-commerce and digital banking, credit card usage has become a routine part of everyday transactions. However, this increase in usage has also led to a surge in fraudulent activities, posing significant threats to both financial institutions and consumers. Detecting fraud in real-time is a complex task, as fraudsters constantly adapt their methods to bypass security systems. Manual review of transactions is not only time-consuming but also impractical given the enormous volume of data generated daily. Traditional rule-based systems often fail to catch subtle or evolving fraud patterns, leading to the need for more adaptive and intelligent solutions.

This project is driven by the need to develop a robust and scalable fraud detection system that can automatically learn and identify suspicious behavior. By utilizing machine learning techniques and addressing key issues such as data imbalance through resampling methods, the project seeks to improve detection accuracy while minimizing false positives. The ultimate motivation is to contribute toward safer financial systems and protect individuals and organizations from the adverse effects of fraud.

1.4 Challenges

Building an effective credit card fraud detection system presents several critical challenges. The foremost issue is the extreme class imbalance, where fraudulent transactions represent only a tiny fraction of the total dataset. This imbalance makes it difficult for models to accurately learn the patterns associated with fraudulent behaviour, often leading to biased predictions in favour of the majority class. Another major challenge stems from the anonymized nature of the dataset, which restricts interpretability and hinders the ability to understand the real-world significance of the features.

1.5 Applications

- Banking and financial institutions to detect and prevent fraudulent transactions.
- E-commerce platforms to ensure secure online payments.
- Mobile payment apps to monitor and block suspicious activity.
- Retail systems to safeguard point-of-sale transactions.

2. LITERATURE REVIEW

AlsharifHasan Mohamad Aburbeian and Huthaifa I. Ashqar (2022) – "Credit Card Fraud Detection Using Enhanced Random Forest Classifier for Imbalanced Data" This study tackled the problem of class imbalance in fraud detection by combining Random Forest with the SMOTE technique. Their approach improved detection accuracy, achieving 98%, and demonstrated robustness in identifying fraudulent patterns from heavily skewed data distributions.

Mohammad Saeed and Reza Behravesh (2021) – "A Novel Ensemble Learning Method for Credit Card Fraud Detection" The authors introduced an ensemble model combining several classifiers to enhance detection rates. The ensemble outperformed individual classifiers, indicating that hybrid approaches could better capture hidden fraud patterns in transaction datasets.

Ravi Kumar and Priyanka Singh (2020) – "Credit Card Fraud Detection Using Machine Learning Algorithms" This research compared multiple machine learning models, including Decision Trees, SVM, and Random Forest, for credit card fraud detection. They found Random Forest to be the most effective, particularly when paired with proper data preprocessing and feature engineering.

N. Kumar et al. (2023) – "Effective Credit Card Fraud Detection Using SMOTE and XGBoost Algorithm" The paper proposed the use of XGBoost in combination with SMOTE to handle data imbalance. Their method improved precision and recall significantly, especially for detecting minority class fraud transactions.

P. Choudhury and S. Agarwal (2022) – "Credit Card Fraud Detection Using Hybrid Machine Learning Approach" This study explored hybrid models combining logistic regression and tree-based algorithms. The hybrid approach led to balanced sensitivity and specificity, suitable for real-time fraud detection applications.

3. METHODOLOGY

3.1 Overview

This project follows a systematic methodology beginning with data collection from a publicly available Kaggle dataset containing credit card transactions. The collected data is prepared and pre-processed to handle missing values and class imbalance. Machine learning models such as Random Forest Classifier and XGBoost are then applied to train and evaluate the system for effective fraud detection.

3.1.1 Data Collection

Data used in this project is a set of product reviews collected from credit card transactions records. This step is concerned with selecting the subset of all available data that you will be working with. ML problems start with data preferably, lots of data (examples or observations) for which you already know the target answer. Data for which you already know the target answer is called labelled data.

3.1.2 Data Preparation

In this step, the dataset is examined for missing values, null values, and irrelevant features. The "Time" column is dropped for better model performance, and the class distribution is analysed to understand the extent of imbalance between fraudulent and non-fraudulent transactions. Appropriate resampling techniques such as Random Oversampling, Undersampling, and SMOTE are applied to prepare a balanced dataset for training.

