Annexure-II

Cover Page

Credit Card Fraud Detection

A Project Report

Submitted in partial fulfilment of the requirements for the

Award of the degree of MCA

"<u>Master of Computer Application</u>"

By

Rohit Kumar

(323101970)

Centre for Distance and Online Education

Lovely PROFESSIONAL UNIVERSITY PHAGWARA, PUNJAB

(2023-25)

Annexure-III

Declaration by the Student

To whom-so-ever it may concern

I, **Rohit Kumar, 323101970** hereby declare that the work done by me on "**Credit Card Fraud Detection**", is a record of original work for the partial fulfilment of the requirements for the award of the degree, **MCA.**

Rohit Kumar (323101970)

Rohit Kumar

Signature of the student

Dated: 14th May, 2025

Acknowledgement

I sincerely thank my project guide and faculty members for their valuable guidance and support throughout this project. I am also grateful to my friends and family for their continuous encouragement. Special thanks to the creators of the open-source tools and the Kaggle community for providing access to the dataset used in this project.

Annexure-I
Project Summary

Registration No: 323101970 Name of student: Rohit Kumar

Title of Capstone Project / Project Work: Credit Card Fraud Detection using ML

Objectives of the Project:

1. To detect fraudulent credit card transactions using machine learning
2. To compare Logistic Regression, Decision Tree, and Random Forest classifiers
3. To improve model performance on imbalanced data using SMOTE.
4. To build a simple interactive interface using Streamlit.

Results and Findings:

After handling class imbalance using SMOTE, all three machine learning models:
Logistic Regression, Decision Tree, and Random Forest - showed improvement in performance metrics.
The evaluation was based on accuracy, precision, recall, and F1-score using the confusion matrix and classification report.

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Regression	0.9462	0.9727	0.9182	0.9447
Decision Tree	0.9981	0.9972	0.9989	0.9981
Random Forest	0.9999	0.9997	1.0000	0.9999

Specific outcomes of the Project:

1. Implemented Logistic Regression, Decision Tree, and Random Forest models for credit card fraud detection.
2. Achieved 99.99% accuracy with Random Forest after addressing class imbalance using SMOTE.
3. Developed a Streamlit-based interface for real-time fraud prediction and visualization.

Learnings from the Project:

1. Gained hands-on experience in handling imbalanced datasets using SMOTE and evaluating classification models.
2. Learned to implement and compare multiple machine learning algorithms effectively.
3. Acquired practical skills in building interactive user interfaces using Streamlit for real-time model deployment.

Any Challenges/issues faced during the Project:

1. **Class Imbalance:** The original dataset had a highly skewed distribution of fraudulent vs. non-fraudulent transactions, which affected model performance until SMOTE was applied.

2. **Overfitting Risk:** Complex models like Random Forest showed high accuracy, requiring careful evaluation to ensure generalization.

3. **Streamlit UI Integration:** Designing a user-friendly and functional interface while linking it with the backend model involved debugging and layout adjustments.

4. **Data Preprocessing:** Understanding and cleaning the dataset took considerable effort, especially handling outliers and scaling features properly.

Rohit Kumar

Signature of the Student:

Table of Content

Chapter-1 Introduction

1.1 Aim

The aim of this project is to build an effective machine learning model that can accurately detect fraudulent credit card transactions. By learning patterns from historical data, the goal is to develop a system that identifies fraud in real time, helping reduce financial losses.

1.2 Importance

With the growth of digital payments and e-commerce, credit card fraud has become a major concern for individuals, businesses, and financial institutions. An intelligent fraud detection system can prevent unauthorized transactions, enhance customer trust, and protect sensitive financial data.

1.3 Applicability

The machine learning methods used in this project can be applied in various sectors such as:

- Banks and financial institutions for secure transaction monitoring
- Online shopping platforms to verify payment authenticity
- Fintech apps to monitor suspicious user behaviour
- Insurance and loan departments for fraud claim detection

1.4 Problem Statement

Traditional fraud detection systems rely on fixed rules and thresholds, which are not flexible enough to detect new and evolving fraud patterns. They either miss genuine fraud or flag too many false positives. A more adaptive, data-driven approach is needed to overcome these limitations.

1.5 Scope of the Project

This project focuses on:

- Preprocessing real-world transaction data
- Applying and comparing three machine learning algorithms: Logistic Regression, Decision Tree, and Random Forest
- Selecting the best-performing algorithm based on evaluation metrics
- Implementing the final model to detect fraud effectively

1.5 Methodology Overview

- Collection and understanding of the dataset
- Data preprocessing and handling of class imbalance using SMOTE
- Training and testing three machine learning models
- Evaluating each using accuracy, precision, recall, and F1-score and choosing best model

Chapter-2 Review of Literature

Credit card fraud poses a serious threat to the financial sector, especially with the rise of digital transactions. Traditional methods of detecting fraud, such as rule-based systems, often fall short due to their inability to adapt to new or evolving fraudulent patterns. This led me to explore machine learning techniques, which are widely acknowledged in recent studies for their effectiveness in identifying complex and subtle patterns in transaction data.

2.1 Understanding Credit Card Fraud Detection

Through my research, I learned that credit card fraud detection has evolved significantly. Earlier systems relied heavily on static rules and manual thresholds, which could not keep up with rapidly changing fraud tactics. In contrast, machine learning models can learn from past transaction data to predict future fraudulent activities more accurately.

I came across several commonly used machine learning algorithms for fraud detection, including Logistic Regression, Decision Trees, and Random Forests. These models vary in complexity and performance. For instance, Logistic Regression is known for its simplicity and speed, but it may not perform well with non-linear data. Decision Trees, on the other hand, can handle non-linear relationships but are prone to overfitting. Random Forests, which are ensemble models, offer better generalization by combining multiple Decision Trees and reducing the risk of overfitting.

2.2 The Challenge of Imbalanced Datasets

One of the major issues I encountered during this project was the **class imbalance** in the dataset. The credit card fraud dataset I used from Kaggle was highly skewed—fraudulent transactions made up a very small percentage of the total data. This imbalance posed a challenge because most machine learning algorithms tend to favour the majority class, leading to high accuracy but poor detection of actual fraud cases.

To handle this, I experimented with **under-sampling** and **over-sampling** techniques. Under-sampling helped by reducing the size of the majority class, while over-sampling increased the number of fraudulent cases in the dataset. I found that these techniques significantly improved the model's ability to learn patterns from both classes.

2.3 Model Selection and Performance Comparison

After preparing the data, I trained three machine learning models: Logistic Regression, Decision Tree, and Random Forest. I evaluated each model's performance using not only **accuracy**, but also

precision, **recall**, and **F1-score**, which are more appropriate metrics for imbalanced classification problems.

I compared the results for each model under the original, under-sampled, and over-sampled datasets. While under-sampling helped improve recall to some extent, over-sampling produced better overall performance across all metrics. Among the three models, **Random Forest consistently gave the best results on the over-sampled data**, with the highest F1-score and precision-recall balance.

