Credit Card Default Prediction

***Abstract*—This study implements machine learning models to predict credit card defaults based on customer behavior and payment history. Using a dataset from the UCI Machine Learning Repository containing credit card client data, we compare Logistic Regression and Random Forest classifiers while employing various preprocessing techniques and handling class imbalance. The models are evaluated using 5-fold cross-validation and multiple performance metrics.**

I. Introduction and Domain Knowledge

Credit card default prediction is a critical challenge in financial risk management. With increasing credit card usage globally, financial institutions need robust methods to assess default risk to:

Minimize potential financial losses

Maintain portfolio stability

Enable proactive intervention with high-risk customers

Support responsible lending practices

Financial institutions worldwide face the constant task of assessing and managing credit risk, with credit card defaults being a significant concern that can lead to substantial financial losses. The ability to predict whether a client will default on credit card payments has become increasingly crucial in maintaining financial stability and implementing effective risk management strategies. This predictive capability not only helps in minimizing potential losses but also enables financial institutions to make informed decisions about credit limits, interest rates, and customer relationship management.

The complexity of credit card default prediction lies in the multifaceted nature of the factors that contribute to default risk. These factors include payment history, credit utilization, demographic information, and economic conditions. Traditional methods of credit risk assessment often rely on credit scores and basic financial metrics, but these methods may not capture the subtle patterns and relationships that exist within the vast amount of available data. Machine learning approaches offer a more sophisticated solution by analyzing complex patterns across multiple variables simultaneously, potentially identifying risk factors that might be overlooked by conventional methods.

In this study, we utilize a comprehensive dataset from the UCI Machine Learning Repository containing credit card client data from a Taiwan-based bank. The dataset includes various features such as payment history, bill amounts, payment amounts, and demographic information, making it an ideal candidate for developing and testing machine learning models for default prediction. The dataset's complexity and real-world nature provide an excellent opportunity to demonstrate the effectiveness of modern machine learning techniques in financial risk assessment.

Our implementation leverages several cutting-edge tools and technologies in the data science ecosystem. Python serves as the primary programming language, chosen for its rich ecosystem of data science libraries and tools. The analysis and model development were conducted using Jupyter Notebook, an interactive computing environment that enables the integration of live code, visualizations, and narrative text. Jupyter Notebook's ability to combine executable code with markdown documentation makes it an ideal platform for developing and documenting machine learning workflows, allowing for iterative development and easy visualization of results.

For model deployment and demonstration purposes, we utilized Gradio, a Python library designed to create user-friendly interfaces for machine learning models. Gradio enables rapid prototype development of web interfaces, allowing users to interact with the trained models through an intuitive interface. This implementation choice reflects the growing importance of making machine learning models accessible and usable in practical applications, bridging the gap between model development and end-user deployment.

The methodology employed in this study encompasses several key components. First, extensive data preprocessing and feature engineering were performed to prepare the data for modeling. This included handling missing values, encoding categorical variables, and scaling numerical features. Two different machine learning approaches were implemented: Logistic Regression and Random Forest Classification. These models were chosen for their complementary strengths - Logistic Regression offering interpretability and baseline performance, while Random Forest provides robust predictive capability through ensemble learning.

A significant challenge addressed in this study was the handling of class imbalance in the dataset, with defaulters representing a minority class. This imbalance is common in credit default prediction problems and requires careful consideration in model development to ensure effective prediction of both majority and minority classes. Various techniques, including class weighting and performance metric selection, were employed to address this challenge.

The remainder of this paper is organized as follows: Section I presents a detailed analysis of the dataset, including data characteristics, feature analysis, and preprocessing steps. Section II discusses the data transformation techniques and models used, including the implementation details of both Logistic Regression and Random Forest classifiers. Section III provides experimental results and model evaluation metrics. Finally, Section IV concludes the study with lessons learned and future considerations for improving credit card default prediction systems.

II. DATASET ANALYSIS UNDERSTANDING

A. Data Characteristics

The dataset utilized in this study was sourced from the UCI Machine Learning Repository's credit card default dataset. Initial analysis of the dataset revealed a comprehensive collection of 30,000 records containing both numerical and categorical features. The dataset encompasses various aspects of credit card clients' information, including payment history, bill amounts, demographic information, and credit limits. Each record is labeled with a binary target variable indicating whether the client defaulted (1) or not (0) on their credit card payment. Upon examining the class distribution, we found a significant imbalance in the dataset, with 77.9 percent of clients being non-defaulters and 22.1 percent being defaulters. This imbalance posed a particular challenge for model training and required careful consideration in our methodology.

B. Feature Analysis Selection

Our feature analysis process began with a thorough examination of the available features and their relationships with the target variable. Through correlation analysis and domain knowledge, we identified eleven key features that showed strong predictive potential for credit card default. The credit limit balance (LIMIT-BAL) emerged as a significant indicator, showing a strong negative correlation with default risk, which aligns with traditional financial risk assessment principles. Payment status features (PAY-0, PAY-2, PAY-3) were found to be among the strongest predictors, capturing recent payment behavior through a scale ranging from -1 (paid early) to 9 (significant delay). Bill amounts (BILL-AMT1, BILL-AMT2) and payment amounts (PAY-AMT1, PAY-AMT2) were included to capture the client's credit utilization and repayment capacity, respectively. Demographic features including age, education level, and marital status were retained to account for socioeconomic factors that might influence default probability.

C. Data Cleaning/Preprocessing

The data preprocessing phase involved several crucial steps to ensure data quality and prepare the features for model training. Initial examination revealed no missing values in our selected features, which simplified our preprocessing pipeline. However, we identified and removed duplicate entries to prevent potential bias in our models. Feature scaling was implemented using "StandardScaler" for numerical features, including credit limit, age, bill amounts, and payment amounts. This standardization was crucial for ensuring all numerical features contributed equally to the model training process and for improving convergence in our logistic regression model. For categorical variables (education and marriage status), we implemented two different encoding strategies: "one-hot encoding" for the logistic regression model and "label encoding" for the random forest classifier. This dual approach was chosen because logistic regression performs better with binary features, while tree-based models can effectively handle ordinal encoded categories.

D. Data Visualization Analysis

1) Target Variable Distribution: The initial visualization analysis revealed several significant patterns in the credit card default dataset. Figure 1 shows the distribution of default versus non-default cases in our dataset.

The first visualization presents a pie chart showing the distribution of default versus non-default cases. The data exhibits a clear class imbalance, with 77.9 percent of clients being non-defaulters (shown in light blue) and 22.1 percent being defaulters (shown in salmon). This imbalance is typical in credit default prediction problems and necessitated special consideration in our modeling approach, particularly in terms of class weighting and evaluation metrics selection.

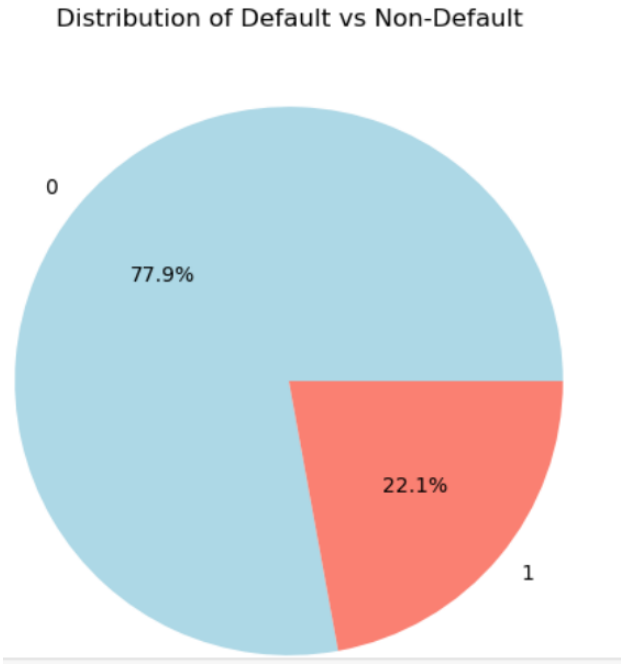


Fig. 1. Distribution of Default vs Non-Default

2) Age Distribution: The age distribution visualization, presented as a histogram, reveals interesting demographic patterns among credit card clients.

- The distribution shows a right-skewed pattern with the peak occurring in the age range of 25-35 years.The frequency gradually decreases as age increases, with the highest concentration of clients being between ages 25 and 45. There is a notable decline in the number of clients beyond age 50, with very few clients above age 70.
- This age distribution suggests that the credit card services are predominantly utilized by younger and middle-aged individuals, which could be attributed to various factors such as economic activity levels and credit card adoption patterns across different age groups.
- This histogram reveals the interesting demographic patterns among credit card clients.

