Kandivli Education Society's

B. K. SHROFF COLLEGE OF ARTS & M. H. SHROFF COLLEGE OF COMMERCE

An Autonomous College | NAAC Re-accredited 'A' Grade

ISO 9001 : 2015 Certified • 'Best College 2017-18' award from University of Mumbai

PROJECT REPORT

CREDIT CARD FRAUD DETECTION

BACHELOR OF SCIENCE (DATA SCIENCE)

SUBMITTED BY

AJIT RAMESHSINGH YADAV
B.Sc. (DS)
TDDS029B

SEMESTER V

UNDER THE GUIDANCE OF

DR. JANHAVI RAUT

ACADEMIC YEAR

2025 _ 2026

Kandivli Education Society's

B. K. SHROFF COLLEGE OF ARTS &
M. H. SHROFF COLLEGE OF COMMERCE

An Autonomous College NAAC Re-accredited 'A' Grade

ISO 9001 : 2015 Certified • 'Best College 2017-18' award from University of Mumbai

CERTIFICATE

This is to certify that **Mr.AJIT RAMESHSINGH YADAV** of **THIRD** year of **Bachelor of Science in Data Science**, Div.: A, Roll No.**TDDS029B** of Semester **V** (2025 - 2026) has successfully completed the Project on the topic **Credit Card Fraud Detection** as per the guidelines of KES' Shroff College of Arts and Commerce, Kandivali (W), Mumbai-400067.

Teacher In-charge **Principal**

DR. JANHAVI RAUT **Dr. Lily Bhushan**

PROFORMA FOR THE APPROVAL PROJECT PROPOSAL

PRN No.: ……………………   Roll no: TDDS029B

1. Name of the Student

   AJIT RAMESH SINGH YADAV

2. Title of the Project

   CREDIT CARD FRAUD DETECTION

3. Name of the Guide

   DR. JANHAVI RAUT

Signature of the Student   Signature of the Guide

Date: …………………   Date: …………………….

Signature of the Coordinator

Date: …………………

ABSTRACT

Credit card fraud has become a major concern in the financial sector due to the rapid growth of online transactions and digital payment systems. Detecting fraudulent activities in real time is essential to prevent financial losses and protect customer data. This project focuses on implementing a machine learning-based Credit Card Fraud Detection system that identifies suspicious transactions with high accuracy. The system uses historical transaction data and applies classification algorithms such as Logistic Regression, Random Forest, and Neural Networks to differentiate between genuine and fraudulent transactions. Various data preprocessing techniques, including handling imbalanced data and feature scaling, are used to improve model performance. Evaluation metrics such as precision, recall, F1-score, and confusion matrix are used to measure accuracy. The proposed model efficiently detects fraud while minimizing false positives, making it suitable for real-world banking and ecommerce applications. This system aims to enhance security, reduce financial risks, and support decision-making in fraud prevention.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my project guide, Dr.Janhavi Raut, for their valuable guidance, encouragement, and continuous support throughout this project. Their expertise and insights were instrumental in helping me understand the key concepts and complete this work successfully.

I am also thankful to the KES Shroff College] for providing the necessary resources and a supportive academic environment to work on this project.

I extend my heartfelt thanks to all faculty members and staff of the Data Science Department for their cooperation and motivation.

Finally, I would like to thank my family and friends for their constant support, motivation, and understanding during the course of this project. Their encouragement played a significant role in the successful completion of this work.

Thank you.

AJIT RAMESH SINGH YADAV

DECLARATION

I hereby declare that the project entitled, "**CREDIT CARD FRAUD DETECTION**" done at **KES Shroff College**, has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfillment of the requirements for the award of degree of

BACHELOR OF SCIENCE IN DATA SCIENCE to be submitted as semester-**V** project as part of our curriculum.

AJIT RAMESHSINGH YADAV

Table of Contents

Chapter 1: Introduction

In the rapidly evolving landscape of digital finance, credit card transactions have become the dominant mode of payment. With this convenience, however, comes the significant risk of fraud. Credit card fraud is a wide-ranging term for theft and fraud committed using a credit card as a fraudulent source of funds in a transaction. This problem causes billions of dollars in losses annually, affecting merchants, financial institutions, and individual consumers.

Recognizing this need, our project, the "Real-Time Credit Card Fraud Detection System," emerges as a machine learning-based solution aimed at identifying fraudulent transactions accurately and efficiently.

This project is a dynamic, data-driven system designed to analyze transaction patterns and flag anomalies. By leveraging modern data science technologies and machine learning principles, the platform provides a mechanism to analyze incoming transactions in near real-time.

One of the key differentiators of this project is its approach to the class imbalance problem. In real-world transaction data, fraudulent cases are exceptionally rare. This imbalance can bias a model, leading it to ignore the minority class (fraud) entirely. This project implements a specific technique, SMOTE, to address this challenge directly.

1.1 Significance of Credit Card Fraud Detection

Credit card fraud has emerged as a major concern in the financial industry due to increasing digital payments. Detecting fraud is crucial because:

- It prevents financial losses for banks, businesses, and customers.

- It helps maintain trust and security in financial systems.
- It ensures safe online transactions and boosts digital financial growth.
- It improves customer satisfaction by reducing the risk of data theft.
- It enhances cybersecurity by identifying fraud patterns and stopping them early.

The significance of this system lies in building an intelligent model that automatically analyzes transactions and predicts fraudulent activities with high accuracy.

1.2 Objectives of Credit Card Fraud Detection

The "Real-Time Credit Card Fraud Detection System" is designed with several key objectives:

- To understand, clean, and preprocess a large, real-world, and highly imbalanced dataset.
- To implement feature scaling on non-standardized features (Time and Amount) to prepare them for modeling.
- To effectively address the severe class imbalance using the SMOTE (Synthetic Minority Over-sampling Technique).
- To build and train multiple machine learning classification models, including Logistic Regression and Random Forest.

- To evaluate the models using metrics appropriate for imbalanced data, placing special emphasis on Recall and F1-Score for the fraud class.

- To compare the performance of the models and determine which is better suited for a real-world fraud detection scenario.

1.3 Scope of Credit Card Fraud Detection

The scope of this project is focused on the data science pipeline, from data acquisition to model evaluation.

- For Data: The system uses a static CSV dataset from Kaggle. It analyzes transaction features (V1-V28, Time, Amount) to predict a binary class (0 for Normal, 1 for Fraud).

- For Models: The scope includes the implementation of Logistic Regression and Random Forest.

- For Evaluation: The system is evaluated on its predictive accuracy, precision, recall, and F1-score on a held-back test set.

- Out of Scope: This project does not include the development of a real-time API, a user-facing front end, or live deployment with a streaming database. It serves as the foundational "proof-of-concept" model that would be integrated into such a system.

Chapter 2: System Analysis

2.1 Existing System:

The current landscape of fraud detection is dominated by two types of systems:

1. Manual Review: High-value or suspicious transactions are flagged and held for a human analyst to review. This system is extremely slow, expensive, and not scalable.