2.4 Key Learnings from Existing Studies

My findings are in line with various research studies that I reviewed. Many authors have highlighted the effectiveness of ensemble models, particularly Random Forests, in fraud detection scenarios—especially when combined with resampling methods to handle class imbalance. Literature also emphasizes the importance of using precision, recall, and F1-score over accuracy when evaluating models on imbalanced data, which reinforced my own evaluation strategy.

2.5 Summary

In summary, the literature provided a strong foundation for my project, guiding my approach to model selection, handling data imbalance, and evaluating performance. Based on both theoretical understanding and experimental results, I chose **Random Forest** trained on an **over-sampled dataset** as the final model for my credit card fraud detection system. It offered the best balance between detecting actual fraud cases and minimizing false positives, which is crucial for any real-world application.

Chapter-3 Implementation of Project

This chapter presents the step-by-step implementation of the credit card fraud detection project using machine learning. The objective is to build a robust predictive model capable of identifying fraudulent transactions from a highly imbalanced dataset. The entire process includes data exploration, preprocessing, sampling strategies, model training, evaluation, and deployment via a web interface using Streamlit.

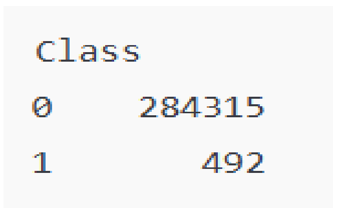
3.1 Tools and Libraries Used

The following libraries were used throughout the implementation:

- **pandas, numpy**: For data manipulation and analysis.
- **matplotlib, seaborn**: For data visualization.
- **scikit-learn**: For preprocessing, model building, and evaluation.
- **imblearn**: For handling class imbalance using RandomUnderSampler and SMOTE.
- **joblib**: For saving the trained model.
- **streamlit**: For building the user interface

3.2 Loading the Dataset

- The dataset was loaded using the pandas library from a CSV file named 'creditcard.csv'.
- Basic information about the dataset was obtained using-
    1. **df.shape** - to get dimensions (no. of row, no. of column) of dataframe.
    2. **df.info()** - to display a concise summary of a dataframe.
    3. **df.describe**() - to generate descriptive statistics for numeric columns in a dataframe.
    4. **df.isnull().sum()** – to check null value.
- The class distribution revealed a high imbalance which means out of 284,807 only 492 are fraud transaction rest of are legit.



```
Class
0    284315
1       492
```

3.3 Data Exploration and Visualization

To understand the class distribution visually, a countplot was created using seaborn:

Code:
```
sns.countplot(x='Class', data=df)
plt.title('Class Distribution')
plt.show()
```

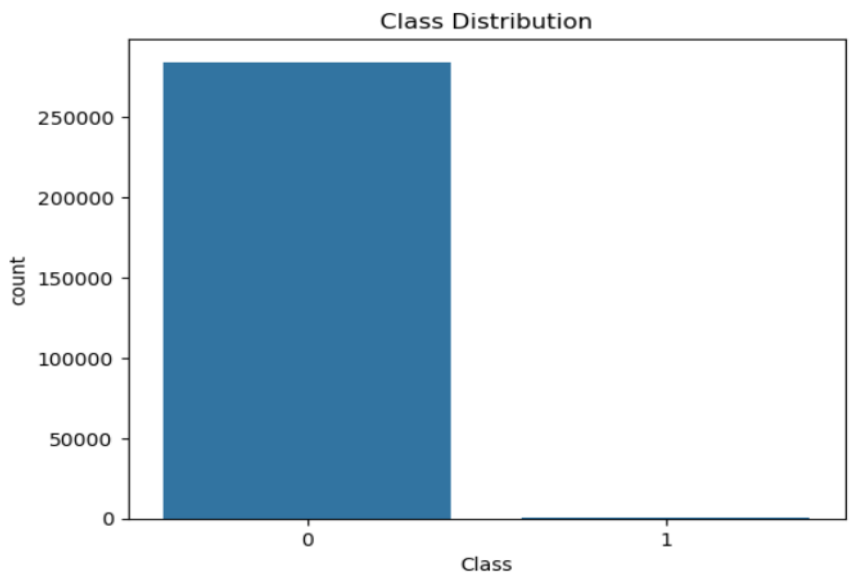


Fig 3.1

3.4 Data Preprocessing

- The **Time** column was dropped as it did not contribute significantly to prediction.
- The **Amount** column was scaled using StandardScaler to normalize the monetary values for better model performance.

Code:
```
df = df.drop(['Time'], axis = 1)
scaler = StandardScaler()
df['Amount'] = scaler.fit_transform(df[['Amount']])
```

- After dropping scaling **Amount** column **Features** (all 28 vectors) and **Target** (Class) were split.

Code:
```
X = df.drop(['Class'], axis = 1)
y = df['Class']
```

3.5 Handling Class Imbalance

3.5.1 Under Sampling

Random under sampling was applied to balance the class distribution by reducing the number of legitimate transactions to match the fraudulent ones:

Code:
```
rus = RandomUnderSampler(random_state=42)
X_under, y_under = rus.fit_resample(X, y)
print("Under Sampled Data Shape:", X_under.shape)
sns.countplot(x=y_under)
plt.title("Under Sampled Class Distribution")
plt.show()
```

Result: Under Sampled Data Shape: (984, 29)

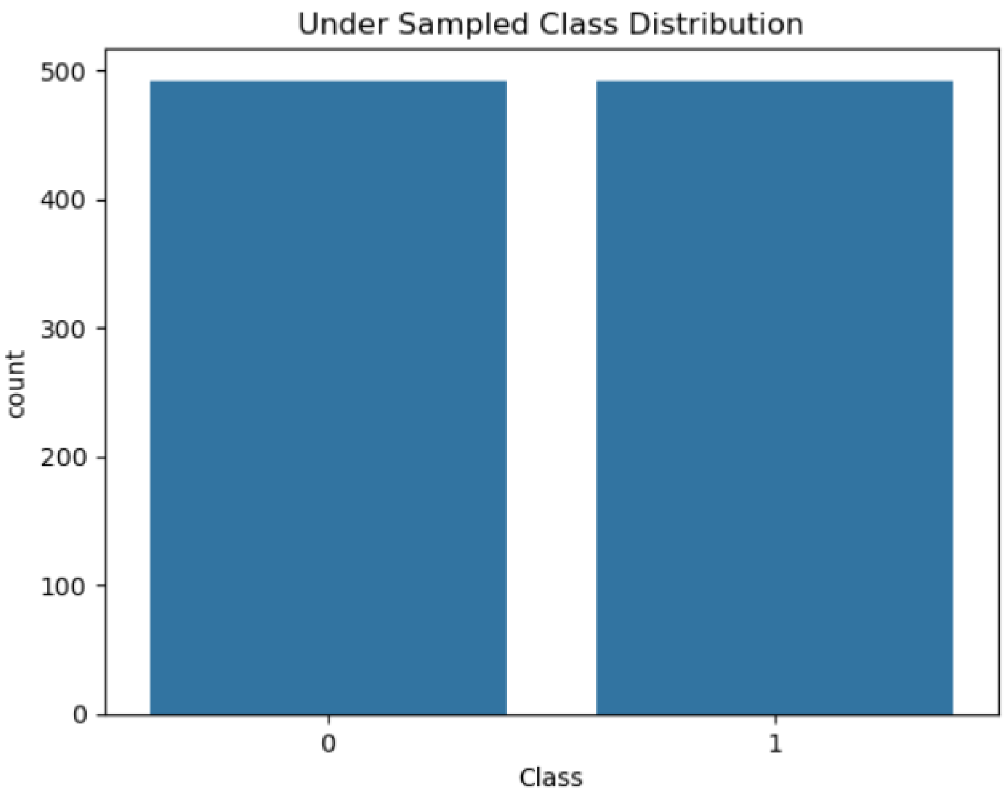


Fig 3.2

3.5.2 Over-Sampling Using SMOTE

Since under-sampling involved removing a large portion of the majority class, which could lead to the loss of valuable information and reduced model accuracy, I chose to use over-sampling instead.