3.1.3 Data Pre-Processing

Pre-processing is the process of three important and common steps as follows:

- **Formatting**: It is the process of putting the data in a legitimate way that it would be suitable to work with. Format of the data files should be formatted according to the need. Most recommended format is .csv files.
- **Cleaning**: Data cleaning is a very important procedure in the path of data science as it constitutes the major part of the work. It includes removing missing data and complexity with naming category and so on. For most of the data scientists, Data Cleaning continues of 80% of work.
- **Sampling**: This is the technique of analyzing the subsets from whole large datasets, which could provide a better result and help in understanding the behaviour and pattern of data in an integrated way.

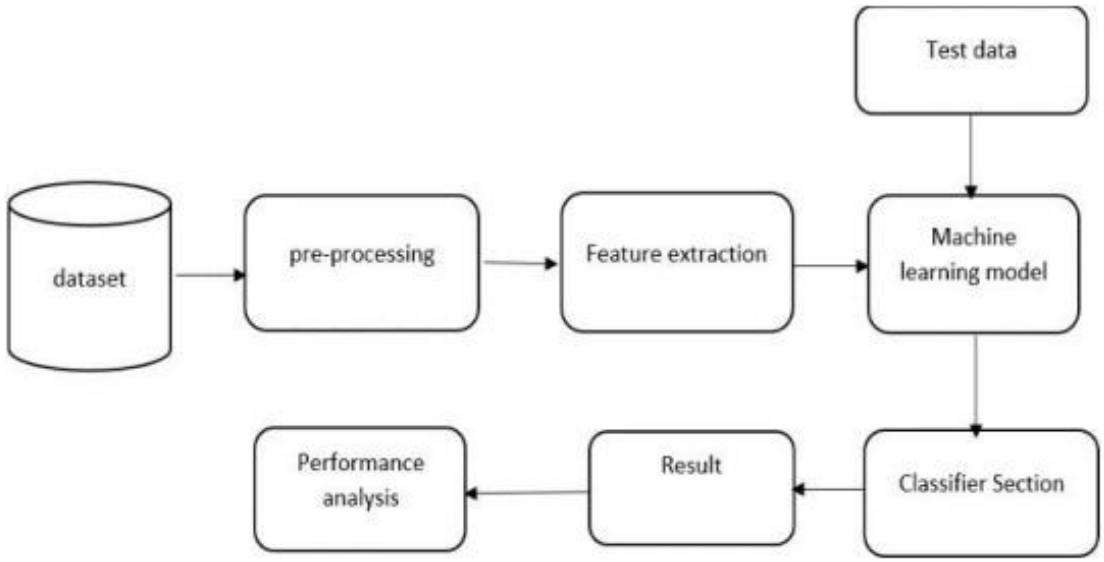
3.1.4 Random Forest Classifier

Random forest is a supervised machine learning algorithm based on ensemble learning. Ensemble learning is an algorithm where the predictions are derived by assembling or bagging different models or similar model multiple times. The random forest algorithm works in a similar way and uses multiple algorithm i.e. multiple decision trees, resulting in a forest of trees, hence the name "Random Forest". The random forest algorithm can be used for both regression and classification tasks.

3.1.5 XGBoost Classifier

XGBoost (Extreme Gradient Boosting) is a supervised machine learning algorithm based on the gradient boosting framework. Gradient boosting is an ensemble technique where multiple weak learners, typically decision trees, are combined to form a strong predictive model. XGBoost builds decision trees sequentially, where each tree tries to correct the errors made by the previous one. The "extreme" in XGBoost refers to its optimized performance, which is achieved through advanced regularization techniques, handling of missing values, and parallel processing. XGBoost can be used for both classification and regression tasks and is known for its high efficiency, scalability, and accuracy, making it a popular choice for many machine learning competitions.