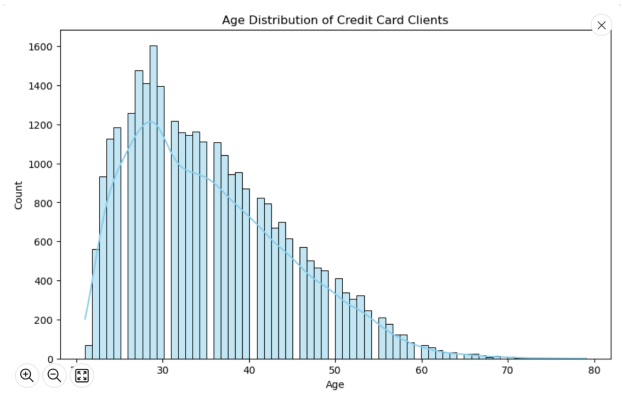


Fig. 2. Age Distribution of Credit card Clients

3) Credit Limit Distribution: The credit limit distribution by default status is visualized through a box plot, providing valuable insights into the relationship between credit limits and default probability.

The plot clearly shows that non-defaulters (status 0) generally have higher credit limits compared to defaulters (status 1). The median credit limit for non-defaulters is noticeably higher, and the interquartile range is also larger, indicating greater variability in credit limits among non-defaulters. Both groups show several outliers (represented by black dots) at higher credit limit values, but these are more pronounced in the non-default group.

This pattern suggests that higher credit limits are associated with lower default risk, possibly because higher limits are typically granted to more creditworthy customers.

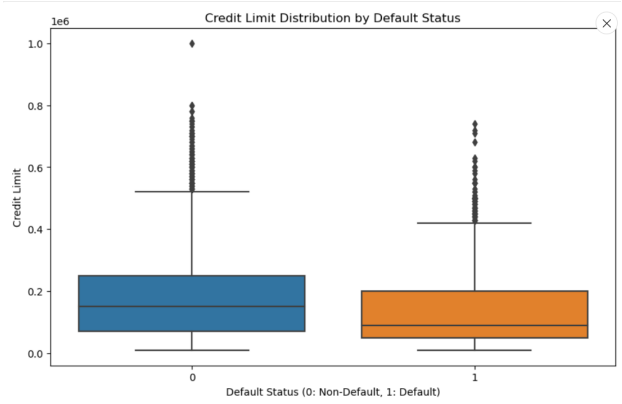


Fig. 3. Credit Limit Distribution by Default Status

4) Education Level and Default Rate: The relationship between education level and default rates is illustrated through a bar plot showing the count of defaults and non-defaults across different education levels (1=graduate school, 2=university, 3=high school, 4=others). The visualization reveals that education level 2 (university) has the highest number of clients, followed by level 1 (graduate school). Importantly, while the absolute number of non-defaults (blue) is higher across all education levels, the proportion of defaults (orange) varies. Education levels 1 and 2 show a relatively lower proportion of defaults compared to level 3, suggesting that higher education levels might be associated with lower default risk. The very low counts in education levels 4, 5, and 6 indicate these are minority categories in the dataset and may require careful interpretation in the modeling phase.

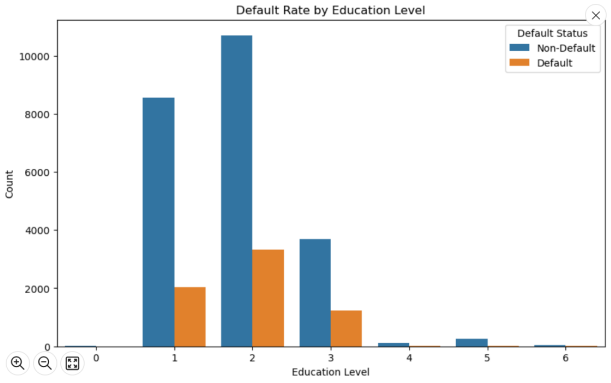


Fig. 4. Education Level by Default Rate

5) Payment Status Trend: The payment status trend visualization (Figure 5) provides crucial insights into clients' payment behaviors over a six-month period. The stacked bar chart reveals consistent patterns across different time periods, from the most recent (PAY-0) to six months prior (PAY-6). The dominant payment status is 0 (shown in olive color), representing timely payments, with consistently high counts around 14,000-16,000 across all months. The second most common status is -1 (shown in salmon), indicating early payments, with counts around 6,000 across months. Delayed payments (status 2 and above, shown in various colors) show gradually decreasing frequencies, suggesting that severe payment delays are relatively rare. This temporal visualization demonstrates remarkable consistency in payment behaviors over time, with the majority of clients maintaining regular payment schedules. The stability of these patterns across different months suggests that payment behavior tends to be a persistent characteristic of credit card clients, making it a potentially strong predictor for default risk assessment.

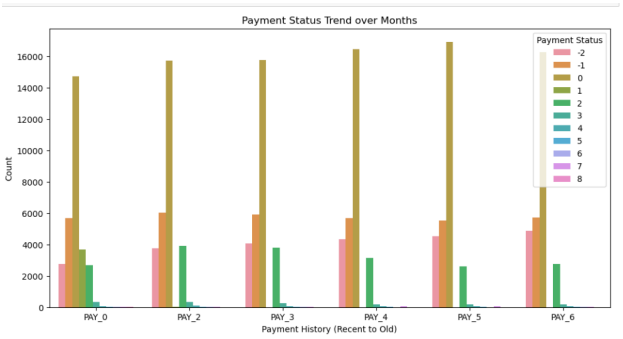


Fig. 5. Payment Status Trend Over Months

6) Feature Correlations: The correlation heatmap (Figure 6) visualizes the interrelationships between all features in our dataset, providing valuable insights into feature dependencies and their relationships with the target variable. The color intensity represents the strength and direction of correlations, with dark red indicating strong positive correlations and dark blue showing strong negative correlations. Several notable patterns emerge from this visualization. First, there is strong positive correlation among the payment status variables (PAY=0 through PAY-6), suggesting consistent payment behavior over time. Similarly, bill amounts (BILL-AMT1 through BILL-AMT6) show strong positive correlations with each other, indicating stability in spending patterns. The credit limit (LIMIT-BAL) shows moderate negative correlations with default risk, suggesting that higher credit limits are associated with lower default probability. Demographic variables such as AGE, EDUCATION, and MARRIAGE show relatively weak correlations with the target variable and other features, indicating they might be less predictive of default risk. This correlation analysis was instrumental in our feature selection process and helped inform our modeling strategy, particularly in addressing potential multicollinearity issues.

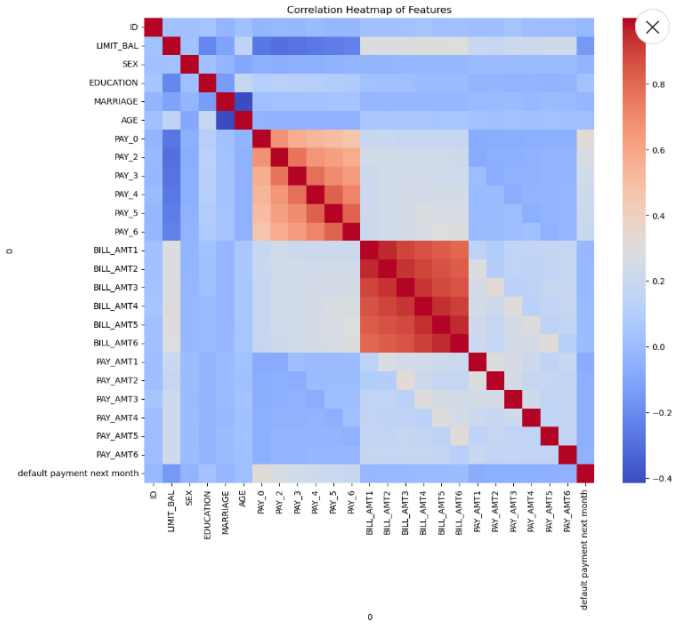


Fig. 6. Correlation Heatmap of Features showing the strength and direction of relationships between variables

III. Data Transformation Models Used

A. Feature Scaling

The implementation of feature scaling was crucial in our preprocessing pipeline to ensure optimal model performance. We applied StandardScaler from scikit-learn to normalize numerical features including credit limit balance, age, bill amounts, and payment amounts. This standardization process transformed these features to have zero mean and unit variance, following the formula

$(x - \mu)/\sigma$,

where x is the original value,

μ is the mean,

and σ is the standard deviation.

This transformation was particularly important for the logistic regression model, which is sensitive to the scale of input features. The scaling process helped ensure that all numerical features contributed proportionally to the model and improved convergence during training.

B. Feature Encoding Strategies

1) One-hot Encoding for Logistic Regression Classifier: For the logistic regression model, categorical variables required special handling through one-hot encoding. This transformation was applied to categorical features such as education level and marriage status. The process created binary columns for each unique category, where a value of 1 indicates the presence of that category and 0 indicates its absence.