To achieve this, I applied SMOTE (Synthetic Minority Over-sampling Technique), which generates synthetic examples of the minority class (fraudulent transactions) rather than simply duplicating existing ones. This helped balance the dataset while preserving all of the original data, allowing the model to learn better patterns from both classes.

Code:
```
smote = SMOTE(random_state=42)
X_over, y_over = smote.fit_resample(X, y)
print("Over Sampled Data Shape:", X_over.shape)
sns.countplot(x=y_over)
plt.title("Over Sampled Class Distribution")
plt.show()
```

Result:
```
Over Sampled Data Shape: (568630, 29)
```

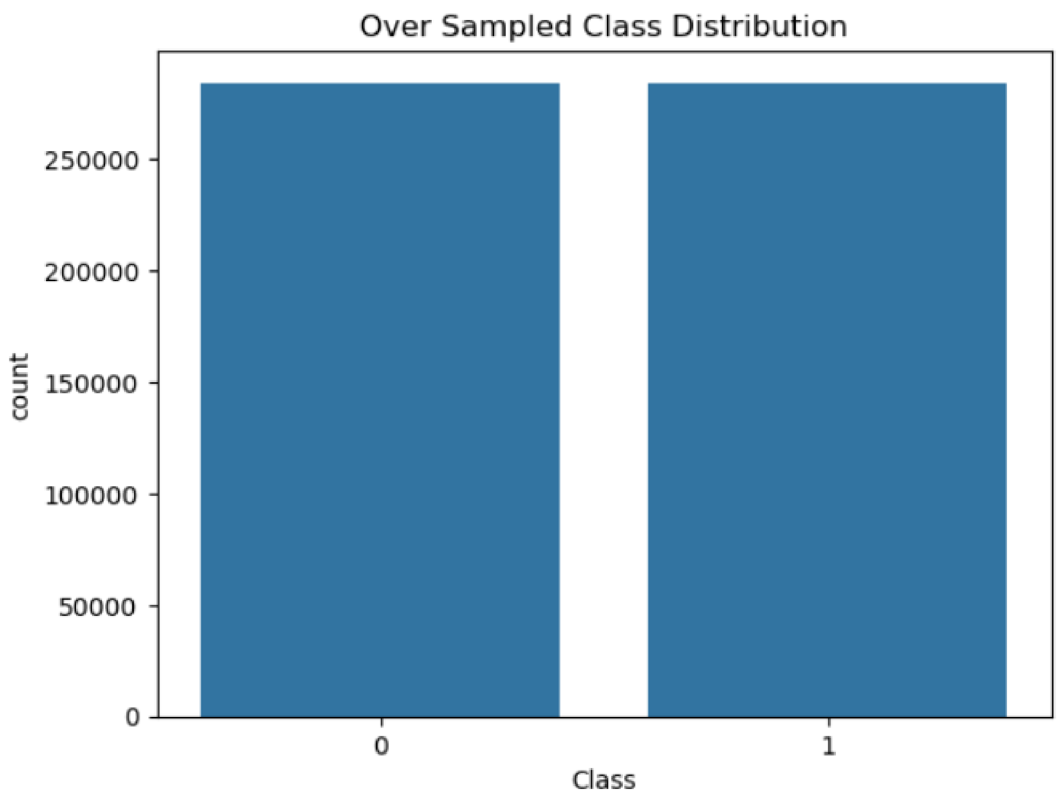


Fig.3.3

3.6 Splitting the Dataset

The balanced dataset was divided into **training and testing sets** using an **80:20 ratio**, where 80% of the data was used for training the model and 20% was reserved for evaluating its performance.

Code:
```
X_train, X_test, y_train, y_test = train_test_split(X_over, y_over, test_size=0.2, random_state=42, stratify=y_over)
```

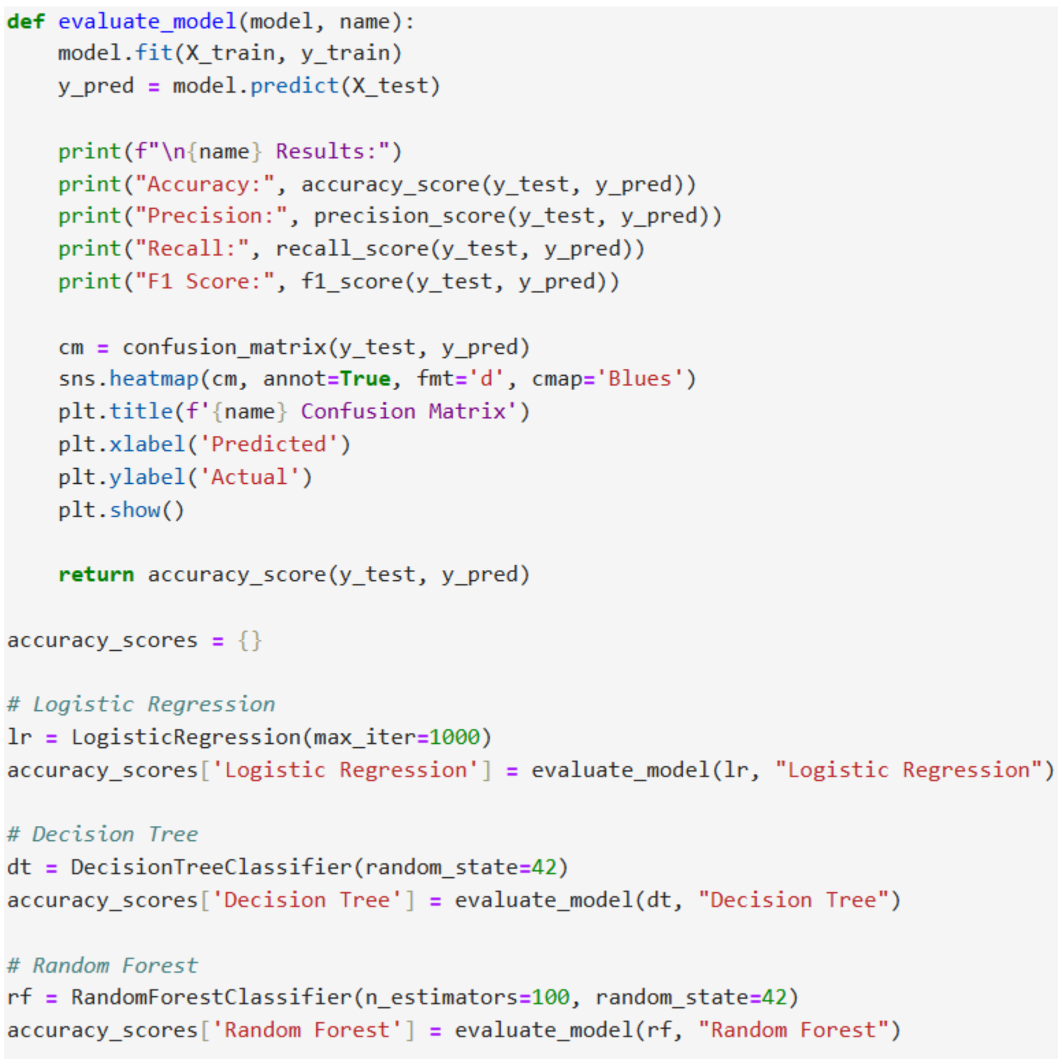
3.7 <u>Model Building and Evaluation</u>