3.2 Architecture of Proposed System



4. SYSTEM REQUIREMENTS

4.1 Software Requirements

- Python
- Anaconda

4.2 Hardware Requirements

- OS – Windows 7, 8, 10, 11 (32 and 64 bit)
- RAM – 4GB

5. CODING

Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Data Loading

```
main_df = pd.read_csv("creditcard.csv")
main_df.head()
```

Data Pre-processing

```
main_df.isna().sum()
main_df.describe()
x = main_df.drop('Class', axis=1)
y = main_df['Class']
x.shape, y.shape
```

Exploratoty Data Analysis(EDA) & Visualization.

```
class_count_df =
pd.DataFrame(main_df['Class'].value_counts().rename_axis('Class').reset_index(name='Coun
ts'))
class_count_df['Class'] = class_count_df['Class'].replace({0: 'Normal', 1: 'Fraud'})
class_count_df.head()

# Create Figure
fig, ax = plt.subplots(figsize=(8, 6))
# Create Barplot
ax = sns.barplot(x=class_count_df['Class'], y=class_count_df['Counts'])
# Show values at the top of each bar
ax.bar_label(ax.containers[0])
# Set labels and title
ax.set_xlabel('Type of Transactions', fontsize=13, fontweight='bold')
ax.set_ylabel('Frequency', fontsize=13, fontweight='bold')
ax.set_title('Count Values of Normal vs Fraud Class', fontsize=16, fontweight='bold')
plt.show()

fraud.Amount.describe()
normal.Amount.describe()

fig, (ax0, ax1) = plt.subplots(nrows=2,ncols=1,sharex=True)
fig.suptitle("Variation of Amount per Class",fontweight='bold')
bins=50
ax0.hist(fraud['Amount'], bins=bins)
ax0.set_title('Fraud')
```

```
ax0.set_ylim(0, 100)
ax0.set_ylabel('No. of Transactions')
ax1.hist(normal['Amount'], bins=bins)
ax1.set_title('Normal')
ax1.set_ylabel('No. of Transactions')
plt.xlim(0, 20000)
plt.xlabel('Amount ($)')
plt.yscale('log')
```

Oversampling

```
from sklearn.utils import resample
#create two different dataframe of majority and minority class
df_majority = main_df[(main_df['Class']==0)]
df_minority = main_df[(main_df['Class']==1)]
# upsample minority class
df_minority_oversampled = resample(df_minority,
                                   replace=True,
                                   n_samples=284315,
                                   random_state=42)
# Combine majority class with upsampled minority class
df_oversampled = pd.concat([df_minority_oversampled, df_majority])
df_oversampled.Class.value_counts()
x_oversampled = df_oversampled.drop('Class', axis=1)
y_oversampled = df_oversampled['Class']
x_oversampled.shape, y_oversampled.shape
```

Undersampling

```
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter
rus = RandomUnderSampler(random_state=42)
x_undersampled, y_undersampled = rus.fit_resample(x, y)
print(f"The number of Classes before the fit {Counter(y)}")
print(f"The number of Classes after the fit {Counter(y_undersampled)}")
from imblearn.over_sampling import SMOTE
```

SMOTE(Synthetic Minority Oversampling Technique)

```
from imblearn.over_sampling import SMOTE
# Resampling the minority class. The strategy can be changed as required.
sm = SMOTE(sampling_strategy='minority', random_state=42)
# Fit the model to generate the data.
x_smote, y_smote = sm.fit_resample(main_df.drop('Class', axis=1), main_df['Class'])
smote_df = pd.concat([pd.DataFrame(x_smote), pd.DataFrame(y_smote)], axis=1)
```

