VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF ECONOMICS AND LAW

REPORT: FINAL PROJECT
CREDIT EVALUATION

Instructor: Lê Thị Thanh An
Group : 5
Class : K22413CA

HO CHI MINH CITY - 2025

Table of Contents

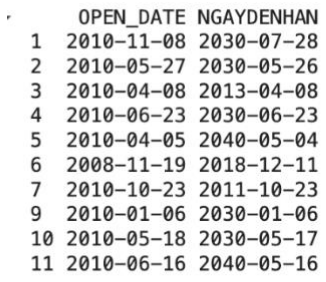
I.DATA EXPLORATION AND PREPARATION

1. Loading the data

The original dataset was loaded using pandas, and a preliminary inspection was performed to identify missing values and incorrect data types.

2. Check and fix the wrong data format type

2.1. Date format



```
    OPEN_DATE  NGAYDENHAN
1   2010-11-08 2030-07-28
2   2010-05-27 2030-05-26
3   2010-04-08 2013-04-08
4   2010-06-23 2030-06-23
5   2010-04-05 2040-05-04
6   2008-11-19 2018-12-11
7   2010-10-23 2011-10-23
9   2010-01-06 2030-01-06
10  2010-05-18 2030-05-17
11  2010-06-16 2040-05-16
```

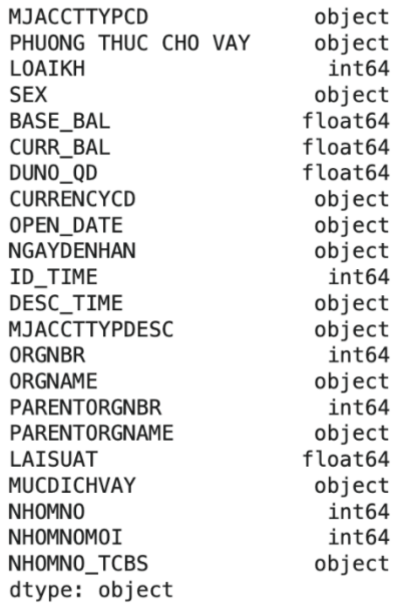
Figure 1.1: Sample output of standardized date columns

After reviewing the initial data, the columns OPEN_DATE and NGAYDENHAN were found to contain inconsistent date formats (Excel serial numbers and string dates). A custom function was applied to standardize all values to the datetime format. The output below confirms that both columns have been successfully converted to the correct date format. This step is crucial to ensure reliable date-based operations such as calculating loan terms or maturity intervals in subsequent analyses.

2.2. Date type

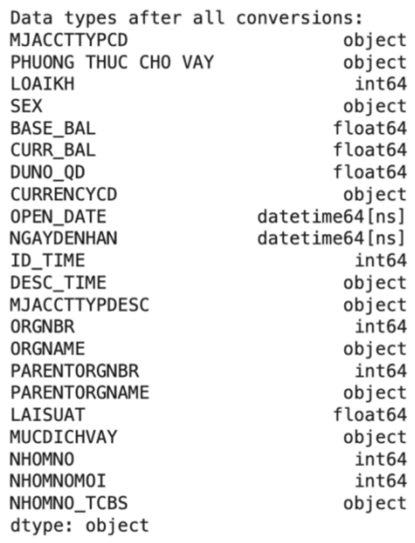
Before performing data analysis, it is important to ensure that all variables are stored in appropriate data types. Incorrect data types can cause errors in subsequent processing and modeling steps.

In this step, we review all columns, then explicitly convert key numerical fields to the correct format. This helps maintain consistency throughout the data pipeline.