Three machine learning models were trained and evaluated for the task of credit card fraud detection:

- Logistic Regression
- Decision Tree
- Random Forest

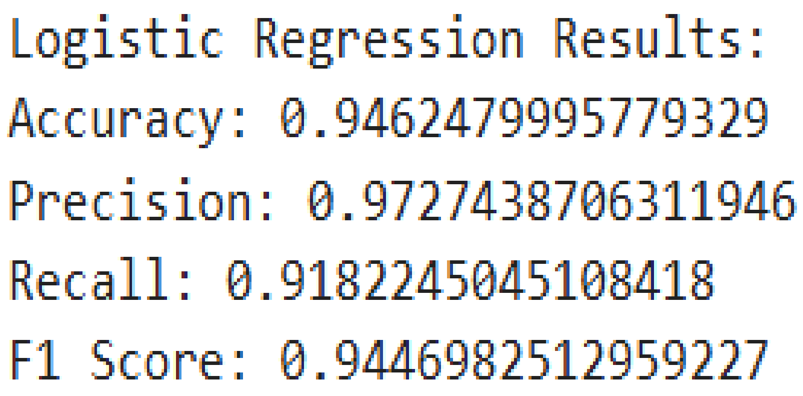
To ensure consistent and comprehensive evaluation, a custom function was created. This function computes key performance metrics—Accuracy, Precision, Recall, and F1 Score—for each model. Additionally, it displays the confusion matrix, providing a clear visual summary of the model's prediction results and helping to better understand how well each model distinguishes between fraudulent and non-fraudulent transactions.

Evaluation Function and Training of Model:



```python
def evaluate_model(model, name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    print(f"\n{name} Results:")
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print("Precision:", precision_score(y_test, y_pred))
    print("Recall:", recall_score(y_test, y_pred))
    print("F1 Score:", f1_score(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'{name} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    return accuracy_score(y_test, y_pred)

accuracy_scores = {}

# Logistic Regression
lr = LogisticRegression(max_iter=1000)
accuracy_scores['Logistic Regression'] = evaluate_model(lr, "Logistic Regression")

# Decision Tree
dt = DecisionTreeClassifier(random_state=42)
accuracy_scores['Decision Tree'] = evaluate_model(dt, "Decision Tree")

# Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
accuracy_scores['Random Forest'] = evaluate_model(rf, "Random Forest")
```

Output of Code:



```
Logistic Regression Results:
Accuracy: 0.9462479995779329
Precision: 0.9727438706311946
Recall: 0.9182245045108418
F1 Score: 0.9446982512959227
```

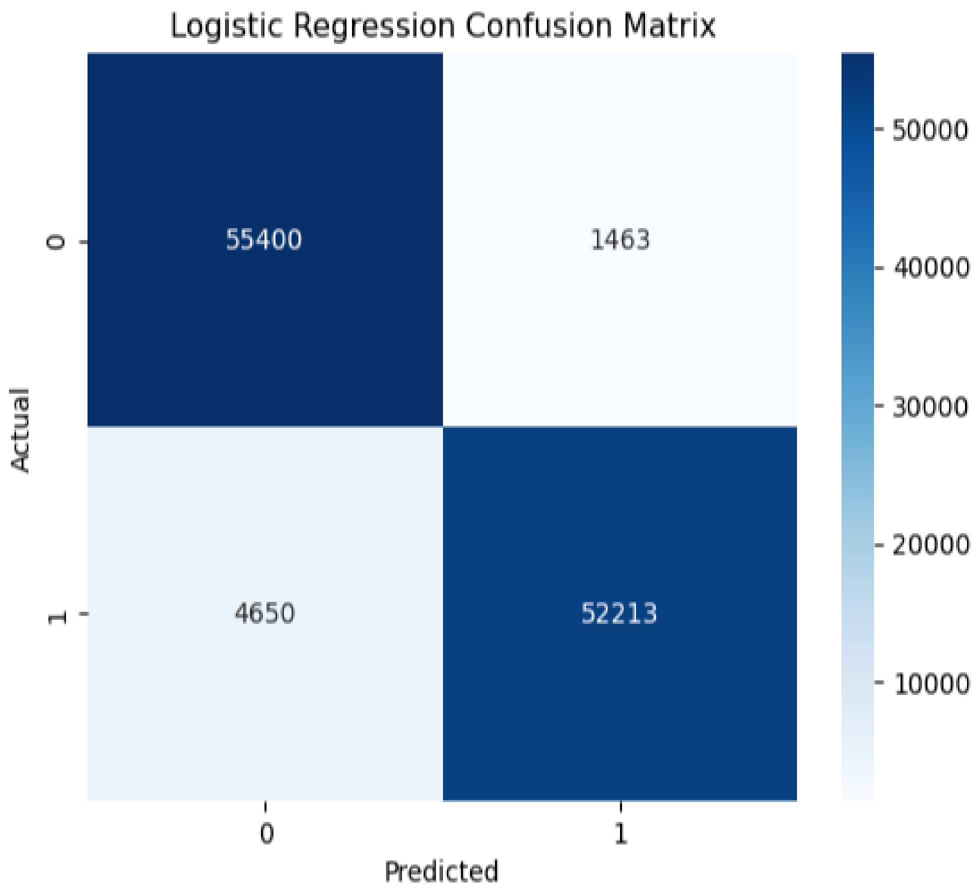
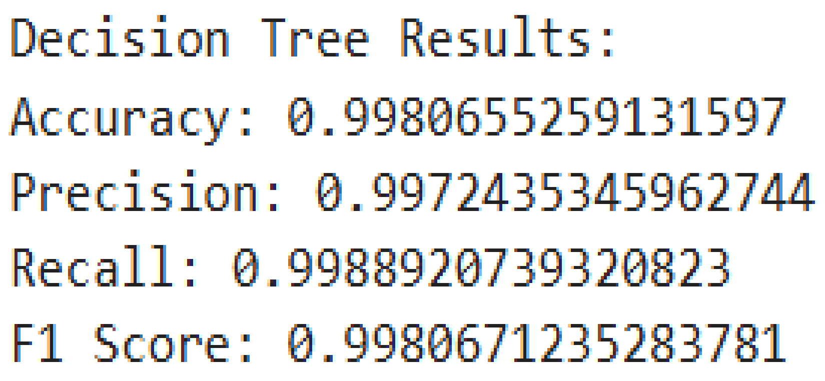