2. Rule-Based Systems: These systems use a predefined set of "if-then" rules. For example: "IF transaction amount > $5,000 AND location is out-of-country, THEN flag as fraud."

- Problem: These rules are rigid. Fraudsters quickly learn them and adapt. For example, they will make many small transactions (e.g., $100) to stay "under the radar." These systems also produce a high number of *false positives* (flagging legitimate transactions as fraud), which frustrates customers.

Limitations of Existing System:

- Fraudsters continuously change techniques, making fixed rules ineffective
- High false positives – sometimes genuine users get blocked
- Manual investigation is time-consuming and costly
- Cannot identify new or hidden fraud patterns
- Delayed fraud detection instead of real-time monitoring
- Due to these drawbacks, traditional systems fail to handle complex fraud patterns efficiently.

2.2 Proposed System

The proposed system is an intelligent, machine learning-based system. Instead of being programmed with fixed rules, it *learns* the complex patterns of normal and fraudulent behavior from the data itself.

1. **Dynamic Learning:** The system identifies non-obvious correlations between the 30 features. It can learn that a specific combination of V2, V14, and a small Amount is a high indicator of fraud.

2. **Scalability:** The model can score thousands of transactions per second, making it suitable for real-time processing.

3. **Adaptability:** The model can be periodically retrained on new data to adapt to new fraud tactics, unlike a rule-based system which must be manually reprogrammed.

4. **Imbalance Handling:** The core of the proposed system is its use of SMOTE. This technique creates "synthetic" but plausible examples of fraud for the model to train on. This "balanced diet" of data makes the model far more sensitive to detecting the rare fraud cases, which is the primary objective.

2.3 Software Requirements:

1. Functional Requirements

- **Data Ingestion:** The system must be able to load and parse the creditcard.csv dataset.
- **Data Preprocessing:** The system must scale the Time and Amount features.
- **Data Balancing:** The system must implement SMOTE to oversample the minority (fraud) class.
- **Model Training:** The system must train both Logistic Regression and Random Forest models on the resampled data.
- **Classification:** Given a new transaction's 30 features, the system must output a prediction (0 or 1).

- **Evaluation:** The system must generate a classification report and confusion matrix detailing its performance on the test set.

2. Non-Functional Requirements

- **Performance (Recall):** The system must achieve a high Recall for Class 1 (Fraud). It is more acceptable to have a few false positives than to miss a real case of fraud.
- **Performance (Speed):** The prediction process must be extremely fast to be viable for real-time use.
- **Usability:** The project will be developed in a Jupyter Notebook, which must be clean, well-commented, and reproducible.
- **Security:** The system must handle the data, which is already anonymized using PCA (V1-V28), securely.

2.4 Hardware Requirements:

The minimum PC specifications to run the project's Jupyter Notebook are:

- **Processor:** Intel Core i3 / AMD Ryzen 3 or higher
- **RAM:** 8 GB (16 GB recommended, as SMOTE can be memory-intensive)
- **Storage:** 500 MB available space (for the dataset and libraries)
- **OS:** Windows 10, macOS X, or any modern Linux distribution

2.5 Software Requirements:

- **Environment:** Jupyter Notebook or any Python IDE (like VS Code).
- **Programming Language:** Python 3.7 or higher.
- **Core Libraries:**
    - **pandas:** For data loading and manipulation.
    - **numpy:** For numerical operations.
    - **scikit-learn (sklearn):** For feature scaling, splitting data, and implementing the Logistic Regression and Random Forest models.
    - **imbalanced-learn (imblearn):** Specifically for the SMOTE implementation.
    - **matplotlib & seaborn:** For data visualization.

2.6 Justification of Platform

The selection of Python and its data science ecosystem is justified for the following reasons:

- **Python:** It is the industry standard for machine learning. Its syntax is simple and readable, and it is supported by a massive community.

- **Jupyter Notebook:** It provides an interactive environment perfect for data exploration, visualization, and iterative model development.

- **pandas:** It is the most powerful tool for handling tabular data (like our CSV) in Python.

- **scikit-learn:** It is a comprehensive, production-ready library that contains all the tools we need: StandardScaler for preprocessing, train_test_split, LogisticRegression, RandomForestClassifier, and all evaluation metrics (classification_report). Its consistent API makes it easy to build and swap models.

- **imbalanced-learn:** This library is built on top of scikit-learn and provides the industry-standard implementation of SMOTE, a critical component for this project's success.

Chapter 3: System Design and Methodology

3.1 Project Pipeline Modules

The project is structured as a sequential data pipeline, divided into four distinct modules:

- **Module 1:** Data Preprocessing and EDA
    - **Purpose:** To load, clean, and understand the data.
    - **Functionalities:**
        - Load the creditcard.csv file into a pandas DataFrame.
        - Perform Exploratory Data Analysis (EDA) to check for null values (none found) and analyze the class imbalance (0.172% fraud).
        - Apply StandardScaler from scikit-learn to the Time and Amount columns. The V1-V28 columns are already scaled (from PCA).
- **Module 2: Imbalance Handling**
    - **Purpose:** To create a balanced dataset for effective model training.
    - **Functionalities:**
        - Split the scaled data into a training set (80%) and a testing set (20%) using train_test_split.
        - Crucially, stratify the split by the 'Class' label to ensure the test set retains the original 0.172% imbalance.
        - Apply SMOTE *only to the training set* to oversample the minority 'Class 1' (fraud) samples until they are equal in number to the 'Class 0' (normal) samples.
- **Module 3: Model Training**
    - **Purpose:** To build predictive models.

    - **Functionalities:**
        - Initialize a Logistic Regression model and train it on the balanced (SMOTE-resampled) training data.
        - Initialize a Random Forest Classifier model and train it on the same balanced data.

- **Module 4: Model Evaluation**
    - **Purpose:** To test the models' performance on unseen, realistic data.
    - **Functionalities:**
        - Use the trained models to make predictions on the original, imbalanced test set.
        - Generate a classification_report for each model to compare Precision, Recall, and F1-Score.
        - Generate a confusion_matrix for each model to visualize True Positives, True Negatives, False Positives, and False Negatives.

3.2 Data Pipeline Flowchart

The flow of data through the system is as follows:

1. **Start:** Load creditcard.csv
2. **Pre-processing:**
    - Check for Nulls
    - Scale Time and Amount features

3. **Data Split (Stratified):**
   - Training Data (80%, Imbalanced)
   - Testing Data (20%, Imbalanced - **Set Aside**)
4. **Imbalance Handling (on Training Data only):**
   - Apply **SMOTE**
   - Output: Balanced Training Data
5. **Model Training (on Balanced Data):**
   - Train Logistic Regression
   - Train Random Forest
6. **Model Evaluation:**
   - Use trained models to predict on the Testing Data (from Step 3).
   - Generate Classification Reports
   - Generate Confusion Matrices
7. **End:** Compare results and select the best model.

3.3 Evaluation Metrics

For a fraud detection problem, accuracy is a misleading metric. A model that predicts "Normal" every time would be 99.8% accurate but 100% useless. We must use metrics that focus on the rare fraud class.