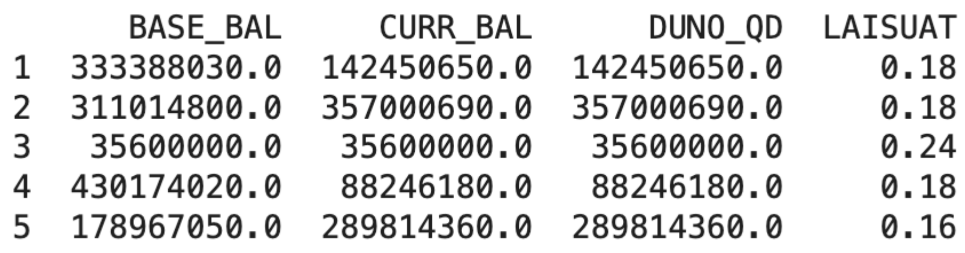
```
MJACCTTYPCD            object
PHUONG THUC CHO VAY    object
LOAIKH                  int64
SEX                    object
BASE_BAL              float64
CURR_BAL              float64
DUNO_QD               float64
CURRENCYCD             object
OPEN_DATE              object
NGAYDENHAN             object
ID_TIME                 int64
DESC_TIME              object
MJACCTTYPDESC          object
ORGNBR                  int64
ORGNAME                object
PARENTORGNBR            int64
PARENTORGNAME          object
LAISUAT               float64
MUCDICHVAY             object
NHOMNO                  int64
NHOMNOMOI               int64
NHOMNO_TCBS            object
dtype: object
```

Figure 2.2.1: Data types before conversion



```
Data types after all conversions:
MJACCTTYPCD                  object
PHUONG THUC CHO VAY          object
LOAIKH                        int64
SEX                          object
BASE_BAL                    float64
CURR_BAL                    float64
DUNO_QD                     float64
CURRENCYCD                   object
OPEN_DATE            datetime64[ns]
NGAYDENHAN           datetime64[ns]
ID_TIME                       int64
DESC_TIME                    object
MJACCTTYPDESC                object
ORGNBR                        int64
ORGNAME                      object
PARENTORGNBR                  int64
PARENTORGNAME                object
LAISUAT                     float64
MUCDICHVAY                   object
NHOMNO                        int64
NHOMNOMOI                     int64
NHOMNO_TCBS                  object
dtype: object
```

Figure 2.2.2: Data types after conversion

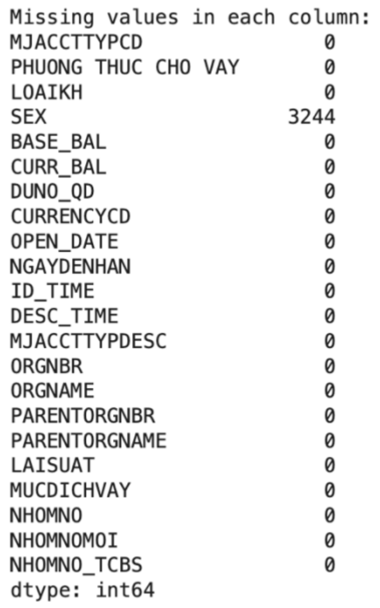


```
      BASE_BAL     CURR_BAL      DUNO_QD  LAISUAT
1  333388030.0  142450650.0  142450650.0     0.18
2  311014800.0  357000690.0  357000690.0     0.18
3   35600000.0   35600000.0   35600000.0     0.24
4  430174020.0   88246180.0   88246180.0     0.18
5  178967050.0  289814360.0  289814360.0     0.16
```

Figure 2.2.3: Sample numerical data after conversion

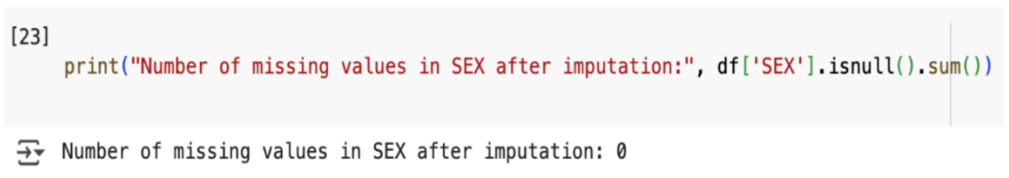
The data types before conversion show that several columns were not in the correct format, with dates and numeric fields stored as object type. After conversion, all columns have the correct types: date fields as **datetime64[ns]** and numeric fields as **float64**. Figure 2.2.3 further demonstrates that all values in numeric columns are valid floats, with no missing or erroneous entries.