Fig 3.4



```
Decision Tree Results:
Accuracy: 0.9980655259131597
Precision: 0.9972435345962744
Recall: 0.9988920739320823
F1 Score: 0.9980671235283781
```

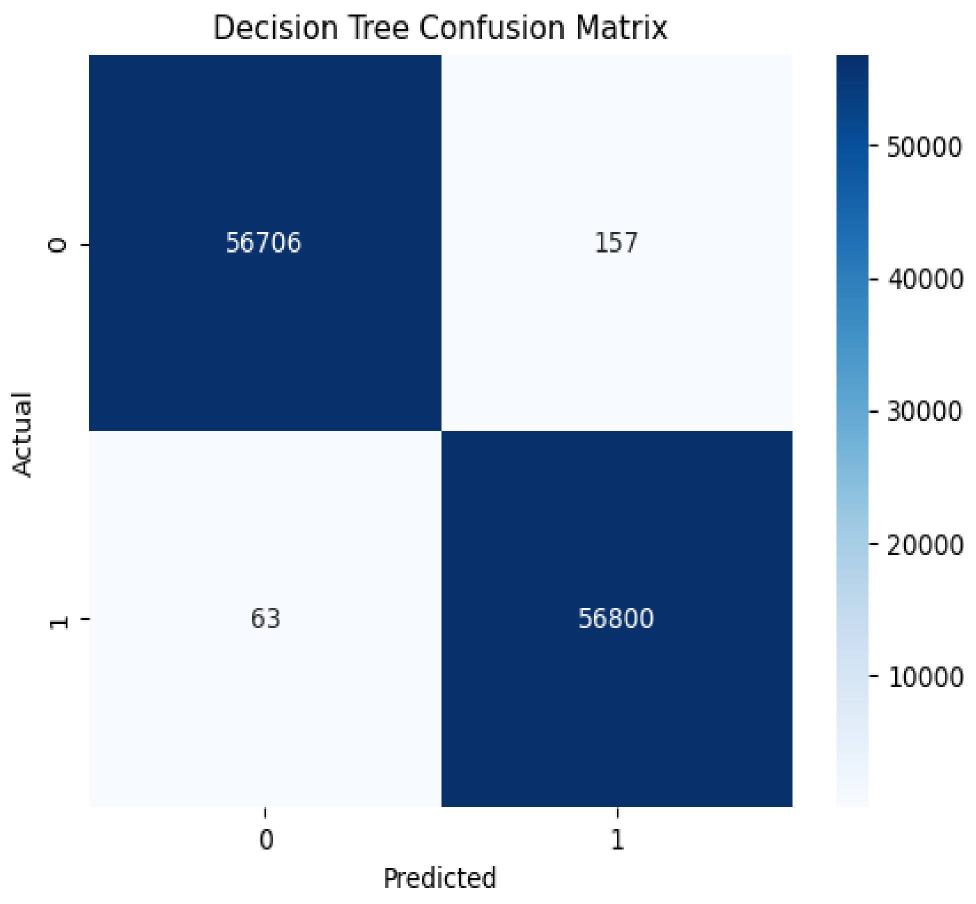
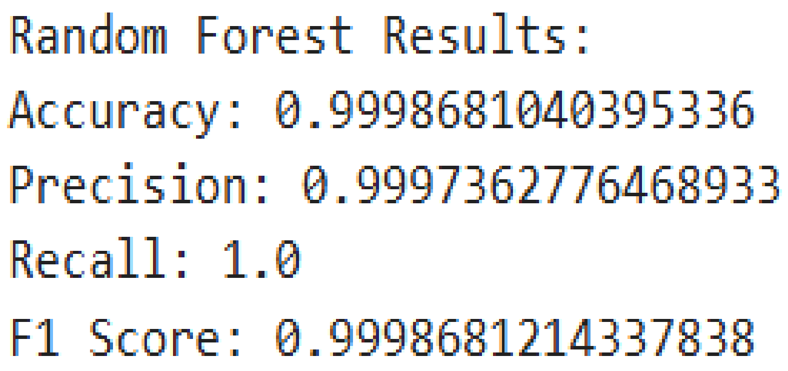


Fig 3.5



```
Random Forest Results:
Accuracy: 0.9998681040395336
Precision: 0.9997362776468933
Recall: 1.0
F1 Score: 0.9998681214337838
```

