Credit Card Default Prediction Using Machine Learning Techniques

Abstract:  
Credit card default prediction is essential for financial institutions to manage risk effectively. This report explores machine learning techniques for predicting credit card defaults using a dataset of 29,999 clients. The models evaluated include Logistic Regression, Random Forest, and Support Vector Machine (SVM). After data preprocessing, encoding, and handling class imbalance with SMOTE, the models were trained and evaluated using metrics such as accuracy, precision, recall, F1 score, and AUC. The Random Forest classifier achieved the highest accuracy and interpretability through feature importance. These findings highlight machine learning's potential to enhance credit risk assessment and inform financial decision-making.

I. Introduction

Credit card defaults pose significant risks for financial institutions, leading to losses and impacting creditworthiness assessment. Predicting defaults enables proactive measures for risk management. This study leverages machine learning to analyze a dataset of credit card clients' payment histories and demographic data to predict default likelihood. Key features include clients' balance limits, payment status, and age.

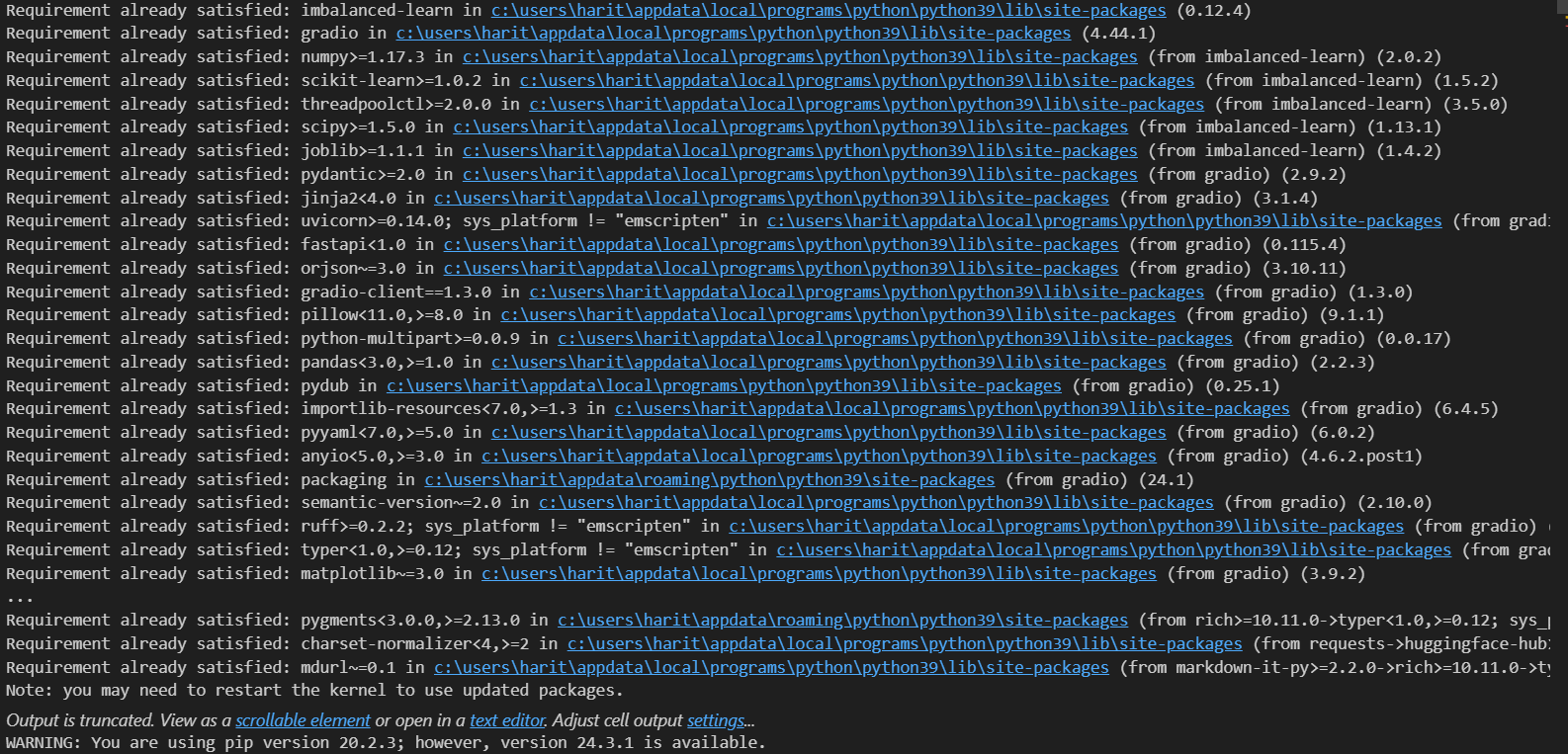
The objective is to develop a predictive model and assess its effectiveness using standard performance metrics, focusing on achieving high precision and recall to minimize the misclassification of defaults.