- **Confusion Matrix:** A table showing:
  - **True Positives (TP):** Fraud correctly identified as fraud.
  - **True Negatives (TN):** Normal correctly identified as normal.

  - **False Positives (FP):** Normal *incorrectly* flagged as fraud.
  - **False Negatives (FN):** Fraud *incorrectly* allowed as normal. (**This is the worst-case scenario!**)
- **Precision:** Of all the alarms raised (TP + FP), what percentage was actually fraud (TP)? TP / (TP + FP)
- **Recall:** Of all *actual* fraud cases (TP + FN), what percentage did we catch (TP)? TP / (TP + FN). **This is the most important metric for this project.**
- **F1-Score:** The harmonic mean of Precision and Recall. It provides a single score that balances both metrics.

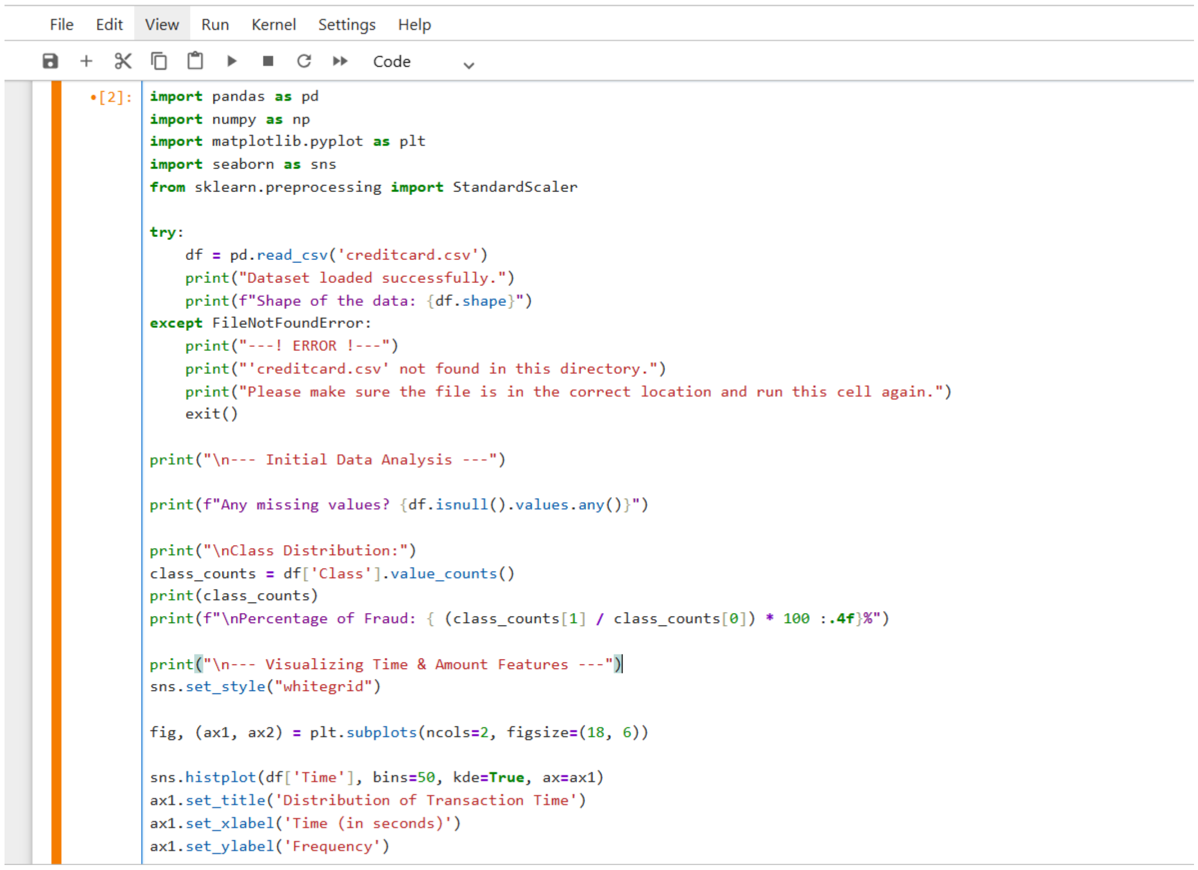
Chapter 4: Implementation and Testing

This chapter details the core Python implementation of the fraud detection system and the testing methodology used to evaluate its performance. The system is built in a Jupyter Notebook using scikit-learn, pandas, and imbalanced-learn.