Function XGBoost Classifier for Model fitting, Model evaluation and Visualization.

```
from xgboost import XGBClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
def XGB_model(x,y):
    print("Spliting Datasets....")
    from sklearn.model_selection import train_test_split
    np.random.seed(42)
    x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.2)
    print("Successfully splitted!!!")

    print("Model Fitting.....")
    xgb = XGBClassifier()
    xgb.fit(x_train, y_train)
    print("Successfully model fitted!!!\n")
    print("------------Training Prediction--------------")
    y_preds = xgb.predict(x_train)
    print(f"Classfifcation Report:\n\n{classification_report(y_train, y_preds)}\n\n")
    cf_matrix = confusion_matrix(y_train, y_preds)
    fig, ax = plt.subplots(figsize=(6,4))
    sns.heatmap(cf_matrix, annot=True, cmap='coolwarm', fmt='g')
    fig.suptitle(t="Confusion Matrix",color="orange",fontsize=16);
    ax.set(xlabel="Predicted Label",ylabel="Actual Label");
    train_acc = accuracy_score(y_train, y_preds)
    print(f"Accuracy Score (Train): {train_acc * 100:2f}%\n")
    print("------------Test Prediction--------------")
    y_preds = xgb.predict(x_test)
    print(f"Classfifcation Report:\n\n{classification_report(y_test, y_preds)}\n\n")
    cf_matrix = confusion_matrix(y_test, y_preds)
    fig, ax = plt.subplots(figsize=(6,4))
    sns.heatmap(cf_matrix, annot=True, cmap='coolwarm', fmt='g')
    fig.suptitle(t="Confusion Matrix",color="orange",fontsize=16);
    ax.set(xlabel="Predicted Label",ylabel="Actual Label");
    test_acc = accuracy_score(y_test, y_preds)
    print(f"Accuracy Score (Test): {test_acc * 100:2f}%\n")
    return test_acc
```

XGBClassifier on Normal Datasets,Oversampling,Undersampling,SMOTE datasets

```
acc_normal_test=XGB_model(main.x, main.y)
acc_under_test=XGB_model(main.x_undersampled, main.y_undersampled)
acc_over_test=XGB_model(main.x_oversampled, main.y_oversampled)
acc_smote_test=XGB_model(main.x_smote, main.y_smote)
```

Function Random Forest Classifier for Model fitting, Model evaluation and Visualization.

```
from sklearn.ensemble import RandomForestClassifier
def RF_model(x,y):
    print("Spliting Datasets....")
    from sklearn.model_selection import train_test_split
    np.random.seed(42)
    x_train, x_test, y_train, y_test = train_test_split(x, y,test_size=0.2)
    print("Successfully splitted!!!")
    print("Model Fitting.....")
    rf = RandomForestClassifier(n_estimators=100,      # Fewer trees (reduces training time)
```

```
    max_depth=10,          # Limits tree depth (prevents overfitting & speeds up)
    n_jobs=-1,             # Uses all available CPU cores
    random_state=42,       # Ensures consistent results
    max_features='sqrt')

rf.fit(x_train, y_train)
print("Successfully model fitted!!!\n")
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print("------------Training Prediction--------------")
y_preds = rf.predict(x_train)
print(f"Classfifcation Report:\n\n{classification_report(y_train, y_preds)}\n\n")
cf_matrix = confusion_matrix(y_train, y_preds)
fig, ax = plt.subplots(figsize=(6,4))
sns.heatmap(cf_matrix, annot=True, cmap='coolwarm', fmt='g')
fig.suptitle(t="Confusion Matrix of Training Datasets",color="orange",fontsize=16);
ax.set(xlabel="Predicted Label",ylabel="Actual Label");
train_acc = accuracy_score(y_train, y_preds)
print(f"Accuracy Score (Train): {train_acc * 100:2f}%\n")
print("------------Test Prediction--------------")
y_preds = rf.predict(x_test)
print(f"Classfifcation Report:\n\n{classification_report(y_test, y_preds)}\n\n")
cf_matrix = confusion_matrix(y_test, y_preds)
fig, ax = plt.subplots(figsize=(6,4))
sns.heatmap(cf_matrix, annot=True, cmap='coolwarm', fmt='g')
fig.suptitle(t="Confusion Matrix of Testing Datasets",color="orange",fontsize=16);
ax.set(xlabel="Predicted Label",ylabel="Actual Label");
test_acc = accuracy_score(y_test, y_preds)
print(f"Accuracy Score (Test): {test_acc * 100:2f}%\n")
return test_acc
```

Random Forest Classifier on Normal Datasets,Oversampling,Undersampling,SMOTE datasets

```
acc_normal_test=RF_model(main.x, main.y)
acc_under_test=RF_model(main.x_undersampled, main.y_undersampled)
acc_over_test=RF_model(main.x_oversampled, main.y_oversampled)
acc_smote_test=RF_model(main.x_smote, main.y_smote)
```