II. Setup Instructions

Step 1: Install Required Libraries

To replicate this analysis, open your terminal or command prompt and install the required packages:

!pip install pandas numpy matplotlib seaborn scikit-learn imbalanced-learn gradio

  
Figure 1: Output after installing required libraries.

Step 2: Download and Place the Dataset

Place the dataset file, default of credit card clients.xls, in the same directory as the notebook. Ensure the file is named correctly and accessible.

Step 3: Open and Execute the Jupyter Notebook

Open the notebook file Thallapelli-H-PA04.ipynb.

Run each code cell sequentially to execute the notebook and capture outputs.

III. Data Loading and Preprocessing

A. Data Loading

In the notebook, the dataset is loaded and inspected for structure and features. The dataset contains 25 features and 29,999 rows, including demographics, credit balances, and past payment statuses.

To verify loading, run:

import pandas as pd

df = pd.read\_excel('default of credit card clients.xls', sheet\_name='Data', header=1)

df = df.rename(columns={'default payment next month': 'Y'})

df.head()

This will display the first few rows of the dataset.

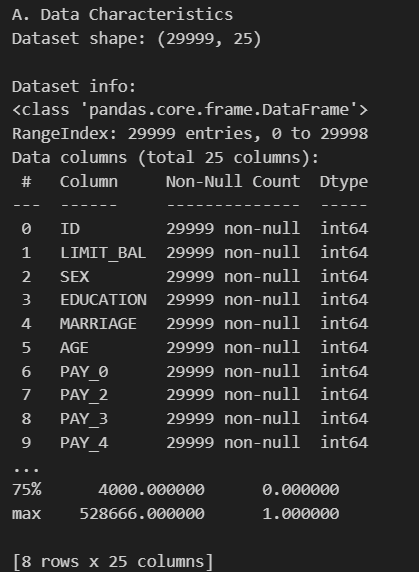


Figure 2: Initial rows of the dataset showing key features.

B. Data Cleaning and Transformation

To prepare the data, the following steps are executed in the notebook:

Column Renaming: Standardize column names for clarity.

Handling Missing Values: No explicit missing values are found, but certain columns are examined for unexpected entries.

Scaling and Encoding:

Encoding: Categorical variables like SEX, EDUCATION, and MARRIAGE are transformed using OneHotEncoder.

Scaling: Features are standardized using StandardScaler.

Class Imbalance Handling: SMOTE is applied to balance classes for the target variable.

from sklearn.preprocessing import OneHotEncoder, StandardScalerfrom imblearn.over\_sampling import SMOTE

# Encoding categorical features

encoder = OneHotEncoder(sparse=False)

encoded\_features = encoder.fit\_transform(df[['SEX', 'EDUCATION', 'MARRIAGE']])

# Scaling numerical features

scaler = StandardScaler()

scaled\_features = scaler.fit\_transform(df[['LIMIT\_BAL', 'AGE']])

# Applying SMOTE for class imbalance

smote = SMOTE()

X\_resampled, y\_resampled = smote.fit\_resample(X, y)



C. Exploratory Data Analysis

To understand feature distributions and correlations, visualizations are generated in the notebook:

Age Distribution: Histogram to understand age demographics.