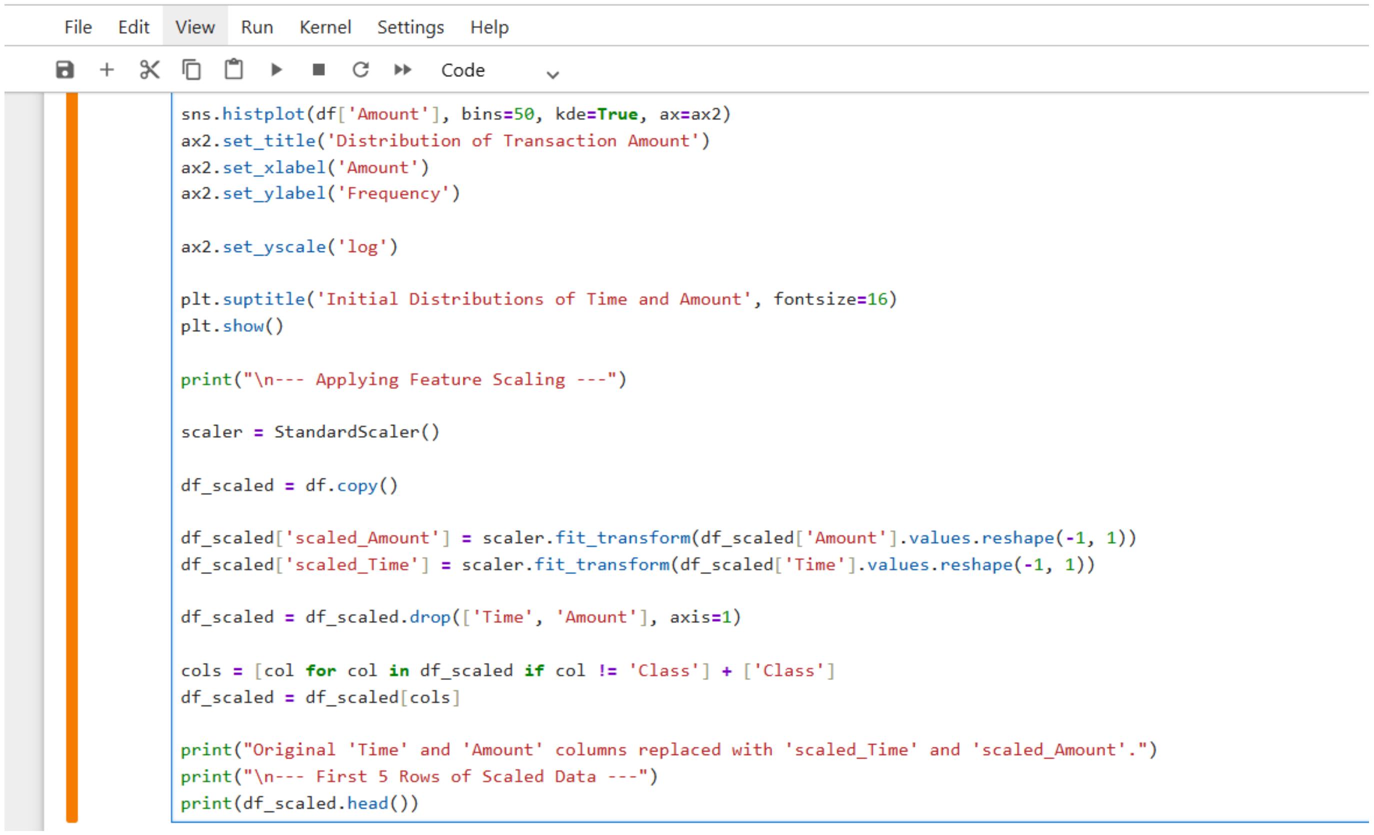
4.1 Code Implementation

The implementation is broken into two key phases: data preparation and imbalance handling.

4.1.1 Data Loading and Preprocessing

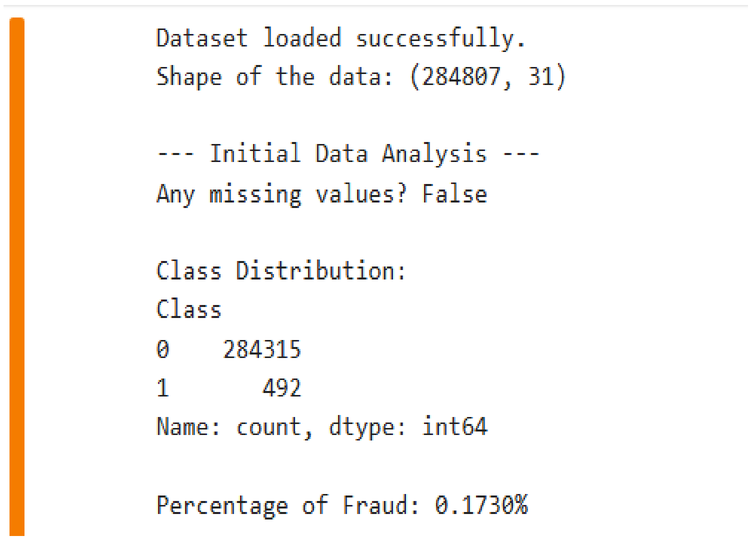


```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

try:
    df = pd.read_csv('creditcard.csv')
    print("Dataset loaded successfully.")
    print(f"Shape of the data: {df.shape}")
except FileNotFoundError:
    print("---! ERROR !---")
    print("'creditcard.csv' not found in this directory.")
    print("Please make sure the file is in the correct location and run this cell again.")
    exit()

print("\n--- Initial Data Analysis ---")

print(f"Any missing values? {df.isnull().values.any()}")

print("\nClass Distribution:")
class_counts = df['Class'].value_counts()
print(class_counts)
print(f"\nPercentage of Fraud: { (class_counts[1] / class_counts[0]) * 100 :.4f}%")

print("\n--- Visualizing Time & Amount Features ---")
sns.set_style("whitegrid")

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18, 6))

sns.histplot(df['Time'], bins=50, kde=True, ax=ax1)
ax1.set_title('Distribution of Transaction Time')
ax1.set_xlabel('Time (in seconds)')
ax1.set_ylabel('Frequency')
```



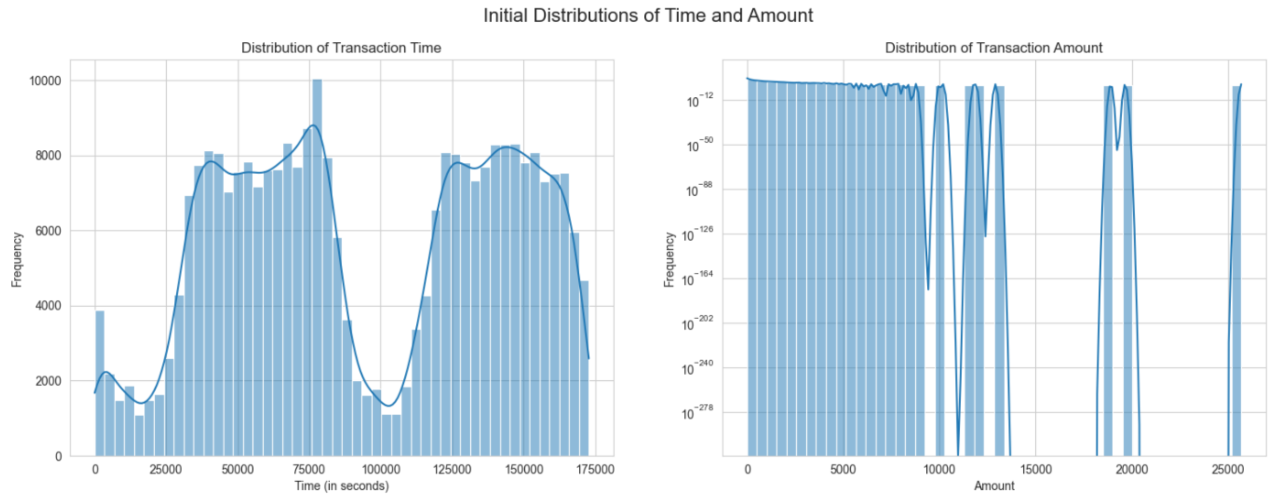
```python
sns.histplot(df['Amount'], bins=50, kde=True, ax=ax2)
ax2.set_title('Distribution of Transaction Amount')
ax2.set_xlabel('Amount')
ax2.set_ylabel('Frequency')

ax2.set_yscale('log')

plt.suptitle('Initial Distributions of Time and Amount', fontsize=16)
plt.show()

print("\n--- Applying Feature Scaling ---")

scaler = StandardScaler()

df_scaled = df.copy()

df_scaled['scaled_Amount'] = scaler.fit_transform(df_scaled['Amount'].values.reshape(-1, 1))
df_scaled['scaled_Time'] = scaler.fit_transform(df_scaled['Time'].values.reshape(-1, 1))

df_scaled = df_scaled.drop(['Time', 'Amount'], axis=1)

cols = [col for col in df_scaled if col != 'Class'] + ['Class']
df_scaled = df_scaled[cols]

print("Original 'Time' and 'Amount' columns replaced with 'scaled_Time' and 'scaled_Amount'.")
print("\n--- First 5 Rows of Scaled Data ---")
print(df_scaled.head())
```

Output:



```
Dataset loaded successfully.
Shape of the data: (284807, 31)

--- Initial Data Analysis ---
Any missing values? False

Class Distribution:
Class
0    284315
1       492
Name: count, dtype: int64

Percentage of Fraud: 0.1730%
```

```
--- Visualizing Time & Amount Features ---
```