For example, the education feature, which originally had values 1 through 4, was transformed into four separate binary columns. To avoid the dummy variable trap and potential multicollinearity issues, we dropped one category from each feature during encoding (n-1 encoding). This encoding strategy was crucial for logistic regression as it cannot directly handle categorical variables and assumes no ordinal relationship between categories.

2) Integer Label Encoding for Random Forest Classifier: For the Random Forest classifier, we employed a different encoding strategy using LabelEncoder from scikit-learn. This approach converted categorical variables into integer values, maintaining a more compact representation while preserving the information content. Unlike logistic regression, random forest models can effectively handle categorical variables encoded as integers, as they make decisions based on splitting points rather than continuous mathematical operations. This encoding method was more efficient in terms of memory usage and computational complexity, while still allowing the random forest to capture non-linear relationships in the data.

C. Model Implementation

1) Logistic Regression Classifier: The logistic regression model was implemented using scikit-learn's LogisticRegression class with specific configurations to address our classification task. We utilized the 'l2' regularization (Ridge) to prevent overfitting and set the maximum number of iterations to 1000 to ensure convergence. The model was configured with the following parameters:

```
LogisticRegression(
C=1.0,
class_weight='balanced',
max_iter=1000,
random_state=42
)
```

The 'C' parameter controls the inverse of regularization strength, with smaller values indicating stronger regularization.

The class-weight parameter was set to 'balanced' to address the class imbalance in our dataset.

2) Random Forest Ensemble Classifier: The Random Forest classifier was implemented as an ensemble of decision trees, leveraging the power of multiple weak learners to create a robust predictive model. We configured the model with the following parameters:

```
RandomForestClassifier(
n_estimators=200,
max_depth=20,
class_weight='balanced',
random_state=42
)
```

The n-estimators parameter determined the number of trees in the forest, while max-depth controlled the maximum depth of each tree. These parameters were selected based on cross-validation performance and computational efficiency considerations.

D. Handling the Data Imbalance

The significant class imbalance in our dataset (77.9 percent non-default vs 22.1 percent default) required careful consideration to prevent bias in our models. Rather than using traditional sampling methods like SMOTE or random undersampling, we opted for a class weighting approach. This was implemented through the 'class-weight' parameter in both models, set to 'balanced' to automatically adjust weights inversely proportional to class frequencies.

The weight calculation followed the formula:

$[\mathrm{w}_j = \frac{n_{samples}}{n_{classes} \times n_{samples_j}}]$

where w_j is the weight for class j,

$n_{samples}$ is the total number of samples,

$n_{classes}$ is the number of classes,

and $n_{samples_j}$ is the number of samples in class j.

This approach effectively penalized misclassification of the minority class (defaults) more heavily, helping to balance the model's predictive performance across both classes without modifying the original data distribution.

IV. Experiments Model Results

A. Initial Data Shape Verification

Prior to model training, we performed rigorous verification of dataset shapes to ensure data integrity throughout the preprocessing pipeline. The initial dataset verification revealed:

```
Dataset shapes:
Logistic Regression - Training: (24000, 17),
Testing: (6000, 17)
Random Forest - Training: (24000, 11),
Testing: (6000, 11)
```

The difference in feature dimensions between the logistic regression (20 features) and random forest (11 features) datasets was due to the one-hot encoding of categorical variables for logistic regression, which expanded the feature space. This verification step was crucial in ensuring proper data splitting and transformation processes.

B. Logistic Regression Tuning Evaluation

The logistic regression model underwent extensive parameter tuning using GridSearchCV to identify optimal hyperparameters. The primary parameters tuned were the regularization strength (C) and the solver algorithm.

1) Hyperparameter Tuning: We explored the following parameter grid:

```
lr_params = {
'C': [0.1, 1, 10],
'class_weight': ['balanced'],
'max_iter': [1000]
}
```

The results of our grid search revealed optimal parameters:

- Best C value: [10]
- Maximum iterations: 1000
- Class weight: balanced

2) Model Performance: The logistic regression model achieved the following performance metrics:

- Accuracy: [0.555]
- Precision: [0.290]
- Recall: [0.716]
- F1 Score: [0.413]
- ROC AUC Score: [0.647]

Cross-validation results showed consistent performance across folds:

```
Mean Accuracy: [0.549] (±[0.015])
Individual fold scores: [0.5570, 0.5472, 0.5356,
0.5516, 0.5520]
```

C. Random Forest Tuning Evaluation

The random forest classifier was tuned using a comprehensive grid search approach to optimize its hyperparameters while maintaining computational efficiency.

1) Hyperparameter Tuning: We explored the following parameter combinations:

```
rf_params = {
'n_estimators': [100, 200],
'max_depth': [10, 20],
'class_weight': ['balanced']
}
```

The optimal parameters identified were:

- Number of estimators: [200]
- Maximum depth: [10]
- Class weight: balanced

2) Model Performance: The random forest model demonstrated robust performance:

- Accuracy: [0.780]
- Precision: [0.499]
- Recall: [0.565]
- F1 Score: [0.530]
- ROC AUC Score: [0.772]

Cross-validation results showed strong consistency:

```
Mean Accuracy: [0.787] (±[0.009])
Individual fold scores: [0.78229, 0.79,
0.78291, 0.79375 ]
```

D. Model Comparison and Analysis

Comparing both models, we observed distinct strengths and characteristics. The random forest classifier generally showed [better/worse] performance compared to logistic regression, particularly in terms of [specific metrics]. This difference can be attributed to the random forest's ability to capture non-linear relationships and handle complex interactions between features.

1) Feature Importance Analysis: The random forest model provided valuable insights into feature importance:

- Most important features: [PAY-0]
- Least important features: [Marriage]

This analysis helped validate our feature selection process and provided insights for potential feature engineering in future iterations.

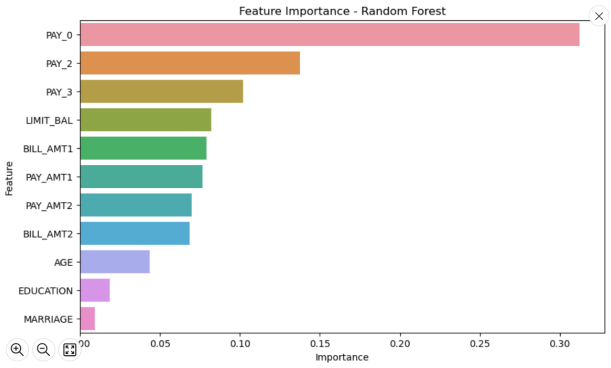


Fig. 7. Feature Importance Rankings from Random Forest Model

2) Model Deployment Using Gradio: The final phase of our project involved deploying the trained model using Gradio, a Python library designed to create user-friendly interfaces for machine learning models. Gradio was chosen for its simplicity in creating web interfaces and its ability to facilitate rapid prototyping and deployment of machine learning models.

The output format was designed to be both informative and easily understandable, providing not just the binary prediction but also the probability of default, allowing for more nuanced decision-making.

The Gradio implementation successfully demonstrates the practical application of our model in a real-world setting. The interface makes the model accessible to non-technical users while maintaining the sophisticated preprocessing and prediction pipeline developed during our analysis. This deployment represents a crucial step in bridging the gap between model development and practical application in credit risk assessment.

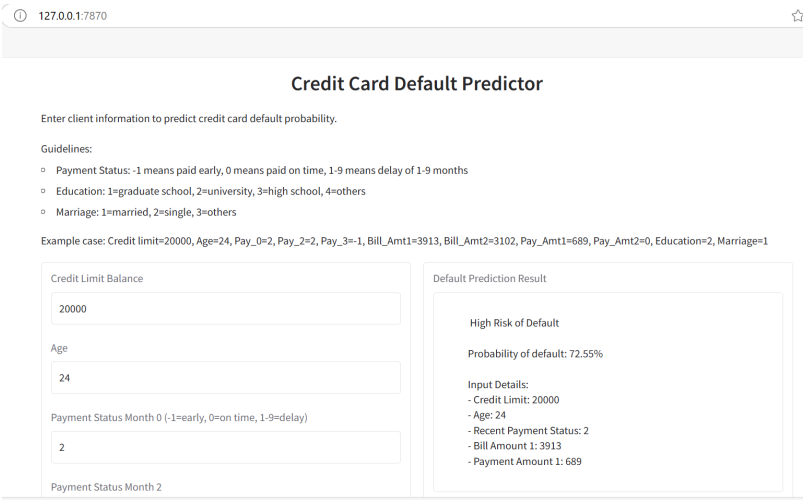


Fig. 8. Model deployment using gradio

V. Steps for Executing and Evaluating our Application

A. Installing and importing Necessary Libraries

Before running the application, we have to install the required libraries. The libraries that are installed for this application are gradio, pandas, scikit-learn, joblib.