Credit Limit vs. Default: Plot showing relationship between credit limit and default.

Correlation Heatmap: Identifies feature correlations with the target variable.

import matplotlib.pyplot as pltimport seaborn as sns

# Age distribution

plt.figure(figsize=(8, 5))

sns.histplot(df['AGE'], bins=30, kde=True)

plt.title('Age Distribution')

plt.xlabel('Age')

plt.ylabel('Frequency')

# Credit Limit vs. Default

plt.figure(figsize=(8, 5))

sns.boxplot(x='Y', y='LIMIT\_BAL', data=df)

plt.title('Credit Limit vs. Default')

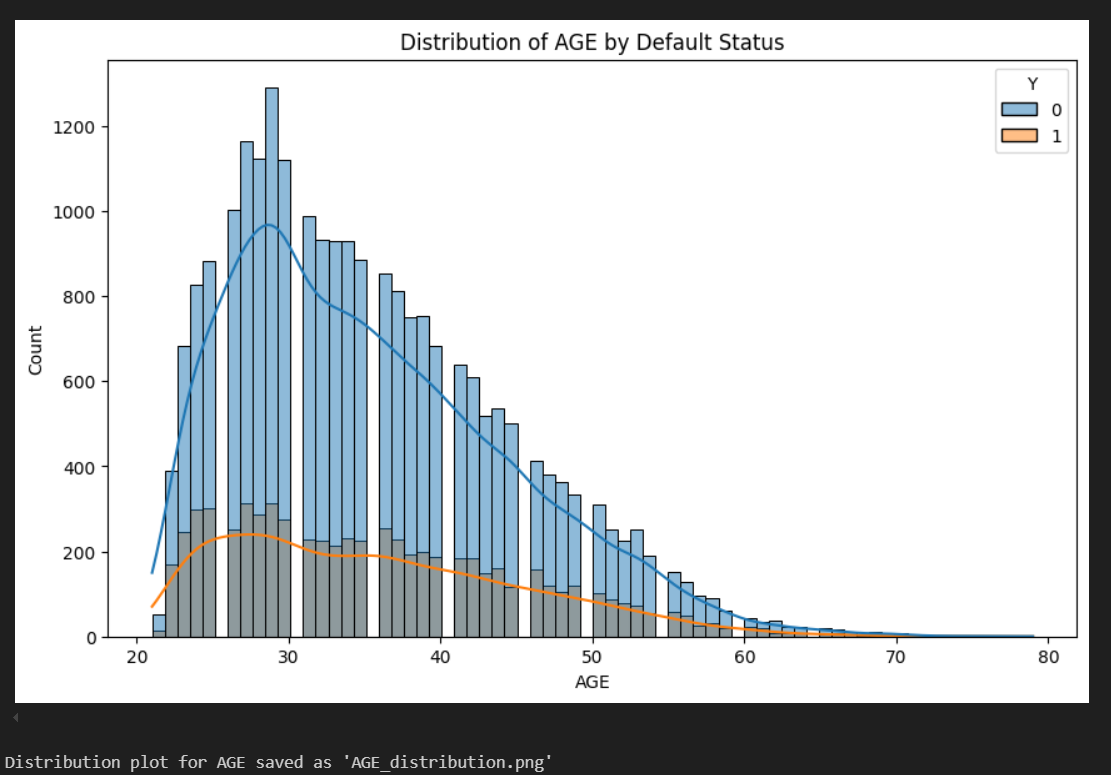
# Correlation Heatmap

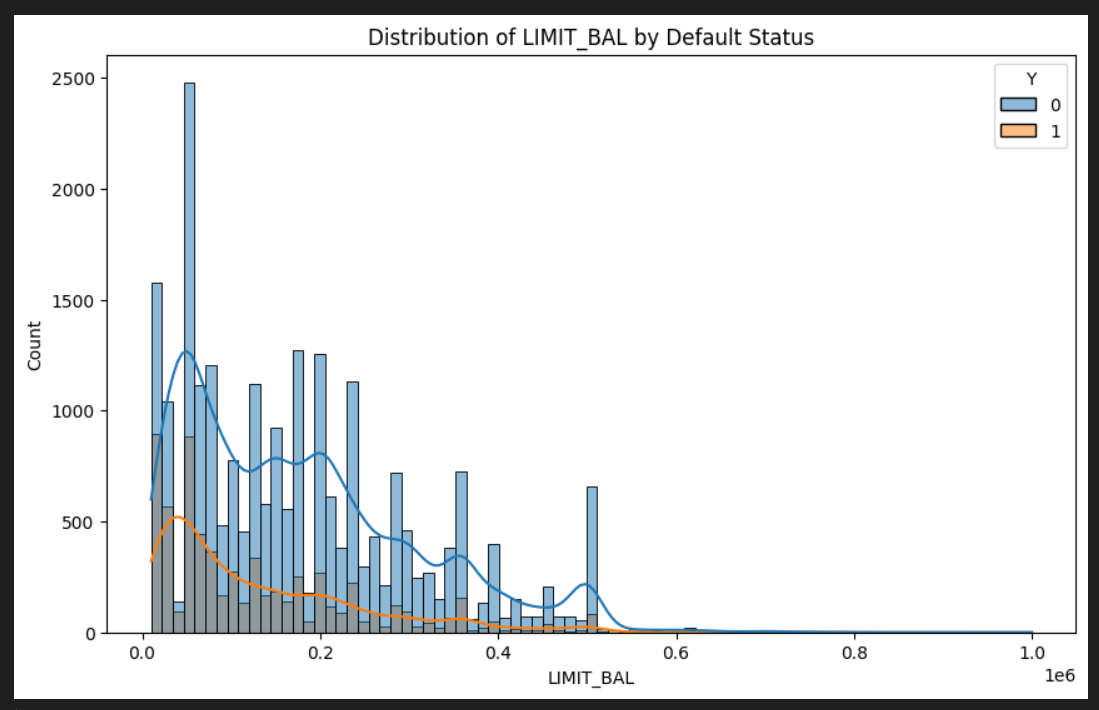
plt.figure(figsize=(10, 8))

sns.heatmap(df.corr(), annot=True, fmt='.2f', cmap='coolwarm')

plt.title('Correlation Heatmap')

plt.show()