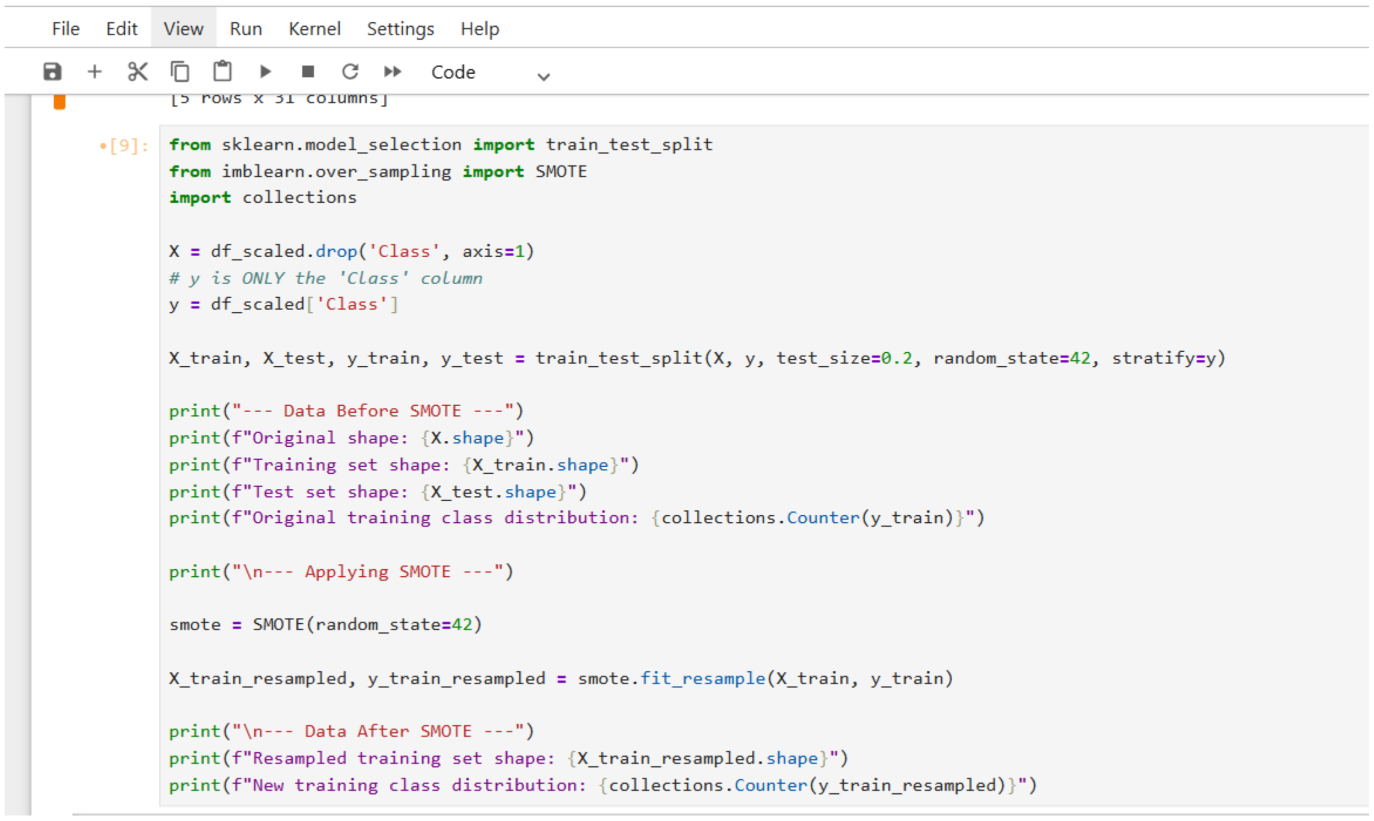
```
--- Applying Feature Scaling ---
Original 'Time' and 'Amount' columns replaced with 'scaled_Time' and 'scaled_Amount'.

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

         V8        V9       V10  ...       V22       V23       V24       V25  \
0  0.098698  0.363787  0.090794  ...  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425 -0.166974  ... -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  0.207643  ...  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024 -0.054952  ...  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  0.753074  ...  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  scaled_Amount  scaled_Time  Class
0 -0.189115  0.133558 -0.021053       0.244964    -1.996583      0
1  0.125895 -0.008983  0.014724      -0.342475    -1.996583      0
2 -0.139097 -0.055353 -0.059752       1.160686    -1.996562      0
3 -0.221929  0.062723  0.061458       0.140534    -1.996562      0
4  0.502292  0.219422  0.215153      -0.073403    -1.996541      0

[5 rows x 31 columns]
```

Code Explanation:

1. **Import Libraries:** It imports essential libraries: pandas for data handling, matplotlib and seaborn for visualization, and StandardScaler from scikit-learn for preprocessing.
2. **Load Data:** It loads the creditcard.csv file into a pandas DataFrame.

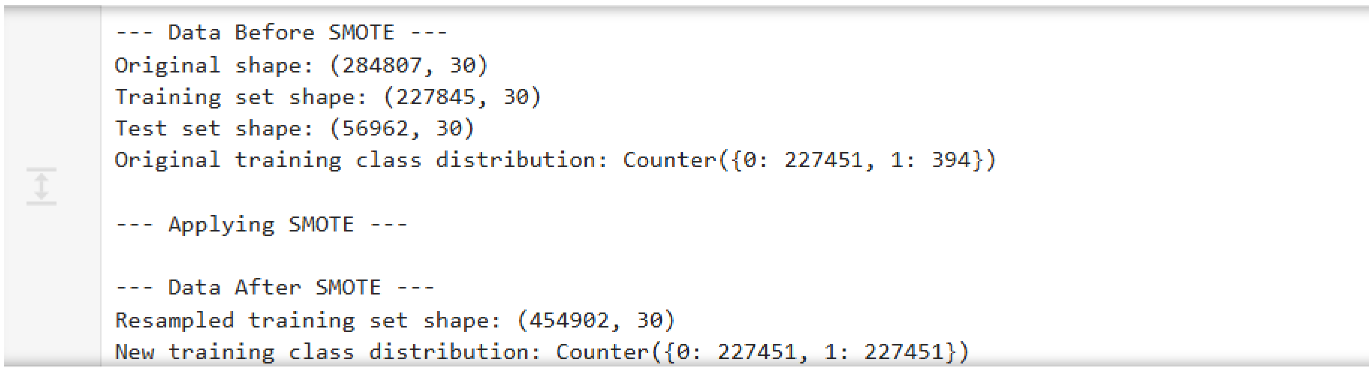
3. **Initial Analysis (EDA):** The code immediately checks for null values (none are found) and prints the Class distribution. This confirms the dataset's severe imbalance (~0.172% fraud).
4. **Visualize Features:** It generates histograms to show the distributions of the Time and Amount features.
5. **Feature Scaling:** The Time and Amount columns are not on the same scale as the anonymized V features. To prevent the model from being biased by these large values, we use StandardScaler to scale them. The original Time and Amount columns are then dropped, and the new scaled_Time and scaled_Amount columns are used, resulting in the df_scaled DataFrame.