3. Missing value



```
Missing values in each column:
MJACCTTYPCD              0
PHUONG THUC CHO VAY      0
LOAIKH                   0
SEX                   3244
BASE_BAL                 0
CURR_BAL                 0
DUNO_QD                  0
CURRENCYCD               0
OPEN_DATE                0
NGAYDENHAN               0
ID_TIME                  0
DESC_TIME                0
MJACCTTYPDESC            0
ORGNBR                   0
ORGNAME                  0
PARENTORGNBR             0
PARENTORGNAME            0
LAISUAT                  0
MUCDICHVAY               0
NHOMNO                   0
NHOMNOMOI                0
NHOMNO_TCBS              0
dtype: int64
```

Figure 2.3.1: Number of missing values in SEX before imputation



```
[23]
    print("Number of missing values in SEX after imputation:", df['SEX'].isnull().sum())
```

```
Number of missing values in SEX after imputation: 0
```

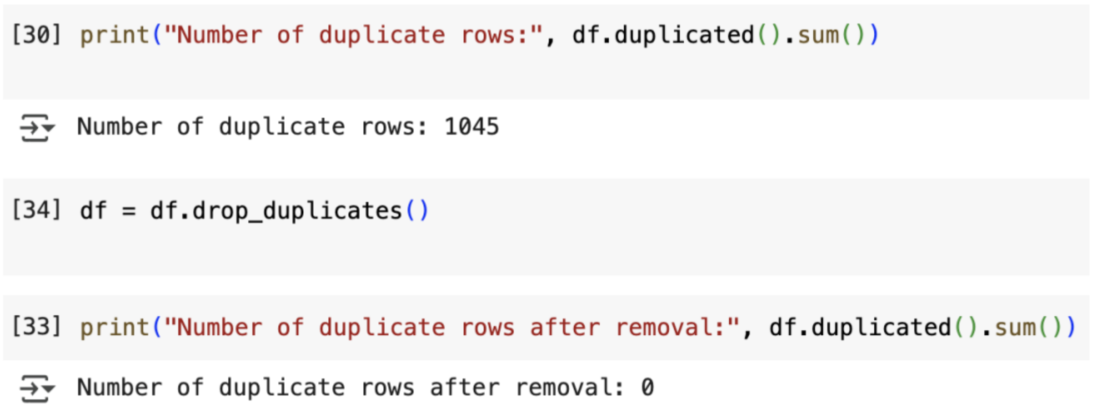
Figure 2.3.2: Number of missing values in SEX after imputation

Column SEX contained 3244 missing values. We used mode imputation to fill missing entries with the most frequent value. This method retains all rows in the dataset and is commonly used for categorical variables. After imputation, no missing values remained in the SEX column.

4. Duplicate

Initially, the dataset contained 1,045 duplicate rows. After using the **drop_duplicates ()** function, there were no duplicate rows remaining in the dataset.

Code and results are shown below:



```
[30] print("Number of duplicate rows:", df.duplicated().sum())
```

```
Number of duplicate rows: 1045
```

```
[34] df = df.drop_duplicates()
```

```
[33] print("Number of duplicate rows after removal:", df.duplicated().sum())
```

```
Number of duplicate rows after removal: 0
```

5. Outlier and extreme value detection

We performed outlier and extreme value detection on the key continuous numerical features: **BASE_BAL**, **CURR_BAL**, and **DUNO_QD**. These variables represent major financial amounts where unusual or extreme values can significantly impact further analysis and modeling.

Outliers and extreme values were detected using the **Z-score method**. Observations with an absolute Z-score greater than 3 were considered as outliers/extreme values and were subsequently removed from the dataset. This process helps ensure data quality by reducing the impact of anomalous records.