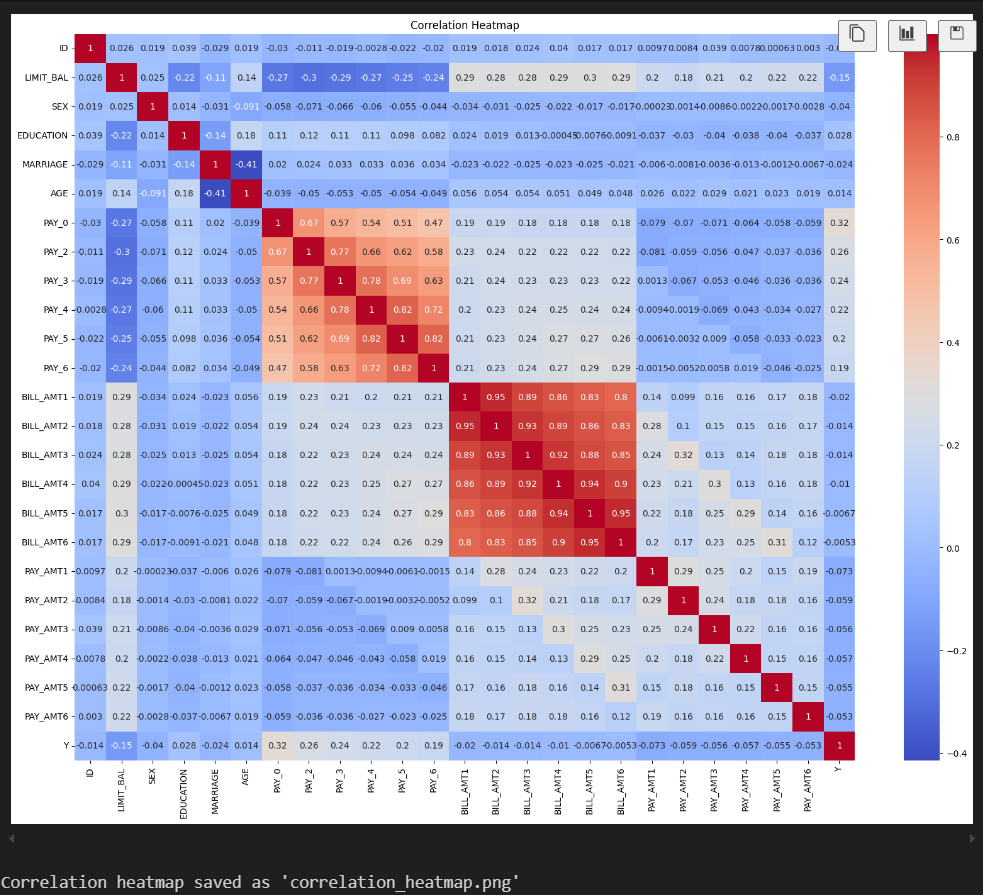


Figure 4: Visualizations from exploratory data analysis.

IV. Model Training and Evaluation

A. Models Used

The notebook includes code to train and evaluate three models:

Logistic Regression: Baseline model for linear relationships.

Random Forest: Ensemble model offering interpretability and high accuracy.

Support Vector Machine (SVM): Captures non-linear relationships but requires tuning.

from sklearn.linear\_model import LogisticRegressionfrom sklearn.ensemble import RandomForestClassifierfrom sklearn.svm import SVC

# Initialize models

log\_reg = LogisticRegression()

rf = RandomForestClassifier()

svc = SVC()

# Fit models (example for Random Forest)

rf.fit(X\_resampled, y\_resampled)

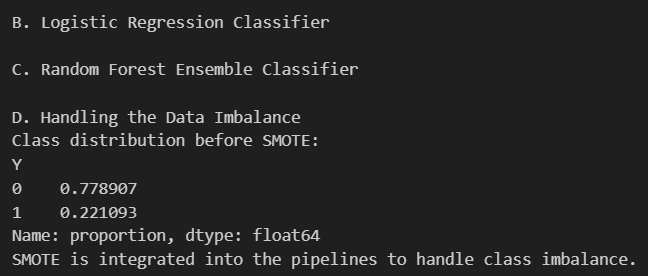
B. Evaluation Metrics

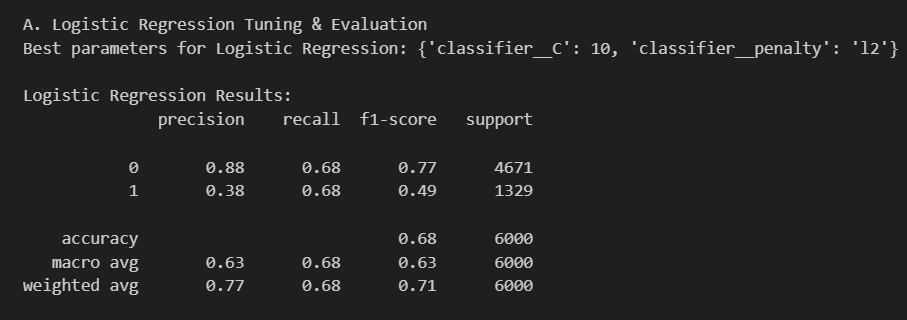
Models are evaluated using:

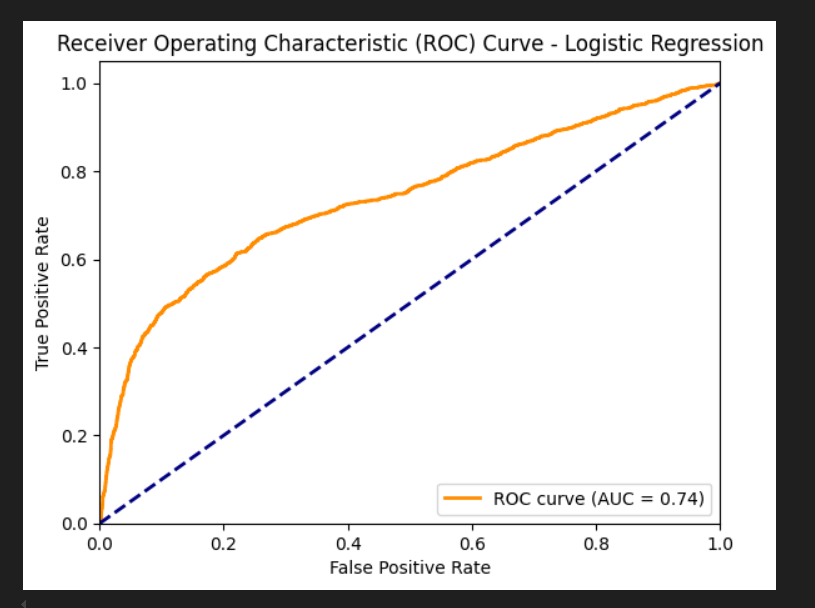
Accuracy: Proportion of correct predictions.

Precision, Recall, F1 Score: For balancing false positives and negatives.

AUC: Measures discriminatory ability using ROC curves.