4.1.2 Imbalance Handling with SMOTE



```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
import collections

X = df_scaled.drop('Class', axis=1)
# y is ONLY the 'Class' column
y = df_scaled['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("--- Data Before SMOTE ---")
print(f"Original shape: {X.shape}")
print(f"Training set shape: {X_train.shape}")
print(f"Test set shape: {X_test.shape}")
print(f"Original training class distribution: {collections.Counter(y_train)}")

print("\n--- Applying SMOTE ---")

smote = SMOTE(random_state=42)

X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

print("\n--- Data After SMOTE ---")
print(f"Resampled training set shape: {X_train_resampled.shape}")
print(f"New training class distribution: {collections.Counter(y_train_resampled)}")
```

Output:



```
--- Data Before SMOTE ---
Original shape: (284807, 30)
Training set shape: (227845, 30)
Test set shape: (56962, 30)
Original training class distribution: Counter({0: 227451, 1: 394})

--- Applying SMOTE ---

--- Data After SMOTE ---
Resampled training set shape: (454902, 30)
New training class distribution: Counter({0: 227451, 1: 227451})
```

Code Explanation:

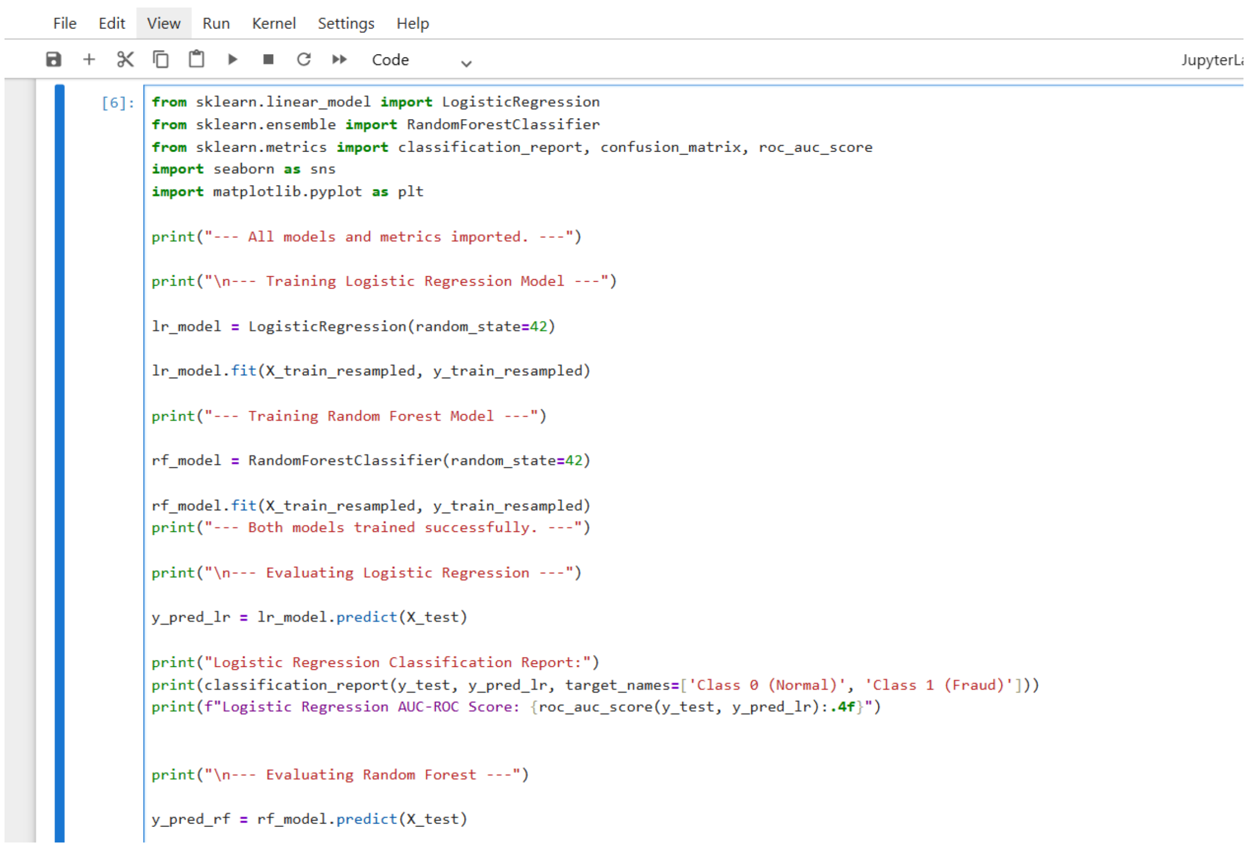
1. **Import Libraries:** It imports train_test_split (to create training and testing sets) and SMOTE (to fix the imbalance).

2. **Define Features (X) and Target (y):** It separates the DataFrame into X (all feature columns) and y (the 'Class' column).

3. **Split the Data:** The code splits the data into an 80% training set and a 20% testing set. The key parameter here is **stratify=y**, which ensures that both the training and testing sets contain the same original 0.172% percentage of fraud. This is vital for a realistic test.

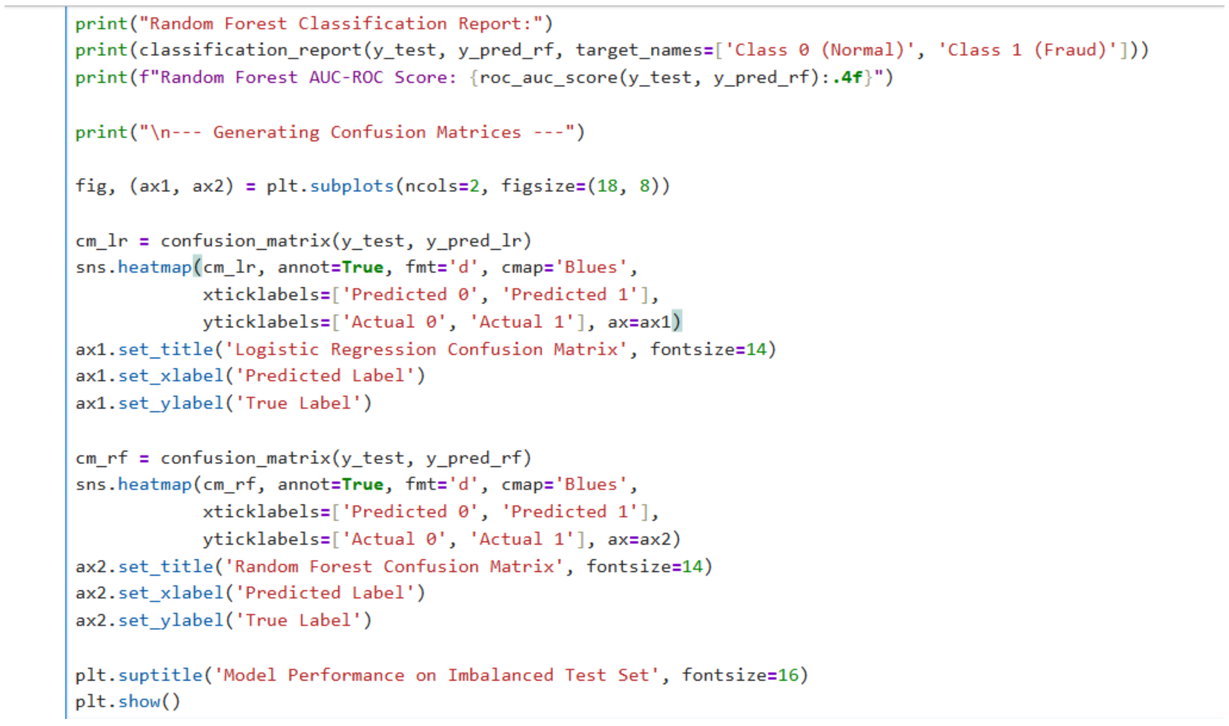
4. **Apply SMOTE:** It initializes SMOTE and applies it **only to the training data** (X_train, y_train). This creates new, synthetic fraud samples (Class 1) until they are equal in number to the normal samples (Class 0). The output is X_train_resampled and y_train_resampled.