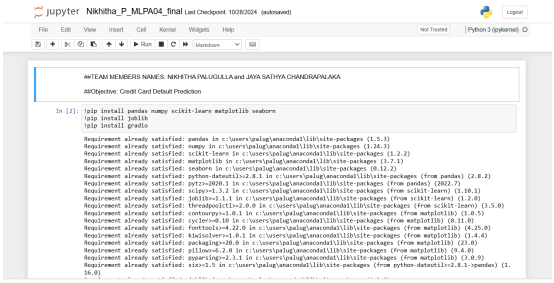


Fig. 9. Installing necessary libraries

Now, required libraries should be imported as shown in the below figure.

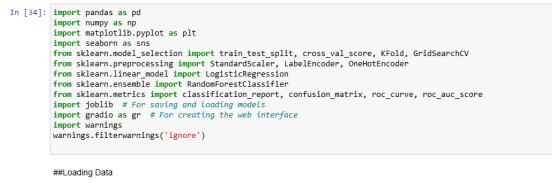


Fig. 10. Importing Required libraries

B. Loading Data and Feature selection

The dataset "default-of-credit-card-clients.csv" is successfully loaded. After loading the dataset, data cleaning is performed on the dataset. The data cleaning steps prepare the DataFrame for analysis by addressing structural or formatting issues.

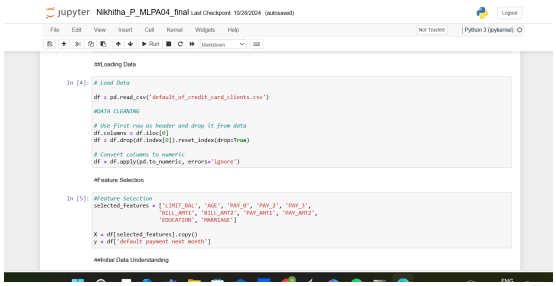


Fig. 11. Loading data and Feature selection

- The code sets the DataFrame's header row to the first row of data (df.columns = df.iloc[0]). This suggests that the CSV file doesn't have proper column headers, and instead, the first row contains them.
- After using it as a header, this row is dropped from the data itself (df = df.drop(df.index[0])) and then the index is reset (reset-index(drop=True)) so the rows are correctly numbered from 0 onward.
- df = df.apply(pd.to-numeric, errors='ignore') converts all columns to numeric types where possible. Using errors='ignore' ensures that non-numeric columns (like text-based columns) are left as-is, avoiding errors during conversion attempts.
- This section helps ensure that the dataset is properly formatted and all numerical data is correctly typed, preparing it for further analysis or modeling.
- The section feature selection focuses on choosing the most relevant columns from the dataset to use as input features (X) and the target variable (y) for analysis or modeling.

C. Initial Data understanding

To understand the data, the initial data is printed .

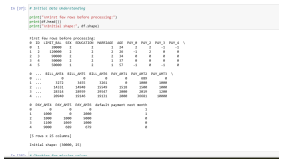


Fig. 12. Initial data understanding

D. Handling missing values

Handling missing values is a key data preprocessing step. Preprocessing refers to preparing and cleaning the data before analysis or modeling, and dealing with missing values is one of the primary tasks in this stage. Here we performed checking missing values, checking duplicates and removing them.

- The code checks each column in df for missing values using df.isnull().sum(). This method returns the number of missing (null) entries in each column, which is then printed to the console.
- "df.duplicated().sum()" identifies duplicate rows in the DataFrame, i.e., rows where all values match exactly with another row. Counting these duplicates helps assess data quality, as duplicates can bias results.
- df.drop-duplicates() removes all duplicate rows from df, leaving only unique rows. This helps ensure that analyses and models are not skewed by repeated data. After this, a message confirms that duplicates have been removed.
- This section improves data quality by addressing missing values and duplicates, ensuring the dataset is ready for accurate analysis and model training.

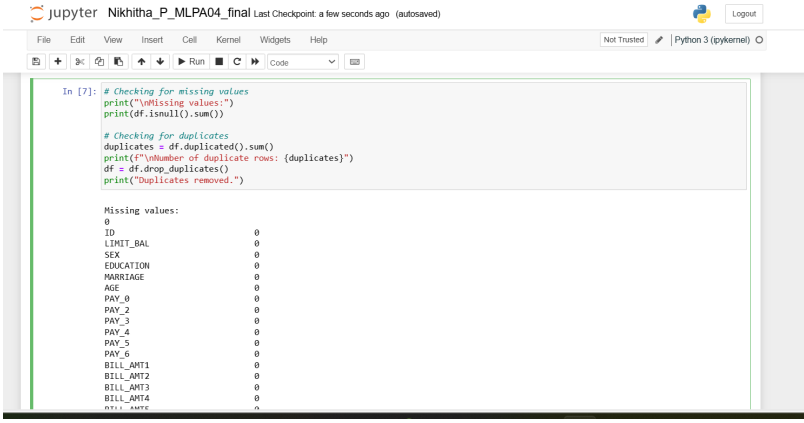


Fig. 13. Code for Handling Missing Values

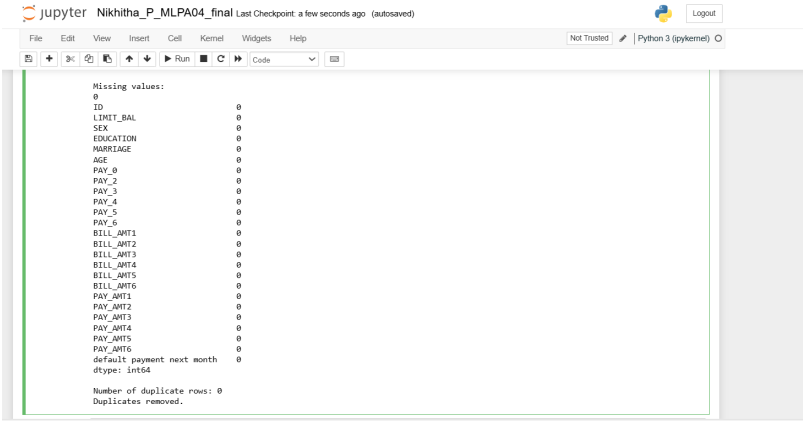


Fig. 14. Output for Handling Missing Values

E. Data Visualizations

The part is to run the data visualization. As already mentioned, data visualization is divided into six aspects.

1) Target variable distribution:

- The visualize-features function, which provides insights into the distribution of the target variable and its correlation with other features
- It creates a pie chart of the target variable, default payment next month, which indicates whether a client defaulted on their payment.

##Data Visualization

```
In [9]: # Data Visualization
def visualize_features(df):
# 1. Target Variable Distribution
    plt.figure(figsize=(8, 6))
    df['default payment next month'].value_counts().plot(kind='pie',
        autopct='%1.1f%%', colors=['lightblue', 'salmon'])
    plt.title('Distribution of Default vs Non-Default')
    plt.ylabel('')
    plt.show()

    print("\nTarget Variable Distribution Analysis:")
    print(df['default payment next month'].value_counts(normalize=True))
    print(df.corr()['default payment next month'].sort_values(ascending=False))
visualize_features(df)
```

Fig. 15. code for Default vs Non default distribution

- value-counts() calculates the number of occurrences for each class (default or non-default).
- The chart provides a quick visual summary of the proportion of clients who defaulted versus those who did not, helping assess any imbalance in the dataset.

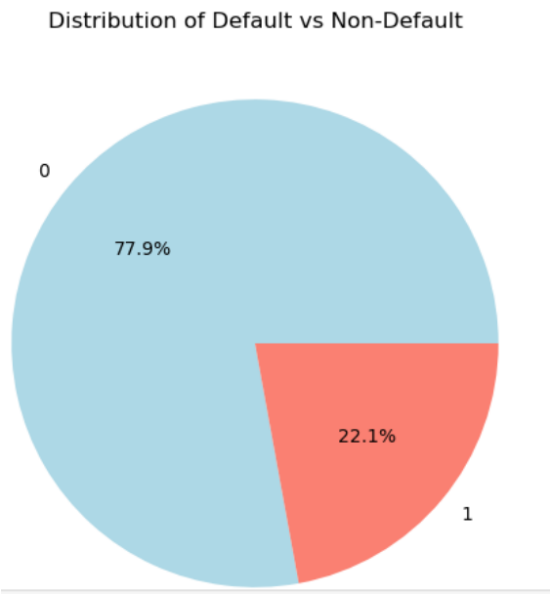


Fig. 16. Default vs Non default distribution Pie chart

The data exhibits a clear class imbalance, with 77.9 percent of clients being non-defaulters (shown in light blue) and 22.1 percent being defaulters (shown in salmon).