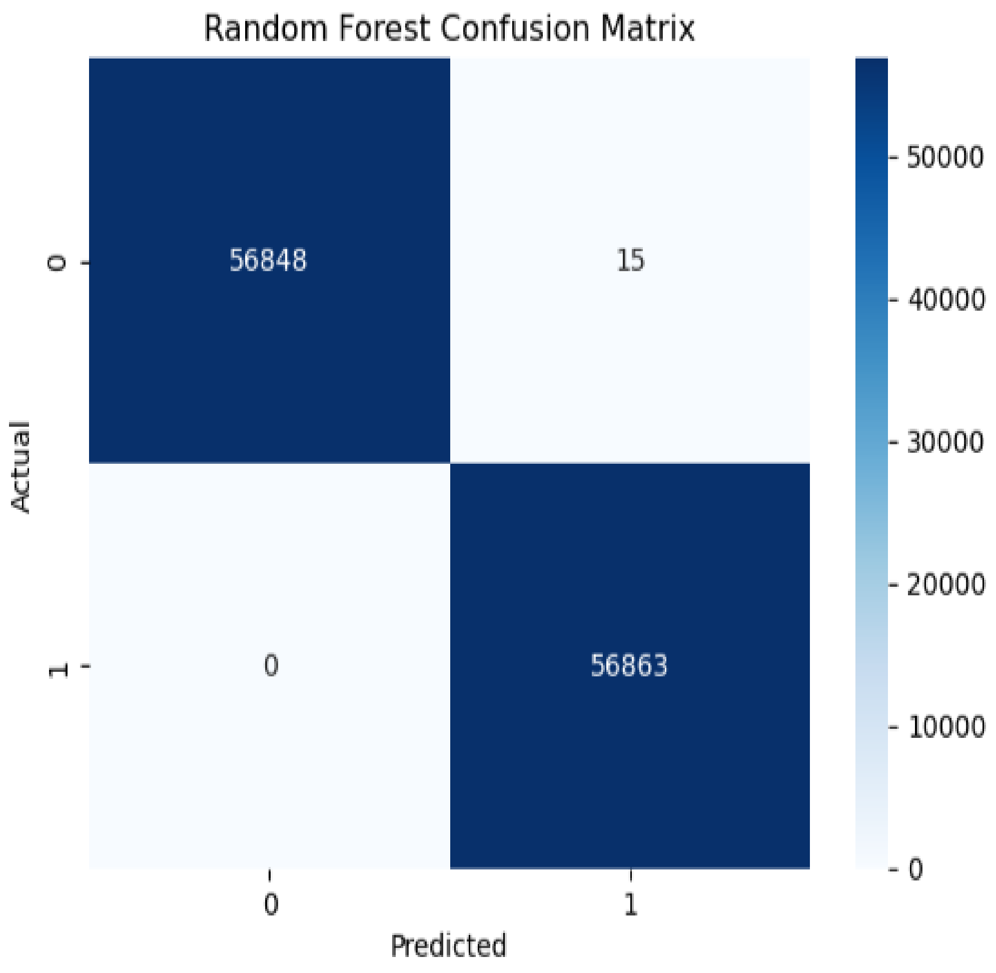


Fig 3.6

3.8 Model Comparison

To determine the most effective algorithm for credit card fraud detection, the three trained models—Logistic Regression, Decision Tree, and Random Forest—were compared using both numerical metrics and visual tools.

A bar plot was used to compare the accuracy scores of each model, providing a quick and intuitive visual representation of overall performance. In addition to accuracy, other important evaluation metrics such as Precision, Recall, and F1 Score were also taken into consideration, as they are particularly crucial in imbalanced classification problems where accuracy alone can be misleading.

The comparison revealed the following insights:

- Random Forest achieved the highest accuracy, as well as the best balance across all other evaluation metrics.
- Logistic Regression performed reasonably well in terms of precision but lagged behind in recall, indicating it missed more actual fraud cases.
- Decision Tree offered moderate performance but was prone to overfitting compared to the ensemble-based Random Forest model.

This comprehensive comparison allowed for a well-informed decision in selecting Random Forest as the final model, as it demonstrated superior performance in detecting fraudulent transactions while maintaining a strong balance between sensitivity and precision.

Code:

```python
plt.figure(figsize=(8,5))
sns.barplot(x=list(accuracy_scores.keys()), y=list(accuracy_scores.values()))
plt.ylabel('Accuracy Score')
plt.title('Model Accuracy Comparison')
plt.ylim(0.9, 1.0)
plt.show()
```

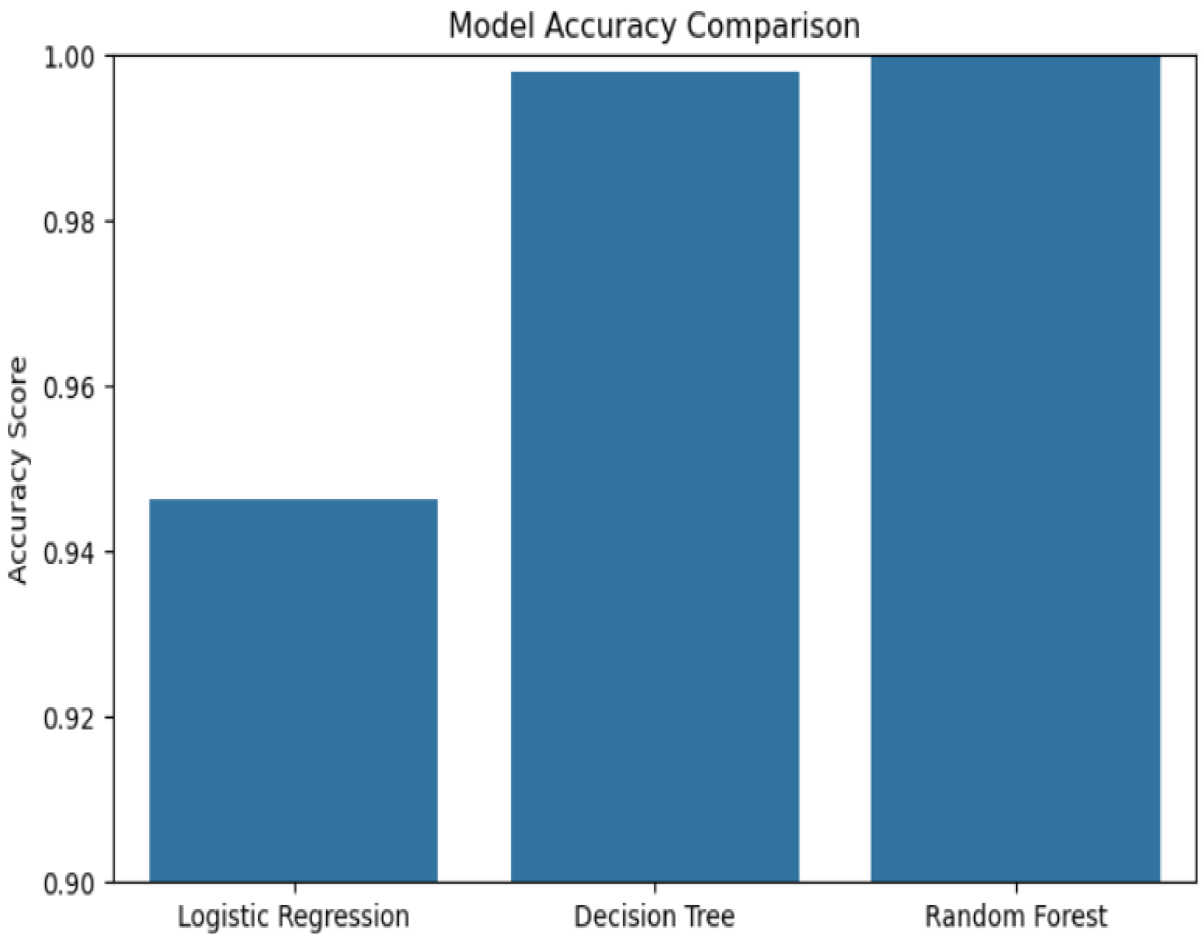


Fig 3.7

3.9 <u>Model Selection and Saving</u>

After evaluating and comparing the performance of all three machine learning models – **Regression**, **Decision Tree**, and **Random Forest** - based on metrics such as **accuracy**, **precision**, **recall**, and **F1-score**, the **Random Forest classifier** emerged as the most effective model for detecting credit card fraud. It demonstrated the best balance between detecting actual fraud cases and minimizing false positives.