Test Accuracy Comparison: XGBoost vs Random Forest

```
import matplotlib.pyplot as plt
# Replace these with your actual test accuracy values
xgb_test = [99.95, 99.99, 91.88, 99.98]    # Normal, Oversample, Undersample, SMOTE
rf_test = [99.96, 99.29, 91.88, 98.76]
labels = ['Normal', 'Oversampled', 'Undersampled', 'SMOTE']
x = range(len(labels))
bar_width = 0.35
plt.figure(figsize=(10, 5))
```

```
# Plotting only test accuracies
xgb_bars = plt.bar([i - bar_width/2 for i in x], xgb_test, width=bar_width, label='XGBoost
Test', color='deepskyblue')
rf_bars = plt.bar([i + bar_width/2 for i in x], rf_test, width=bar_width, label='Random Forest
Test', color='darkorange')
# Annotate bars
for bar in xgb_bars + rf_bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, height + 0.1, f'{height:.2f}%',
             ha='center', va='bottom', fontsize=9)
# Axis & Labels
plt.xticks(x, labels)
plt.xlabel('Sampling Technique')
plt.ylabel('Test Accuracy (%)')
plt.title('Test Accuracy Comparison: XGBoost vs Random Forest')
plt.ylim(90, 101)
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

Save the Model

```
xgb1 = XGBClassifier()
xgb1.fit(main.x_oversampled, main.y_oversampled)
import joblib
joblib.dump(xgb1,'Credit_Card_Model.pkl')
```

Credit Card Fraud Detection UI (Streamlit)

```
import streamlit as st
import joblib
import numpy as np

# Load your model
model = joblib.load("Credit_Card_Model")

st.title("Credit Card Fraud Detection")

# Create input fields for 29 features
inputs = []
for i in range(29):
    value = st.number_input(f"Feature {i+1}", value=0.0)
    inputs.append(value)