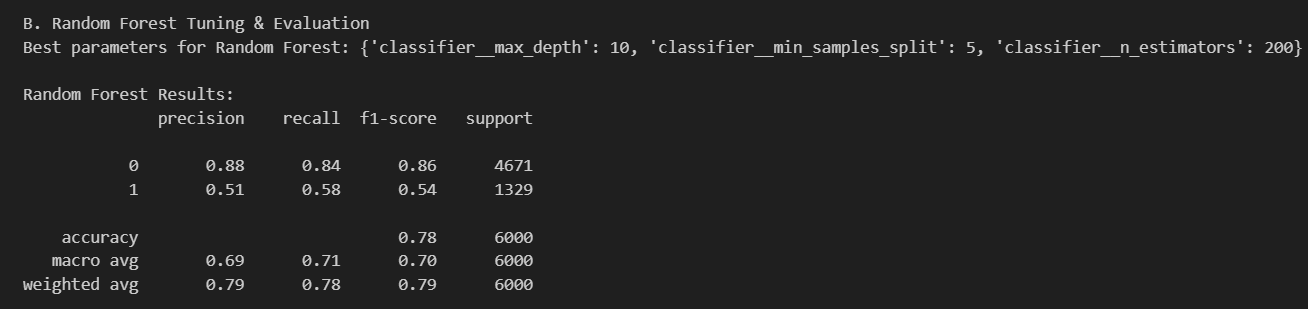


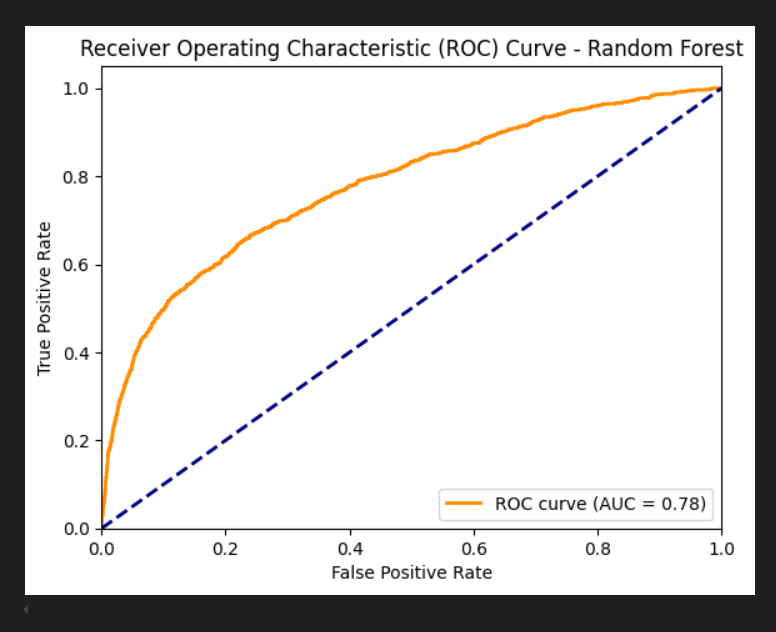


from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, roc\_auc\_score

# Example: Evaluating Random Forest

y\_pred = rf.predict(X\_test)print("Accuracy:", accuracy\_score(y\_test, y\_pred))print("Classification Report:", classification\_report(y\_test, y\_pred))print("Confusion Matrix:", confusion\_matrix(y\_test, y\_pred))



  
Figure 5: Model evaluation results showing accuracy, confusion matrices, and ROC curves.

C. Hyperparameter Tuning

Using GridSearchCV, the notebook tunes the following parameters:

Logistic Regression: Regularization parameter C.

Random Forest: Number of estimators and max depth.