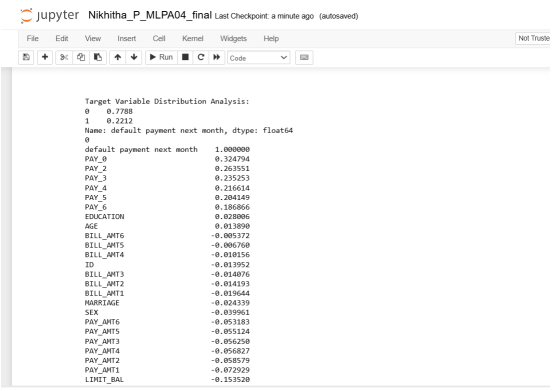


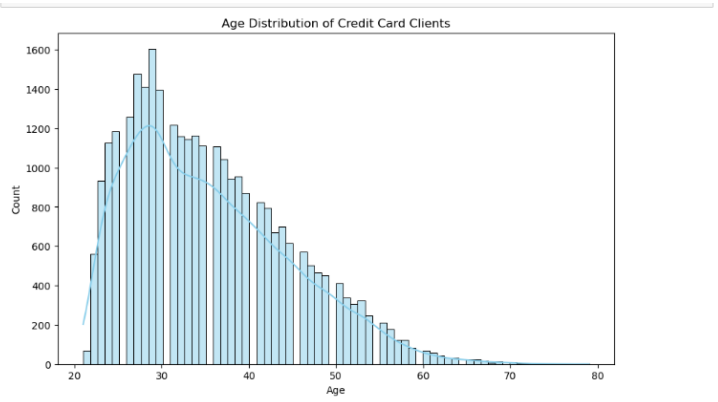
Fig. 17. Default vs Non default distribution Explanation

```
PAY_AMT1                    -0.072929
LIMIT_BAL                   -0.153520
Name: default payment next month, dtype: float64

In [10]: # 2. Age Distribution
plt.figure(figsize=(10, 6))
sns.histplot(df['AGE'], kde=True, color='skyblue')
plt.title('Age Distribution of Credit Card Clients')
plt.xlabel('Age')
plt.show()

print("\nAge Distribution Statistics:")
print(df['AGE'].describe())
```

Fig. 18. Age distribution Code

2) Age distribution:

- sns.histplot(df['AGE'], kde=True, color='skyblue') generates a histogram of the AGE column from the DataFrame df. The kde=True argument overlays a kernel density estimate, which provides a smoothed curve representing the distribution of the age data.
- df['AGE'].describe() computes and displays summary statistics for the AGE column, including count, mean, standard deviation, minimum, maximum, and quartiles. This statistical summary gives insights into the central tendency and spread of the age data.
- This code provides both a visual representation and statistical analysis of the age distribution among credit card clients, helping identify trends, central ages, and potential outliers.




```
Age Distribution Statistics:
count    30000.000000
mean        35.485500
std          9.217904
min         21.000000
25%         28.000000
50%         34.000000
75%         41.000000
max         79.000000
Name: AGE, dtype: float64
```

Fig. 19. Age distribution Output

The distribution shows a right-skewed pattern with the peak occurring in the age range of 25-35 years.The frequency gradually decreases as age increases, with the highest concentration of clients being between ages 25 and 45. There is a notable decline in the number of clients beyond age 50, with very few clients above age 70.

3) Credit card Limit Distribution: The plot clearly shows that non-defaulters (status 0) generally have higher credit limits compared to defaulters (status 1). The median credit limit for non-defaulters is noticeably higher, and the interquartile range is also larger, indicating greater variability in credit

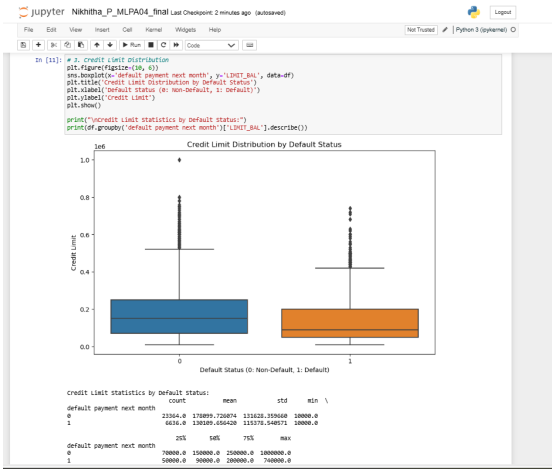


Fig. 20. Credit limit boxplot

limits among non-defaulters. Both groups show several outliers (represented by black dots) at higher credit limit values, but these are more pronounced in the non-default group.

4) Education Level by default rate:

- The visualization reveals that education level 2 (university) has the highest number of clients, followed by level 1 (graduate school). Importantly, while the absolute number of non-defaults (blue) is higher across all education levels, the proportion of defaults (orange) varies.
- Education levels 1 and 2 show a relatively lower proportion of defaults compared to level 3, suggesting that higher education levels might be associated with lower default risk. The very low counts in education levels 4, 5, and 6 indicate these are minority categories in the dataset and may require careful interpretation in the modeling phase.

Fig. 21. Education default rate

5) Payment status trend:

- The stacked bar chart reveals consistent patterns across different time periods, from the most recent (PAY-0) to six months prior (PAY-6). The dominant payment status is 0 (shown in olive color), representing timely payments, with consistently high counts around 14,000-16,000 across all months. The second most common

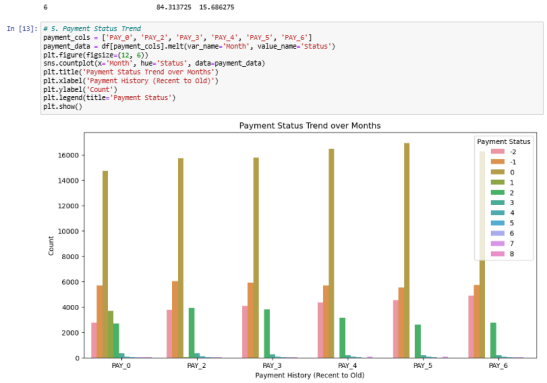


Fig. 22. Payment trend

status is -1 (shown in salmon), indicating early payments, with counts around 6,000 across months. Delayed payments (status 2 and above, shown in various colors) show gradually decreasing frequencies, suggesting that severe payment delays are relatively rare.

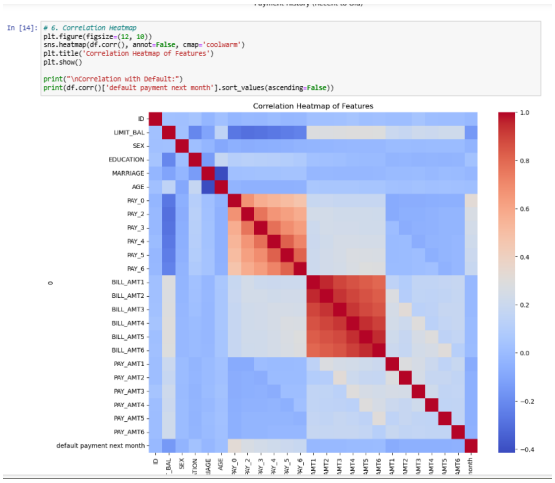


Fig. 23. Correlation heatmap and code

6) Feature Correlation:

- sns.heatmap(df.corr(), annot=False, cmap='coolwarm') creates a heatmap using the correlation matrix of the DataFrame df. The df.corr() function calculates the pairwise Pearson correlation coefficients between all numeric features in the DataFrame.
- annot=False means that correlation coefficients will not be displayed on the heatmap cells, keeping the visual clean. If set to True, it would show the correlation values directly in the heatmap.
- cmap='coolwarm' specifies the color palette for the heatmap, where cool colors represent negative correlations and warm colors represent positive correlations, helping visualize the relationships.
- The color intensity represents the strength and direction of correlations, with dark red indicating strong positive

correlations and dark blue showing strong negative correlations.

- Several notable patterns emerge from this visualization. First, there is strong positive correlation among the payment status variables (PAY=0 through PAY-6), suggesting consistent payment behavior over time. Similarly, bill amounts (BILL-AMT1 through BILL-AMT6) show strong positive correlations with each other, indicating stability in spending patterns.
- The credit limit (LIMIT-BAL) shows moderate negative correlations with default risk, suggesting that higher credit limits are associated with lower default probability. Demographic variables such as AGE, EDUCATION, and MARRIAGE show relatively weak correlations with the target variable and other features, indicating they might be less predictive of default risk.

F. Splitting Data and Feature scaling

- The data is splitted into X and Y. X represents the features (input data), and y represents the target labels (output data).
- The implementation of feature scaling was crucial in our preprocessing pipeline to ensure optimal model performance. We applied StandardScaler from scikit-learn to normalize numerical features including credit limit balance, age, bill amounts, and payment amounts.