if st.button("Predict"):
    prediction = model.predict([inputs])
    if prediction[0] == 0:
        st.success("Normal Transaction")
    else:
        st.error("Fraud Transaction")
```

6. EXPERIMENTS AND RESULT

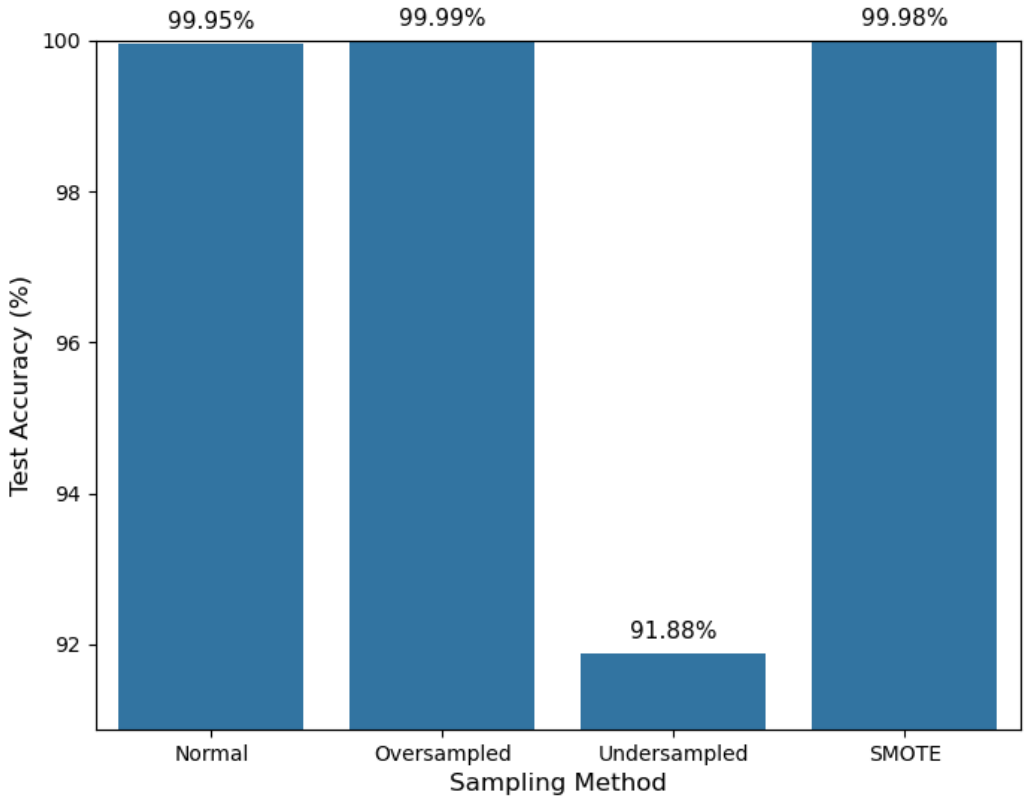
In this project, various experiments were conducted to evaluate the performance of machine learning models on the imbalanced credit card fraud dataset. The dataset was processed using different resampling techniques—Random Oversampling, Random Undersampling, and SMOTE—to handle the class imbalance issue.

Two popular classifiers, Random Forest and XGBoost, were trained and tested on each sampling version. The evaluation metrics used include Accuracy, Confusion Matrix, and Classification Report (Precision, Recall, F1-Score).

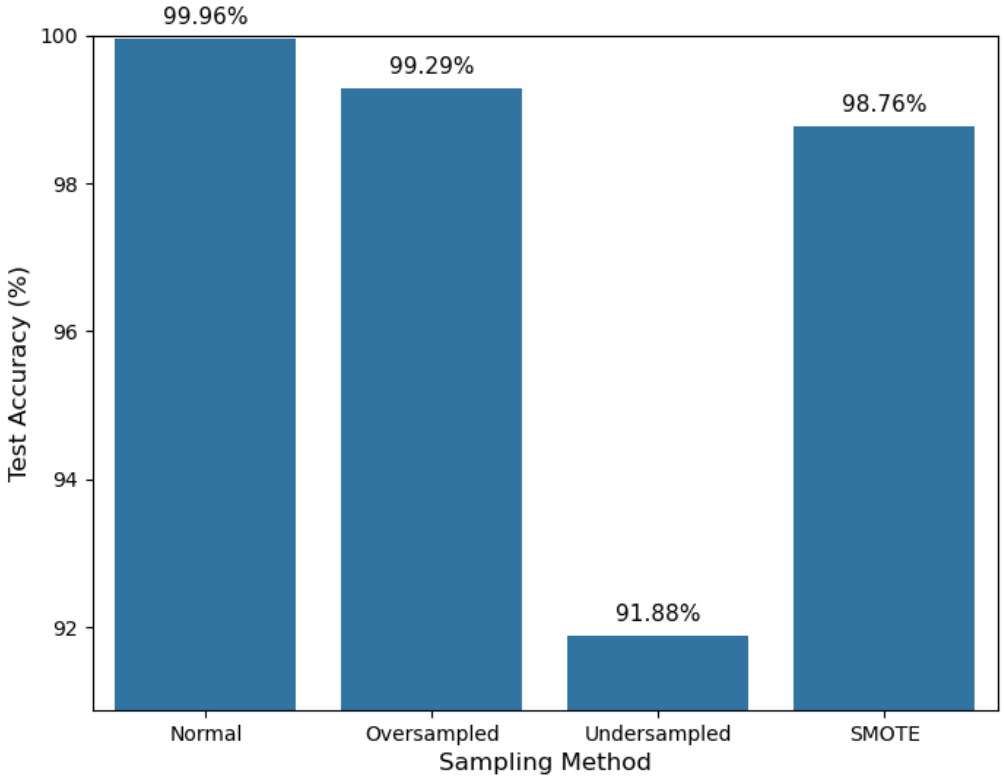
The experiment showed that:

- XGBoost with Oversampling gave the highest accuracy of approximately 99.99%, making it the best-performing model.
- Random Forest with SMOTE also performed well with an accuracy above 99%.
- Results were compared visually using bar plots showing accuracy across different sampling methods.

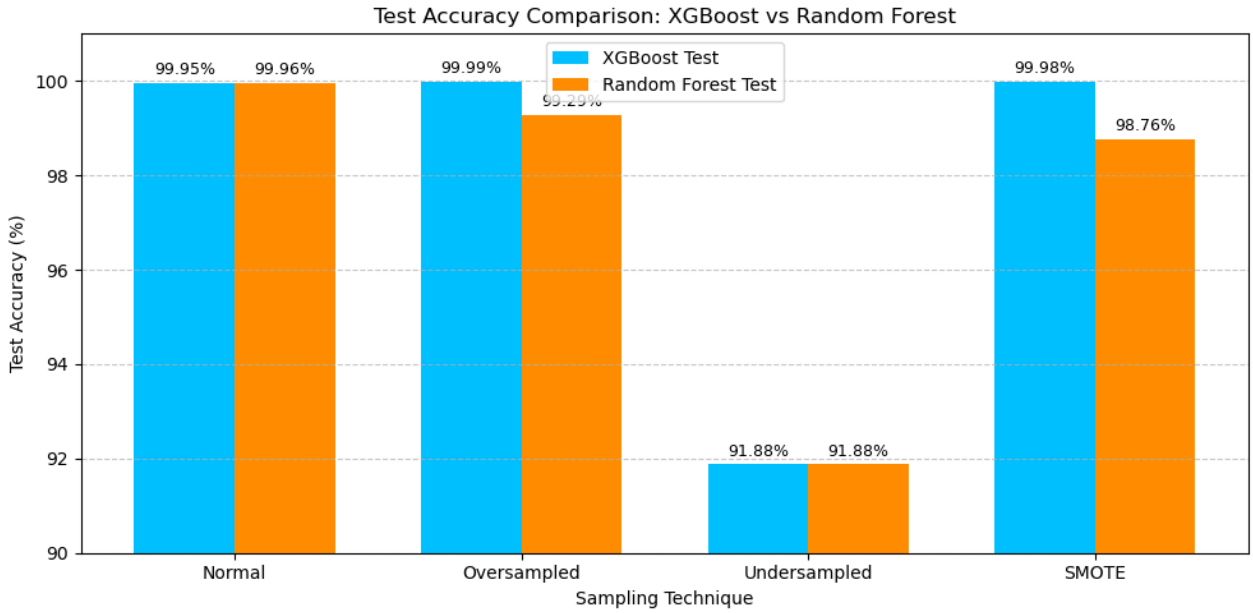
Bar Plot comparing XGBoost accuracies (Normal, Over, Under, SMOTE)



Bar Plot comparing Random Forest accuracies (Normal, Over, Under, SMOTE)



Combined Bar Plot – showing both XGBoost and Random Forest test accuracies



7. CONCLUSION AND FUTURE SCOPE