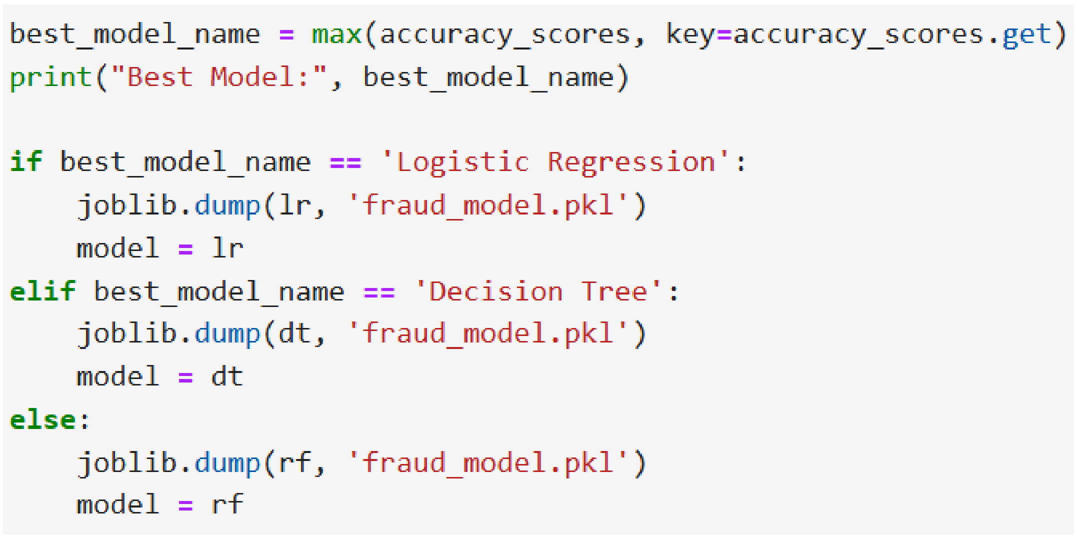
To preserve this trained model for future use, I used the **joblib** library, which is well-suited for saving machine learning models that contain large NumPy arrays (such as tree-based models). This allows me to avoid retraining the model every time the application runs and enables quick deployment or further testing later on.

Saving the model has several benefits:

- **Reusability**: The model can be easily reloaded for predictions without retraining.

- **Efficiency**: Reduces computation time by avoiding repeated training.
- **Integration**: Makes it easier to integrate the model into production systems or web applications.

The saved model file can be loaded later using **joblib.load()**, making it convenient to deploy in real-time fraud detection systems or further refine with new data.

Code: 

```
best_model_name = max(accuracy_scores, key=accuracy_scores.get)
print("Best Model:", best_model_name)

if best_model_name == 'Logistic Regression':
    joblib.dump(lr, 'fraud_model.pkl')
    model = lr
elif best_model_name == 'Decision Tree':
    joblib.dump(dt, 'fraud_model.pkl')
    model = dt
else:
    joblib.dump(rf, 'fraud_model.pkl')
    model = rf
```

Result: Best Model: Random Forest

3.10 Creating a Streamlit Interface

To make the fraud detection model accessible and user-friendly, I developed a simple web interface using **Streamlit**, an open-source Python framework for building interactive web applications.

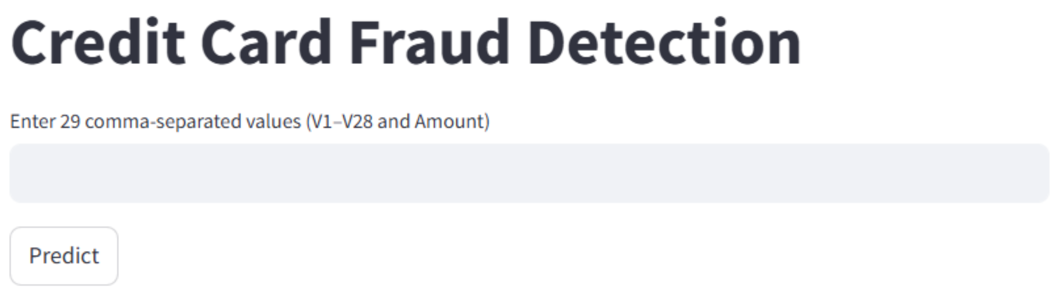
The Streamlit app allows users to input transaction data in the form of **29 values**—including **V1 to V28** (the anonymized features) and the **Amount**—entered as **comma-separated numbers**. Once the user submits the input, the app processes the data, feeds it into the trained Random Forest model, and displays the prediction result, indicating whether the transaction is likely to be **fraudulent** or **legitimate**.

This interface enables easy testing and demonstration of the model's functionality without requiring any technical background or coding knowledge from the user.

User Interface Code:

```python
# Load model
model = joblib.load('fraud_model.pkl')

# Define feature names
feature_names = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10',
                 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20',
                 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount']

st.title("Credit Card Fraud Detection")

# Single input box for all features
csv_input = st.text_input("Enter 29 comma-separated values (V1-V28 and Amount)", "")

if st.button("Predict"):
    try:
        # Convert CSV string to list of floats
        user_input = [float(x.strip()) for x in csv_input.split(",")]

        if len(user_input) != 29:
            st.error("Please enter exactly 29 comma-separated values.")
        else:
            input_df = pd.DataFrame([user_input], columns=feature_names)
            prediction = model.predict(input_df)
            result = "🔴 Fraudulent Transaction" if prediction[0] == 1 else "🟢 Normal Transaction"
            st.success(f"Prediction: {result}")
    except ValueError:
        st.error("Invalid input. Please make sure all values are numbers.")
```

User Interface after running my code:

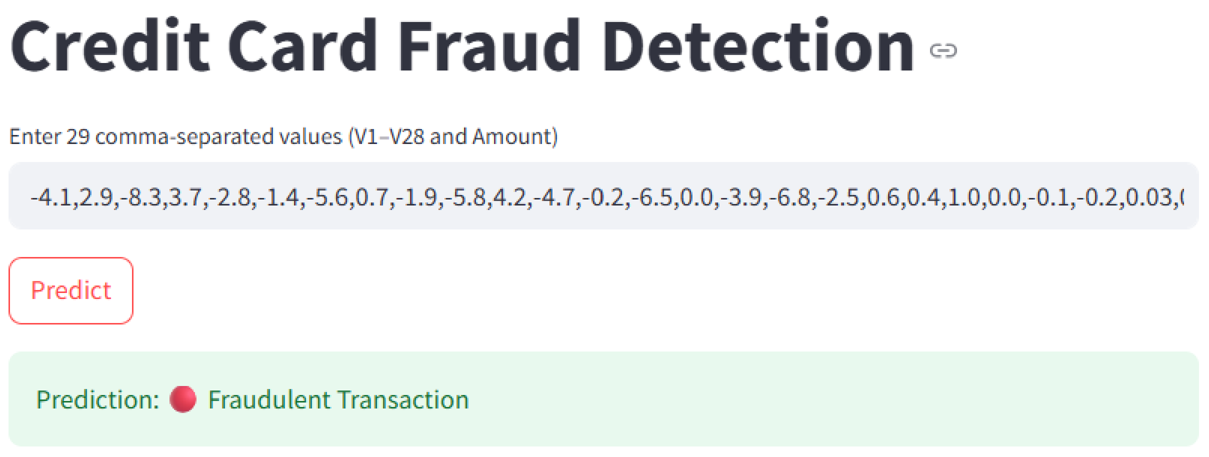


Credit Card Fraud Detection

Enter 29 comma-separated values (V1–V28 and Amount)