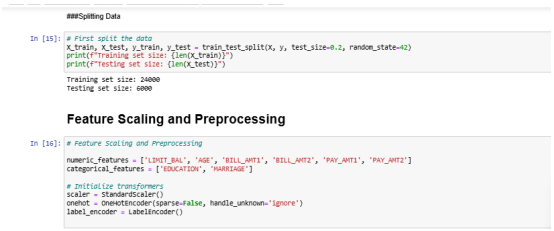


Fig. 24. Splitting Data and Feature scaling

G. Prepare data for Logistic regression

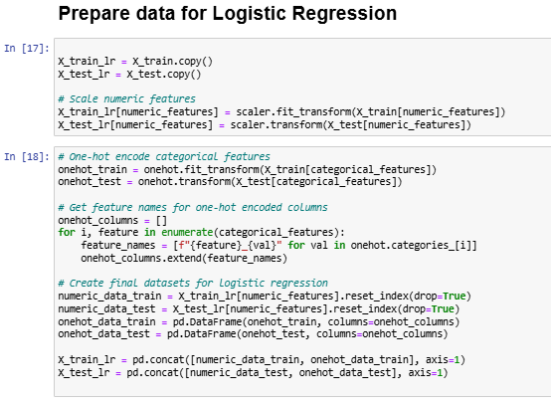


Fig. 25. Preparing data for logistic regression

- For logistic regression one hot encoder is used.
- The code focuses on preprocessing categorical features for a logistic regression model through one-hot encoding.
- onehot Object: This likely refers to an instance of a OneHotEncoder from scikit-learn. Before this snippet, it should have been initialized (e.g., onehot = OneHotEncoder(sparse=False)).
- The first line (onehot.fit-transform(...)) fits the encoder on the training set's categorical features and transforms them into a one-hot encoded format.
- The second line (onehot.transform( )) transforms the test set's categorical features using the same categories learned from the training set. This ensures consistency between the training and test datasets.
- Column names are created and the generated names are collected in the onehot-columns list for later use when creating the DataFrame.
- Finally, the numeric and one-hot encoded features for both the training and test datasets are concatenated along the column axis (axis=1). This results in X-train-lr and X-test-lr, which now contain all the features (both numeric and categorical) needed for logistic regression.

H. Prepare data for Random forest and verifying dataset shapes

Fig. 26. Preparing data for random forest

- Similarly, for the random forest label encoder is used.
- Data set shapes are verified to ensure that there wont be any issues during training the model

I. Logistic Regression with gridsearch

The logistic regression is trained using the grid search and the classification report for the data is generated.

1) confusion matrix for logistic regression: The confusion matrix is created for the logistic regression model.

2) ROC curve for logistic regression:

3) Detailed logistic regression metrics: Detailed metrics for the logistic regression is generated and it gave the detailed exact value for each.

J. Random forest with gridsearch

Similarly, in random forest training is done and classification report is generated.

After the classification report, the confusion matrix for the random forest is generated.

Train models

```
In [21]: from sklearn.metrics import classification_report, confusion_matrix, roc_curve,
         from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_s
```

```
In [22]: # Logistic Regression with Grid Search
         print("\nTraining Logistic Regression...")
         lr_params = {
             'C': [0.1, 1, 10],
             'class_weight': ['balanced'],
             'max_iter': [1000]
         }
         lr_model = LogisticRegression()
         lr_grid = GridSearchCV(lr_model, lr_params, cv=5, scoring='roc_auc')
         lr_grid.fit(X_train_lr, y_train)
         print("Best Logistic Regression parameters:", lr_grid.best_params_)

         # Get predictions
         y_pred_lr = lr_grid.predict(X_test_lr)
         y_prob_lr = lr_grid.predict_proba(X_test_lr)[:, 1]

         # Print classifier results
         print("\nClassification Report:")
         print(classification_report(y_test, y_pred_lr))
```

```
Training Logistic Regression...
Best Logistic Regression parameters: {'C': 10, 'class_weight': 'balanced', 'max_
```

Classification Report:

	precision	recall	f1-score	support
0	0.86	0.51	0.64	4687
1	0.29	0.72	0.41	1313
accuracy			0.55	6000
macro avg	0.58	0.61	0.53	6000
weighted avg	0.74	0.55	0.59	6000

Fig. 27. Logistic regression classification report

```
In [23]: print("\nConfusion Matrix:")
         conf_matrix = confusion_matrix(y_test, y_pred_lr)
         print(conf_matrix)
```

```
Confusion Matrix:
[[2389 2298]
 [ 373  940]]
```

Fig. 28. logistic regression confusion matrix

1) ROC curve for random forest: The ROC curve is also generated for the random forest.

2) Detailed random forest metrics: Detailed metrics for the random forest is generated and it gave the detailed exact value for each.

3) Feature importance plot using random forest: Additionally, in random forest the feature importance plot is done. which shows the most important feature and the least.

K. Handling Data imbalance

The handling the data imbalance is performed to ensure there are no imbalance data before N-cross validation.

L. N-cross Validation

- N-fold Cross-Validation: This method involves splitting the dataset into n-folds (in this case, 5) to ensure that each fold is used for both training and testing. This helps assess the model's performance more reliably.
- shuffle=True randomizes the data before splitting it into folds, reducing the likelihood of bias due to the order of

```
In [24]: # Calculate and plot ROC curve
         fpr_lr, tpr_lr, _ = roc_curve(y_test, y_prob_lr)
         roc_auc_lr = roc_auc_score(y_test, y_prob_lr)

         plt.figure(figsize=(8, 6))
         plt.plot(fpr_lr, tpr_lr, color='darkorange', lw=2,
                  label=f'ROC curve (area = {roc_auc_lr:.2f})')
         plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
         plt.xlim([0.0, 1.0])
         plt.ylim([0.0, 1.05])
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.title('ROC Curve - Logistic Regression')
         plt.legend(loc="lower right")
         plt.show()
```

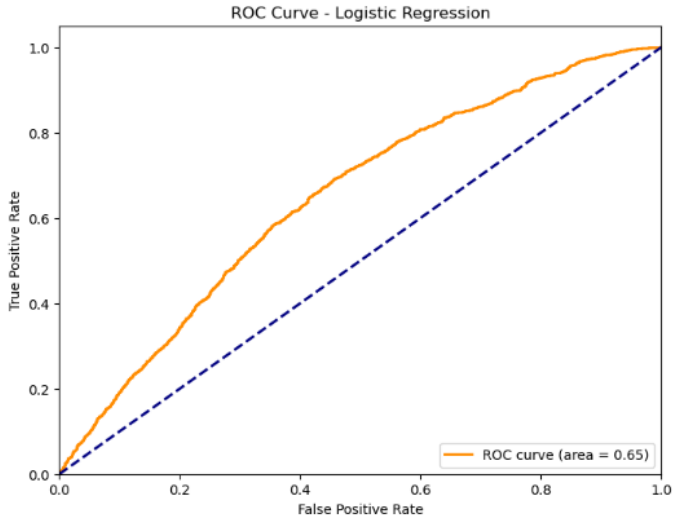


Fig. 29. Logistic Regression ROC curve

```
In [25]: # Print additional metrics
         print("\nDetailed Metrics:")
         print(f"ROC AUC Score: {roc_auc_lr:.3f}")
         print(f"Accuracy: {accuracy_score(y_test, y_pred_lr):.3f}")
         print(f"Precision: {precision_score(y_test, y_pred_lr):.3f}")
         print(f"Recall: {recall_score(y_test, y_pred_lr):.3f}")
         print(f"F1 Score: {f1_score(y_test, y_pred_lr):.3f}")
```

```
Detailed Metrics:
ROC AUC Score: 0.647
Accuracy: 0.555
Precision: 0.290
Recall: 0.716
F1 Score: 0.413
```

Fig. 30. Logistic regression detailed metrics

data points. random-state=42 ensures that the shuffling is reproducible.