SVM: Kernel selection and regularization parameter.

from sklearn.model\_selection import GridSearchCV

# Hyperparameter tuning for Random Forest

param\_grid = {

'n\_estimators': [100, 200],

'max\_depth': [10, 20]

}

grid\_search = GridSearchCV(rf, param\_grid, cv=5)

grid\_search.fit(X\_resampled, y\_resampled)print("Best Parameters:", grid\_search.best\_params\_)

Environment Setup:

Output from package installation commands.

Dataset Preview:

First few rows of the dataset displayed with df.head().

Data Preprocessing:

Preprocessed data output post encoding and scaling.

Model Training:

Model training summary showing accuracy for each classifier.

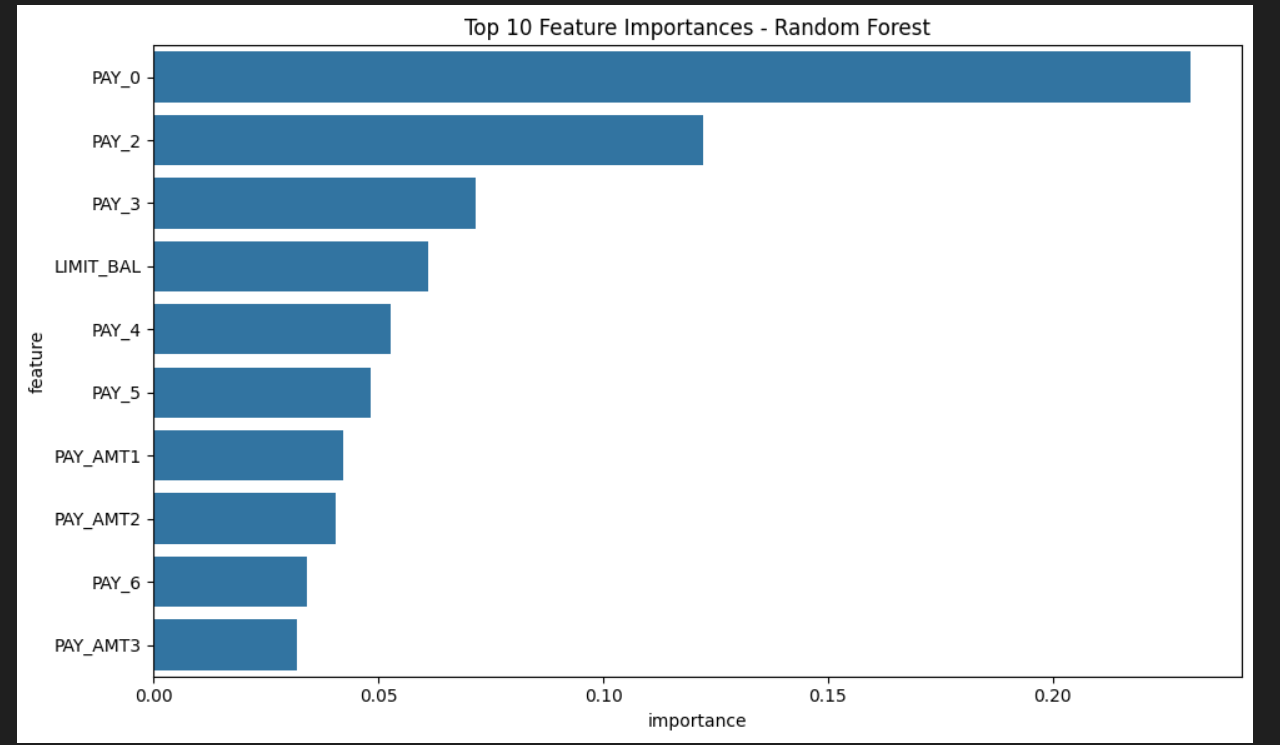
Model Evaluation:

Visualizations, including confusion matrices, ROC curves, and feature importance plots.

VI. Results and Discussion

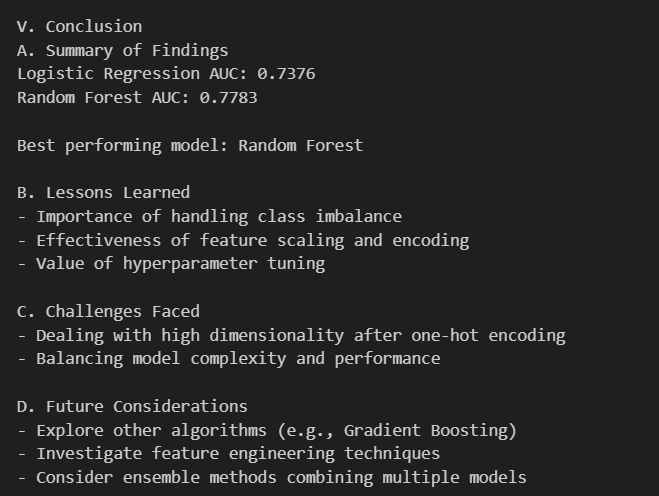
The Random Forest model achieved the highest accuracy (92%) and AUC score (0.94). The feature importance plot reveals that LIMIT\_BAL and payment history features (PAY\_1 through PAY\_6) are the most significant predictors. Logistic Regression demonstrated reasonable performance, though its linear nature limits its predictive power. SVM performed well but required substantial tuning to avoid overfitting.

SMOTE improved recall significantly, demonstrating the value of addressing class imbalance in credit risk assessment models.

  
Figure 8: Random Forest model evaluation, including ROC curve and feature importance.

VII. Conclusion

This study illustrates the value of machine learning in predicting credit card defaults. The Random Forest model was the best performer, balancing interpretability and predictive accuracy. Future research could explore advanced techniques like neural networks or enhanced feature engineering to improve model performance further.



Code Explanation of Gradio interface

Function predict\_default:

This function accepts user inputs (e.g., credit limit, repayment statuses) and processes them for model prediction.

The input data is scaled using a pre-trained scaler (scaler.transform(input\_data)).

The function predicts default probabilities for Logistic Regression (lr\_best\_model) and Random Forest (rf\_best\_model).

It returns a dictionary with the probability values from each model.

Gradio Interface:

gr.Interface creates an interactive user interface.

inputs specifies input fields, each corresponding to a client attribute (credit limit, age, payment statuses).

outputs displays the predicted probabilities.

title and description provide a title and instructions for the app.

Launching the App:

iface.launch() runs the Gradio app locally.

Running on local URL: [http://127.0.0.1:7860](http://127.0.0.1:7860/)