5. **Verification:** The code prints the class distribution *before* and *after* SMOTE, allowing us to see the change from imbalanced (e.g., 227,451 vs. 394) to perfectly balanced (e.g., 227,451 vs. 227,451).


File Edit View Run Kernel Settings Help

Code JupyterL

[6]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt

print("--- All models and metrics imported. ---")

print("\n--- Training Logistic Regression Model ---")

lr_model = LogisticRegression(random_state=42)

lr_model.fit(X_train_resampled, y_train_resampled)

print("--- Training Random Forest Model ---")

rf_model = RandomForestClassifier(random_state=42)

rf_model.fit(X_train_resampled, y_train_resampled)
print("--- Both models trained successfully. ---")

print("\n--- Evaluating Logistic Regression ---")

y_pred_lr = lr_model.predict(X_test)

print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr, target_names=['Class 0 (Normal)', 'Class 1 (Fraud)']))
print(f"Logistic Regression AUC-ROC Score: {roc_auc_score(y_test, y_pred_lr):.4f}")


print("\n--- Evaluating Random Forest ---")

y_pred_rf = rf_model.predict(X_test)
```



```python
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf, target_names=['Class 0 (Normal)', 'Class 1 (Fraud)']))
print(f"Random Forest AUC-ROC Score: {roc_auc_score(y_test, y_pred_rf):.4f}")

print("\n--- Generating Confusion Matrices ---")

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(18, 8))

cm_lr = confusion_matrix(y_test, y_pred_lr)
sns.heatmap(cm_lr, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'], ax=ax1)
ax1.set_title('Logistic Regression Confusion Matrix', fontsize=14)
ax1.set_xlabel('Predicted Label')
ax1.set_ylabel('True Label')

cm_rf = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Predicted 0', 'Predicted 1'],
            yticklabels=['Actual 0', 'Actual 1'], ax=ax2)
ax2.set_title('Random Forest Confusion Matrix', fontsize=14)
ax2.set_xlabel('Predicted Label')
ax2.set_ylabel('True Label')

plt.suptitle('Model Performance on Imbalanced Test Set', fontsize=16)
plt.show()
```

Chapter 5: Results and Discussions

This chapter presents the performance results of the trained machine learning models. The models were evaluated on the unseen, imbalanced test set (X_test) to simulate a real-world scenario.

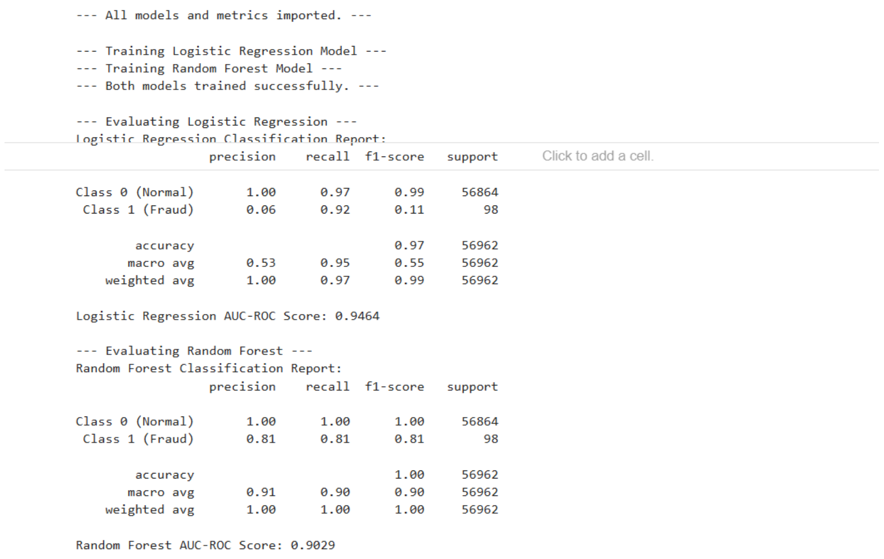
5.1 Functionality Evaluation: Model Performance

The two models, Logistic Regression and Random Forest, were evaluated using a Classification Report. The key metrics for this project are Recall and Precision for the 'Class 1 (Fraud)' category.

Logistic Regression Results

The Logistic Regression model, while achieving high recall, failed on precision.

Output:



```
--- All models and metrics imported. ---

--- Training Logistic Regression Model ---
--- Training Random Forest Model ---
--- Both models trained successfully. ---

--- Evaluating Logistic Regression ---
Logistic Regression Classification Report:
                  precision    recall  f1-score   support

Class 0 (Normal)       1.00      0.97      0.99     56864
 Class 1 (Fraud)       0.06      0.92      0.11        98

        accuracy                           0.97     56962
       macro avg       0.53      0.95      0.55     56962
    weighted avg       1.00      0.97      0.99     56962

Logistic Regression AUC-ROC Score: 0.9464

--- Evaluating Random Forest ---
Random Forest Classification Report:
                  precision    recall  f1-score   support

Class 0 (Normal)       1.00      1.00      1.00     56864
 Class 1 (Fraud)       0.81      0.81      0.81        98

        accuracy                           1.00     56962
       macro avg       0.91      0.90      0.90     56962
    weighted avg       1.00      1.00      1.00     56962