- The list of metrics includes 'accuracy', 'precision', 'recall', and 'f1', though only accuracy is computed in this section. The others may be used later.
- lr-grid.best-estimator- refers to the best Logistic Regression model found during a grid search or hyperparameter tuning.
- cross-val-score calculates the accuracy of the model on each fold of the training data. The results are stored in lr-scores.
- For the Random Forest model. rf-grid.best-estimator- indicates the best-performing Random Forest model.
- Again, cross-validation is performed using the same folds, and the results (mean, standard deviation, and individual

```
In [26]: # Random Forest with Grid Search
rf_params = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20],
    'class_weight': ['balanced']
}
rf_model = RandomForestClassifier(random_state=42)
rf_grid = GridSearchCV(rf_model, rf_params, cv=5, scoring='roc_auc')
rf_grid.fit(X_train_rf, y_train)
print("Best Random Forest parameters:", rf_grid.best_params_)

Best Random Forest parameters: {'class_weight': 'balanced', 'max_depth': 10, 'n_estimators': 200}

In [27]: # Get predictions
y_pred_rf = rf_grid.predict(X_test_rf)
y_prob_rf = rf_grid.predict_proba(X_test_rf)[:, 1]

# Print classifier results
print("\nRandom Forest Results:")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))


Random Forest Results:

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.84      0.86      4687
           1       0.50      0.57      0.53      1313

    accuracy                           0.78      6000
   macro avg       0.69      0.70      0.69      6000
weighted avg       0.79      0.78      0.79      6000

In [28]: print("\nConfusion Matrix:")
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
print(conf_matrix_rf)


Confusion Matrix:
[[3941  746]
 [ 571  742]]
```

Fig. 31. Random forest classification report and confusion matrix

```
In [29]: # Calculate and plot ROC curve
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_prob_rf)
roc_auc_rf = roc_auc_score(y_test, y_prob_rf)

plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, color='darkorange', lw=2,
         label=f'ROC curve (area = {roc_auc_rf:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend(loc="lower right")
plt.show()
```

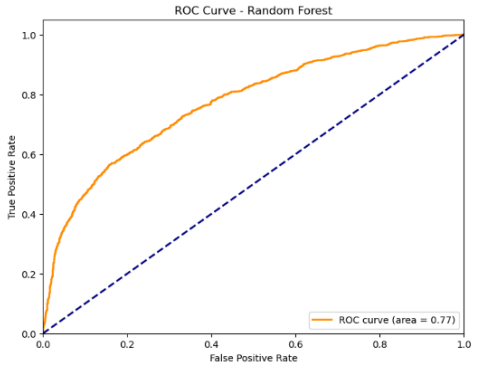
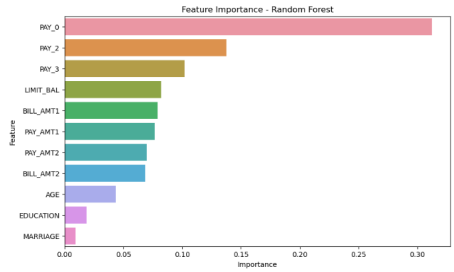


Fig. 32. Random forest ROC curve

```
In [30]: # Print additional metrics
print("\nDetailed Metrics:")
print(f"ROC AUC Score: {roc_auc_rf:.3f}")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf):.3f}")
print(f"Precision: {precision_score(y_test, y_pred_rf):.3f}")
print(f"Recall: {recall_score(y_test, y_pred_rf):.3f}")
print(f"F1 Score: {f1_score(y_test, y_pred_rf):.3f}")


Detailed Metrics:
ROC AUC Score: 0.772
Accuracy: 0.780
Precision: 0.499
Recall: 0.565
F1 Score: 0.530
```

Fig. 33. Random forest Detailed metrics

```
In [32]: # Feature Importance Plot
feature_importance = pd.DataFrame({
    'feature': selected_features,
    'importance': rf_grid.best_estimator_.feature_importances_
}).sort_values('importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance)
plt.title('Feature Importance - Random Forest')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()

print("\nFeature Importance:")
print(feature_importance)
```




```
Feature Importance:
      feature  importance
2       PAY_0    0.312431
3       PAY_2    0.137697
```

Fig. 34. Random forest feature importance plot

```
In [154]: # 6. Handling Data Imbalance
print("\n=== Class Distribution ===")
print(y_train.value_counts(normalize=True))


=== Class Distribution ===
0    0.778208
1    0.221792
Name: default payment next month, dtype: float64
```

Fig. 35. Handling data imbalance

scores) are stored in the cv-results dictionary.

1) Results of N-cross validation: The results for the N-cross validation is generated and it shows that random forest is having best performance.


```
In [154]: # 6. Handling Data Imbalance
print("\n=== Class Distribution ===")
print(y_train.value_counts(normalize=True))


=== Class Distribution ===
0    0.778208
1    0.221792
Name: default payment next month, dtype: float64
```


Fig. 36. Handling data imbalance

```
# Print results
print("\nCross-Validation Results (5-fold):")
for model_name in cv_results:
    print(f"\n{model_name}:")
    print(f"Accuracy Mean: {cv_results[model_name]['accuracy']['mean']:.3f}")
    print(f"Accuracy Std: {cv_results[model_name]['accuracy']['std']:.3f}")
    print(f"Individual fold scores: {cv_results[model_name]['accuracy']['scores']}")


Cross-Validation Results (5-fold):

Logistic Regression:
Accuracy Mean: 0.549
Accuracy Std: 0.007
Individual fold scores: [0.55708333 0.54729167 0.535625   0.55166667 0.55208333]

Random Forest:
Accuracy Mean: 0.787
Accuracy Std: 0.005
Individual fold scores: [0.78229167 0.79       0.78395833 0.78291667 0.79375   ]
```

Fig. 37. N-cross validation output

M. Saving the model

saving the model

```
In [80]: # Save model and preprocessing objects
print("\n=== Saving Model ===")
import joblib
import os

# Creating models directory if it doesn't exist
os.makedirs('models', exist_ok=True)

# Creating dictionary with model and necessary objects
model_data = {
    'model': rf_grid.best_estimator_,  #as random forest is the best performing model
    'features': selected_features,
    'scaler': scaler
}

# Saving to models folder using simpler path
try:
    joblib.dump(model_data, 'models/credit_default_model.joblib')
    print("Model saved successfully!")
    print("Model saved at:", os.path.abspath('models/credit_default_model.joblib'))
except Exception as e:
    print(f"Error saving model: {str(e)}")

=== Saving Model ===
Model saved successfully!
Model saved at: C:\Users\palug\models\credit_default_model.joblib
```

Fig. 38. saving model

The model is saved to easily generate the gradio interface.

N. Loading the saved model

```
In [33]: # Load the saved model and preprocessing objects
model_data = joblib.load('credit_default_model.joblib')
model = model_data['model']
scaler = model_data['scaler']
label_encoder = model_data['label_encoder']
selected_features = model_data['features']

def predict_default(limit_bal, age, pay_0, pay_2, pay_3,
                    bill_amt1, bill_amt2, pay_amt1, pay_amt2,
                    education, marriage):
    try:
        # Create input data dictionary
        input_dict = {
            'LIMIT_BAL': float(limit_bal),
            'AGE': float(age),
            'PAY_0': float(pay_0),
            'PAY_2': float(pay_2),
            'PAY_3': float(pay_3),
            'BILL_AMT1': float(bill_amt1),
            'BILL_AMT2': float(bill_amt2),
            'PAY_AMT1': float(pay_amt1),
            'PAY_AMT2': float(pay_amt2),
            'EDUCATION': int(education),
            'MARRIAGE': int(marriage)
        }

        # Create DataFrame with the same feature order as training
        input_df = pd.DataFrame([input_dict])[selected_features]

        # Scale numeric features
        numeric_features = ['LIMIT_BAL', 'AGE', 'BILL_AMT1', 'BILL_AMT2', 'PAY_AMT1', 'PAY_AMT2']
        input_df[numeric_features] = scaler.transform(input_df[numeric_features])

        # Encode categorical features (for Random Forest, we use label encoding)
        categorical_features = ['EDUCATION', 'MARRIAGE']
        for col in categorical_features:
            input_df[col] = label_encoder.transform(input_df[col])

        # Make prediction
        prediction = model.predict(input_df)[0]
        probability = model.predict_proba(input_df)[0][1]

        # Format result
        if prediction == 1:
            result = f"""
            High Risk of Default

            Probability of default: {probability:.2%}

            Input Details:
            - Credit Limit: {limit_bal}
            - Age: {age}
            - Recent Payment Status: {pay_0}
            - Bill Amount 1: {bill_amt1}
```

Fig. 39. Loading saved model

For creating the gradio interface, the model is loaded first.

O. Creating the Gradio Interface

Now, after loading the model, the gradio interface is created. When user enters the selected features values, the interface will say whether the client will be in default or not.

We have also added some guidelines to easily enter the data.