Fig 3.8

Screen Shots where my model is predicting Fraudulent and Normal transaction



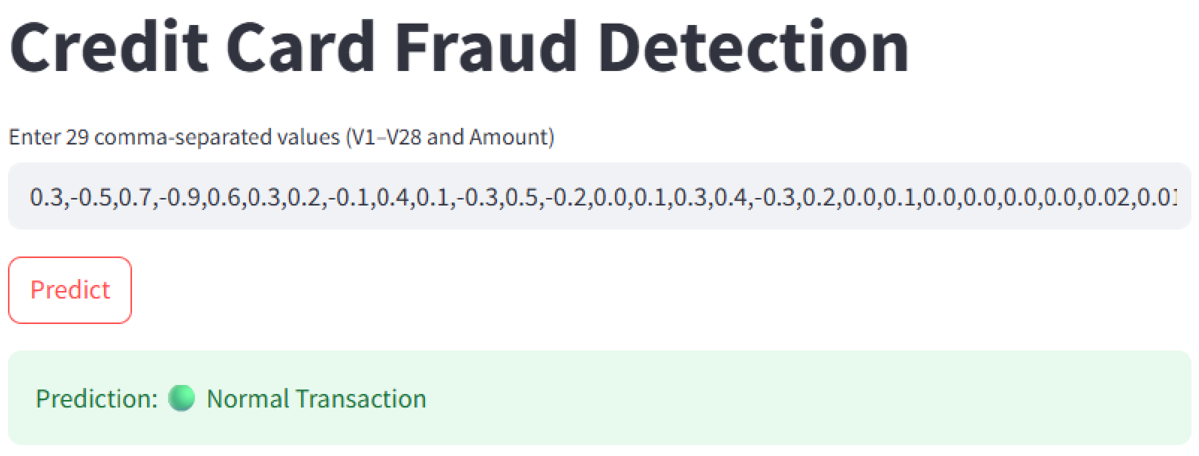
Credit Card Fraud Detection

Enter 29 comma-separated values (V1–V28 and Amount)

-4.1,2.9,-8.3,3.7,-2.8,-1.4,-5.6,0.7,-1.9,-5.8,4.2,-4.7,-0.2,-6.5,0.0,-3.9,-6.8,-2.5,0.6,0.4,1.0,0.0,-0.1,-0.2,0.03,(

Prediction: 🔴 Fraudulent Transaction

Fig 3.9



Credit Card Fraud Detection

Enter 29 comma-separated values (V1–V28 and Amount)

0.3,-0.5,0.7,-0.9,0.6,0.3,0.2,-0.1,0.4,0.1,-0.3,0.5,-0.2,0.0,0.1,0.3,0.4,-0.3,0.2,0.0,0.1,0.0,0.0,0.0,0.0,0.02,0.01

Prediction: 🟢 Normal Transaction

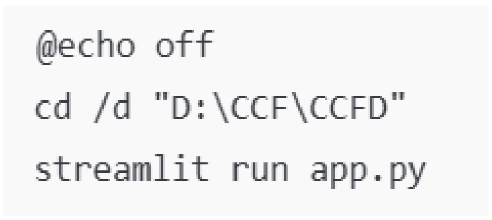
Fig 3.10

3.11 Deployment Setup

To make the fraud detection system easily accessible, I organized all the necessary components into a deployable folder structure. The following files are included:

- **fraud_model.pkl**: This is the saved Random Forest model, trained on the balanced dataset using over-sampling. It was saved using the joblib library for efficient storage and quick loading during predictions.
- **app.py**: This Python script contains the **Streamlit user interface**. It allows users to enter transaction data (features V1 to V28 and Amount) and receive a real-time prediction indicating whether the transaction is likely to be fraudulent or legitimate.
- **app.bat**: To simplify the launch process, I created a **batch file** that automatically runs the Streamlit app. This allows the application to be started with a double-click, without needing to manually open a terminal or enter commands.

  app.bat file code:



```
@echo off
cd /d "D:\CCF\CCFD"
streamlit run app.py
```

All three files are placed in the **same directory** to ensure smooth execution. This setup enables users to run the app with a single action, making the solution both practical and user-friendly.

Chapter-4 Results and Discussions

4.1 Results

The performance of all three models—Logistic Regression, Decision Tree, and Random Forest—was evaluated after training on the balanced dataset. The following metrics summarize their performance:

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.9462	0.9727	0.9182	0.9447
Decision Tree	0.9981	0.9972	0.9989	0.9981
Random Forest	0.9999	0.9997	1.0000	0.9999

Among all models, Random Forest demonstrated the highest accuracy and the best balance between precision and recall.

4.2 Discussion

The results clearly indicate that Random Forest outperformed the other models in detecting fraudulent transactions. Its ensemble approach contributed to better generalization and robustness against overfitting.

Compared to Logistic Regression and Decision Tree, Random Forest achieved a superior trade-off between correctly identifying fraud and minimizing false positives. This makes it a strong candidate for real-world fraud detection systems, where both sensitivity and precision are critical.

The model was further integrated into a Streamlit-based interface, allowing users to easily test the system by entering transaction data and instantly receiving prediction outcomes. This added layer of usability demonstrates the model's potential for practical deployment in financial applications.

Final Chapter- Conclusion and Future Scope

Conclusion

This project successfully demonstrated the application of machine learning techniques for credit card fraud detection, addressing the challenges posed by imbalanced datasets. By incorporating SMOTE for oversampling and leveraging ensemble methods—particularly the Random Forest classifier—the system achieved strong predictive performance in identifying fraudulent transactions.
Additionally, the integration of a Streamlit-based user interface enhanced the usability of the model, allowing for interactive and accessible fraud prediction. The combination of data preprocessing, model selection, and user interface design resulted in a complete and practical fraud detection solution.

Future Scope

The project can be further improved and expanded in the following ways:

- The system can be enhanced by implementing real-time fraud detection using streaming transaction data, allowing immediate identification and prevention of fraudulent activities.
- The current binary classification model can be extended to a multi-class classification system, enabling the detection and categorization of different types or levels of fraudulent behaviour
- To ensure ongoing effectiveness, the model should be periodically retrained with updated transaction data to adapt to changing fraud patterns and tactics.
- User feedback integration can help refine predictions. Allowing users to confirm or correct fraud predictions would support continuous model improvement.
- The existing Streamlit interface can be further improved with features such as batch input via file upload, transaction history viewing, and downloadable summary reports.
- The model can be integrated into real-world banking or payment platforms using APIs, supporting seamless fraud detection within live environments.
- Performance optimization through parameter tuning, cross-validation, and comparison of other resampling techniques can improve accuracy and reduce processing time.
- Finally, the system should be tested and validated on larger and more diverse datasets to ensure it generalizes well across different populations and transaction types.

References

- Kaggle for dataset: Credit Card Fraud Detection
- YouTube (Data Thinkers): 18. Project 13 Credit Card Fraud Detection using ML | Handling Imbalanced Dataset | ML Project - YouTube
- Chat GPT
- Geek for Geeks: Credit Card Fraud Detection – ML | GeeksforGeeks

NOTE : Press (Ctrl + Click) for redirecting on link