In this project, machine learning techniques were successfully utilized to detect fraudulent credit card transactions, with special emphasis on handling the class imbalance issue using resampling methods like Random Oversampling, Undersampling, and SMOTE. Among the models tested, XGBoost trained on oversampled data yielded the best performance in terms of accuracy and reliability. The chosen model was deployed using a Streamlit-based user interface for real-time fraud prediction. For future enhancements, the system can be improved with more advanced algorithms, continuous learning from live data, and further optimization for better real-time performance and scalability.

8. REFERENCES

Kaggle. *Credit Card Fraud Detection Dataset*. https://www.kaggle.com/mlg-ulb/creditcardfraud

AlsharifHasan Mohamad Aburbeian, Huthaifa I. Ashqar. *Credit Card Fraud Detection Using Enhanced Random Forest Classifier for Imbalanced Data*, ResearchGate, 2023. https://www.researchgate.net

Tushar A. Sontakke, Dr. V. B. Gadicha. *Credit Card Fraud Detection Using Machine Learning Techniques*, IEEE Xplore, 2022.

Chen, T., & Guestrin, C. *XGBoost: A Scalable Tree Boosting System*. Proceedings of the 22nd ACM SIGKDD, 2016.

Streamlit Documentation. *Streamlit: Turn data scripts into shareable web apps*. https://docs.streamlit.io

Dhanashree S. Sonawane and Prof. Shweta Ghatage, "Credit Card Fraud Detection Using XGBoost and Ensemble Methods," *ResearchGate*, 2020.

Jisha J. John and Nitha K. A., "Credit Card Fraud Detection Using Machine Learning Algorithms," *IEEE Xplore*, 2021.