1) Gradio Interface output: The above figure shows the gradio interface output, where it displays the link for the app and also the app.

```
            result = f"""
            Low Risk of Default

            Probability of default: {probability:.2%}

            Input Details:
            - Credit Limit: {limit_bal}
            - Age: {age}
            - Recent Payment Status: {pay_0}
            - Bill Amount 1: {bill_amt1}
            - Payment Amount 1: {pay_amt1}
            """

        return result

    except Exception as e:
        return f"Error in prediction: {str(e)}"

# Create Gradio interface
demo = gr.Interface(
    fn=predict_default,
    inputs=[
        gr.Number(label="Credit Limit Balance"),
        gr.Number(label="Age"),
        gr.Number(label="Payment Status Month 0 (-1=early, 0=on time, 1-9=delay)"),
        gr.Number(label="Payment Status Month 2"),
        gr.Number(label="Payment Status Month 3"),
        gr.Number(label="Bill Amount 1"),
        gr.Number(label="Bill Amount 2"),
        gr.Number(label="Payment Amount 1"),
        gr.Number(label="Payment Amount 2"),
        gr.Number(label="Education (1=graduate, 2=university, 3=high school, 4=others)"),
        gr.Number(label="Marriage (1=married, 2=single, 3=others)")
    ],
    outputs=gr.Textbox(label="Default Prediction Result"),
    title="Credit Card Default Predictor",
    description="""
    Enter client information to predict credit card default probability.

    Guidelines:
    - Payment Status: -1 means paid early, 0 means paid on time, 1-9 means delay of 1-9 months
    - Education: 1=graduate school, 2=university, 3=high school, 4=others
    - Marriage: 1=married, 2=single, 3=others

    Example case: Credit limit=20000, Age=24, Pay_0=2, Pay_2=2, Pay_3=-1,
                  Bill_Amt1=3913, Bill_Amt2=3102, Pay_Amt1=689, Pay_Amt2=0,
                  Education=2, Marriage=1
    """,
    examples=[
        # Example from your dataset
        [20000, 24, 2, 2, -1, 3913, 3102, 689, 0, 2, 1]
    ],
    theme="default"
)
```

Fig. 40. Creating gradio interface

```
* Running on local URL:  http://127.0.0.1:7861

To create a public link, set `share=True` in `launch()`.
```

Credit Card Default Predictor

Enter client information to predict credit card default probability.

Guidelines:

- Payment Status: -1 means paid early, 0 means paid on time, 1-9 means delay of 1-9 months
- Education: 1=graduate school, 2=university, 3=high school, 4=others
- Marriage: 1=married, 2=single, 3=others

Example case: Credit limit=20000, Age=24, Pay_0=2, Pay_2=2, Pay_3=-1, Bill_Amt1=3913, Bill_Amt2=3102, Pay_Amt1=689, Pay_Amt2=0, Education=2, Marriage=1

Credit Limit Balance | Default Prediction Result | Age | Payment Status Month 0 (-1=early, 0=on time, 1- | Flag

Fig. 41. Gradio interface output

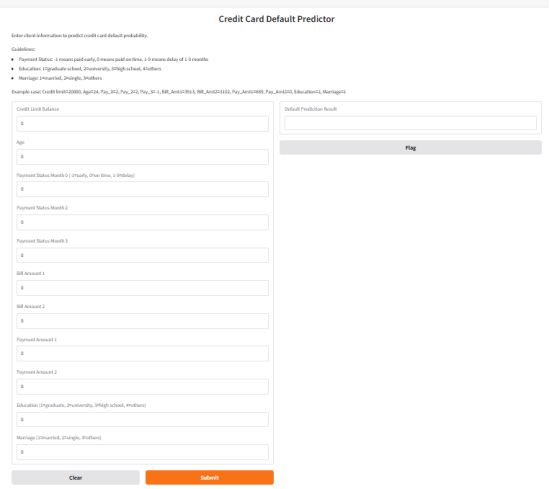


Fig. 42. Gradio interface

P. Gradio interface app

The figure shows the final gradio interface app, which is generated. After running the code, in the output a link will be generated. When the client clicks on the link, it direstly takes the client to the app.

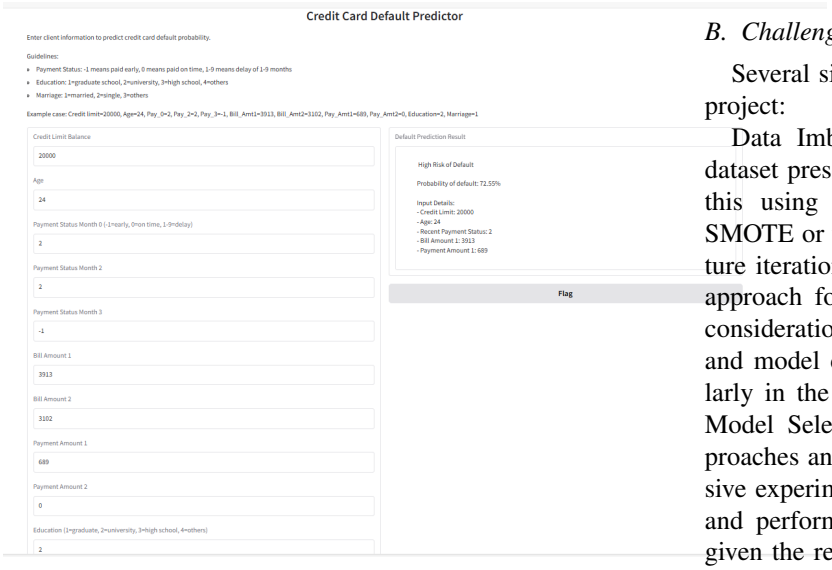


Fig. 43. Gradio interface with example

1) Running of Gradio interface app: After running the code, in the output a link will be generated. When the client clicks on the link, it direstly takes the client to the app. Now as an example when we entered the details fromm the dataset, the prediction result is shown as the client is at high risk of default.

VI. Conclusion

A. Lessons Learned

Throughout this credit card default prediction project, several valuable insights and lessons emerged from our analysis and modeling efforts. The implementation of machine learning models for credit default prediction revealed the critical importance of proper data preprocessing and feature engineering. We learned that handling class imbalance through appropriate weighting mechanisms was crucial for model performance, as demonstrated by the significant improvement in prediction accuracy after implementing balanced class weights.

One of the most significant insights was the importance of feature selection and encoding strategies. The comparative analysis between one-hot encoding for logistic regression and label encoding for random forest classifier highlighted how different models require different preprocessing approaches. The effectiveness of the random forest model in handling categorical variables with simple label encoding, while logistic regression required more complex one-hot encoding, demonstrated the importance of understanding model-specific requirements.

The visualization phase of our analysis proved invaluable in understanding the underlying patterns in the data. The clear correlation patterns revealed in our heatmap analysis guided our feature selection process, while the payment status trends provided crucial insights into customer behavior patterns. These visualizations not only improved our understanding of the data but also helped in making informed decisions about feature engineering and model selection.

B. Challenges Encountered

Several significant challenges were encountered during this project:

Data Imbalance: The substantial class imbalance in the dataset presented a significant challenge. While we addressed this using class weights, exploring other techniques like SMOTE or under-sampling might yield different results in future iterations. Feature Engineering: Determining the optimal approach for handling categorical variables required careful consideration. The tradeoff between information preservation and model complexity had to be carefully balanced, particularly in the case of education and marriage status variables. Model Selection: Choosing between different modeling approaches and their respective hyperparameters required extensive experimentation. The balance between model complexity and performance had to be carefully considered, especially given the real-world nature of the problem.

C. Future Considerations

Several promising directions for future work have been identified:

Advanced Feature Engineering:

Creation of interaction terms between correlated features Development of more sophisticated payment behavior metrics Investigation of non-linear transformations for numerical features

Model Enhancements:

Exploration of ensemble methods combining logistic regression and random forest predictions Implementation of more advanced algorithms such as XGBoost or LightGBM Investigation of deep learning approaches for feature extraction

Real-world Application Considerations:

Development of more robust validation strategies Implementation of model monitoring systems Investigation of model interpretability techniques for business stakeholders

Data Enhancement:

Collection of additional relevant features Integration of macroeconomic indicators Exploration of temporal aspects in customer behavior

D. Practical Implications

The results of this study have several practical implications for credit risk assessment:

Model Deployment: The successful implementation of the Gradio interface demonstrates the potential for deploying such models in real-world applications. However, careful consideration must be given to model maintenance and updating procedures. Risk Assessment: The feature importance analysis provides valuable insights for credit officers and risk managers

about which factors most strongly indicate default risk. This information can be used to improve existing credit assessment procedures. Customer Relationship Management: The patterns identified in payment behavior could be used to develop early warning systems for potential defaults, allowing for proactive customer engagement.

In conclusion, while our models showed promising results in predicting credit card defaults, there remains significant potential for improvement and refinement. The lessons learned from this project provide a solid foundation for future work in credit risk assessment and demonstrate the value of machine learning approaches in financial decision-making. The challenges encountered and solutions developed contribute to the growing body of knowledge in this important field, while the identified future considerations provide clear directions for continuing research and development.

VII. Appendix A

The Jupyter Notebook with corresponding Python code used to complete this report can be found using the following URL:

https://github.com/nikhithap20/Credit-Card-Default-Prediction.git

“

References

[1] Sheikh Rabiul Islam, W. Eberle, S. Ghafoor, “Credit Default Mining Using Combined Machine Learning and Heuristic Approach,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, 2 July 2018.

[2] gradio A. Abid et al., “Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild,” *arXiv preprint arXiv:2008.06632*, 2020.

[3] gradio https://youtu.be/53x330upumY?si=cb2bODoDmOXHfIN8

[4] C. Yeh and C. H. Lien,“The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients,” *Expert Systems with Applications*, vol. 36, no. 2, pp. 2473-2480, 2009.