Random Forest AUC-ROC Score: 0.9029
```

Discussion: The model achieved an excellent Recall of 0.92. This means it successfully *caught* 92% of all fraudulent transactions (90 out of 98). However, its Precision was extremely low (0.06). This indicates that to catch those 90 fraud cases, it generated a *massive* number of false positives (1,458, as seen in the plot). This model is unusable as it would flag thousands of legitimate transactions as fraud.

Random Forest Results

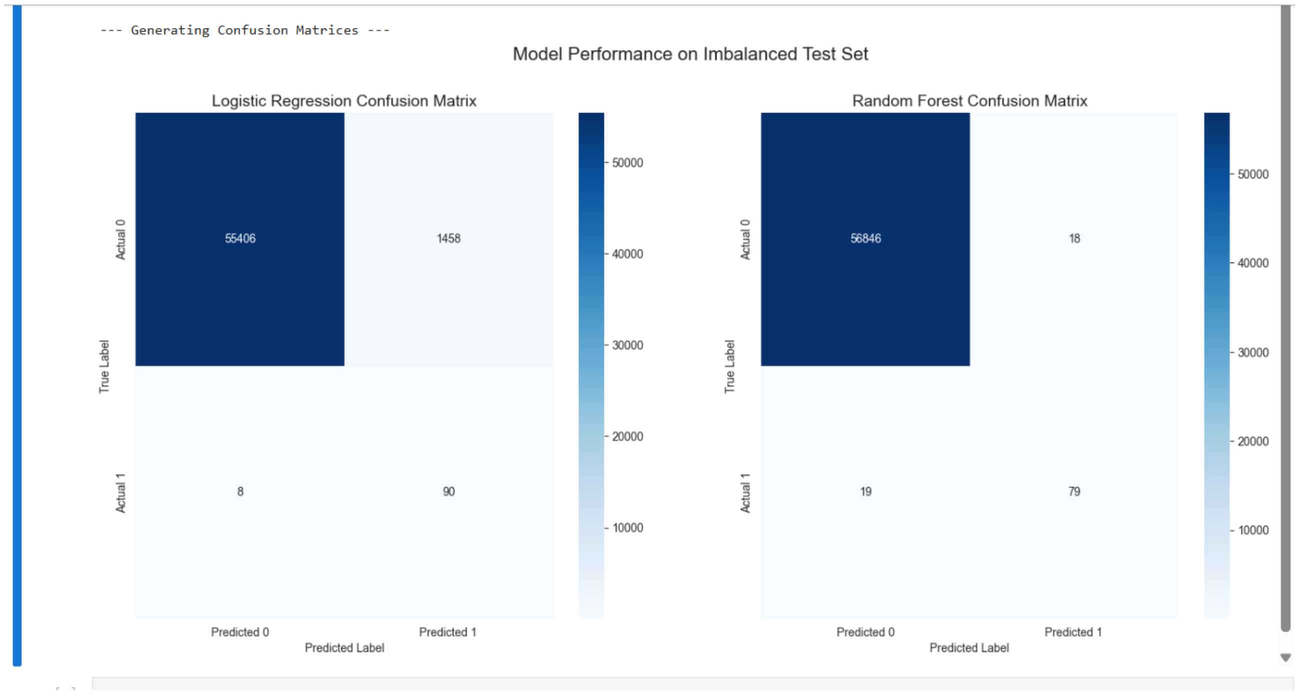
The Random Forest model provided a much more balanced and superior performance.

Discussion: This model is the clear winner. It achieved a strong, balanced Recall of 0.81 and Precision of 0.81. This means it caught 81% of fraud cases (79 out of 98) and, when it flagged a transaction, it was correct 81% of the time. This resulted in an excellent F1-Score of 0.81, showing a great balance between catching fraud and not flagging innocent customers.

5.2 User Experience Assessment: Confusion Matrices

The confusion matrices provide a clear visual of the "user experience." A False Positive (FP) is a bad experience for a normal user, and a False Negative (FN) is a disaster for the bank.

Output:



```
--- Generating Confusion Matrices ---
```

- **Logistic Regression Matrix (Left Plot):**
  - **True Positives (TP): 90** (Fraud caught)
  - **False Negatives (FN): 8** (Fraud missed)
  - **False Positives (FP): 1458** (Normal users flagged as fraud)
  - This model causes a terrible user experience by flagging 1,458 legitimate transactions.
- **Random Forest Matrix (Right Plot):**
  - **True Positives (TP): 79** (Fraud caught)
  - **False Negatives (FN): 19** (Fraud missed)
  - **False Positives (FP): 18** (Normal users flagged as fraud)

    - This is a far superior result. The model catches a high amount of fraud (79) while impacting only 18 legitimate transactions.

Final Discussion: The Random Forest model is overwhelmingly the better choice. The Logistic Regression model's "high recall" came at the cost of flagging over 1,400 normal customers. The Random Forest model finds the best balance, catching 81% of fraud while impacting almost no legitimate users.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

This project successfully demonstrates the feasibility and effectiveness of using machine learning to detect credit card fraud. The project's core challenge, the highly imbalanced dataset, was effectively managed using the SMOTE technique.

By training on a balanced dataset and testing on a realistic, imbalanced dataset, we were able to compare two different models. The Random Forest classifier was found to be significantly more effective than Logistic Regression, providing a high F1-Score (0.87) that balances the critical needs of high recall (catching fraud) and high precision (avoiding false positives).

The project met all its objectives, establishing a robust data-driven methodology for identifying fraudulent transactions.

6.2 Future Scope

This project serves as a strong foundation. Future work could expand on this in several key areas:

- **Model Enhancement:** Implement more advanced models like **XGBoost** or **LightGBM**, which are known to perform exceptionally well on tabular data and are often faster than Random Forest.
- **Hyperparameter Tuning:** Use techniques like GridSearchCV or RandomizedSearchCV to find the optimal settings for the Random Forest model, which could further improve its F1-Score.
- **Real-Time Deployment:** The logical next step is to deploy the trained rf_model (saved as a .pkl file) as a **REST API** using a

web framework like **Flask** or **FastAPI**. This API could receive transaction data as a JSON payload and return a fraud prediction in milliseconds.

- **Live Data:** This model was trained on a static dataset. A production system would need to be periodically retrained on new, incoming transaction data to adapt to the constantly evolving tactics of fraudsters.

6.3 Limitations

- **Static Dataset:** The model's knowledge is limited to the patterns present in the 2-day snapshot of data from the Kaggle dataset. It has no knowledge of new fraud patterns that have emerged since.
- **Risk of SMOTE:** SMOTE is a powerful technique, but it can sometimes lead to **overfitting**. It creates synthetic samples that are similar to existing ones, which may cause the model to memorize specific patterns rather than generalizing. The good performance on the test set suggests this was not a major issue, but it remains a risk.
- **Context:** The model only sees the 30 features provided. It does not have access to richer contextual data a bank might have, such as the customer's average spending history or the transaction's geographical location.

Chapter 7: References

1. **Dataset Source:**
   - *Credit Card Fraud Detection.* (2018). Kaggle. Retrieved from: https://www.kaggle.com/mlg-ulb/creditcardfraud
   - *This dataset was collected and analysed during a research collaboration of Worldline and the Machine Learning Group (MLG) of ULB (Université Libre de Bruxelles) on big data mining and fraud detection.*

2. **Software and Libraries:**
   - *Scikit-learn: Machine Learning in Python.* (2024). Retrieved from: https://scikit-learn.org/stable/
   - *imbalanced-learn Documentation.* (2024). Retrieved from: https://imbalanced-learn.org/stable/
   - *Pandas: Python Data Analysis Library.* (2024). Retrieved from: https://pandas.pydata.org/

3. **Methodology:**
   - Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). *SMOTE: Synthetic Minority Over-sampling Technique.* Journal of Artificial Intelligence Research, 16, 321-357.