A total of 282 records were identified as outliers/extreme values and removed. After this step, the dataset retained only records with valid, representative values for these key financial variables.

As illustrated in the box plots below, removing outliers and extreme values resulted in more compact and representative distributions for each variable.

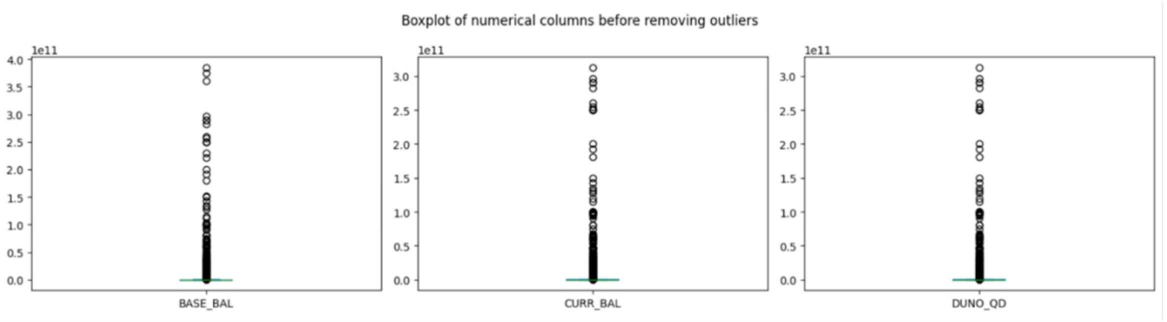


Figure 5.1:Boxplot of BASE_BAL, CURR_BAL, and DUNO_QD before outlier removal

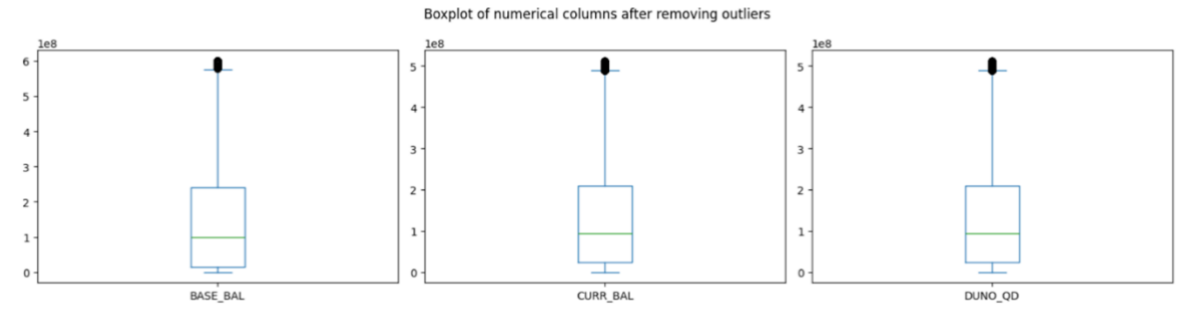


Figure 5.2: Boxplot of BASE_BAL, CURR_BAL, and DUNO_QD after outlier removal

6. Unbalanced data

The initial distribution of the target variable is highly imbalanced, with a significant majority of samples belonging to class 1 (best), and only a small fraction of observations in classes 2, 3, 4, and 5 (see Figure 6.1, left panel). Such severe imbalance can bias the classifier towards the dominant class and significantly reduce its ability to recognize minority classes, which is particularly problematic in credit risk modeling.

To address this challenge, we applied the **SMOTE (Synthetic Minority Over-sampling Technique)** algorithm. SMOTE generates synthetic examples for minority classes by interpolating between neighboring samples in feature space, resulting in a balanced dataset where each of the five classes contains an equal number of records (see Figure 6.1, right panel).

By balancing the target distribution, we help the model avoid a bias towards the majority class and enhance its ability to accurately classify customers into all five credit groups. All subsequent modeling steps are performed on the SMOTE-balanced training data, while the test and validation sets remain untouched for unbiased performance evaluation.

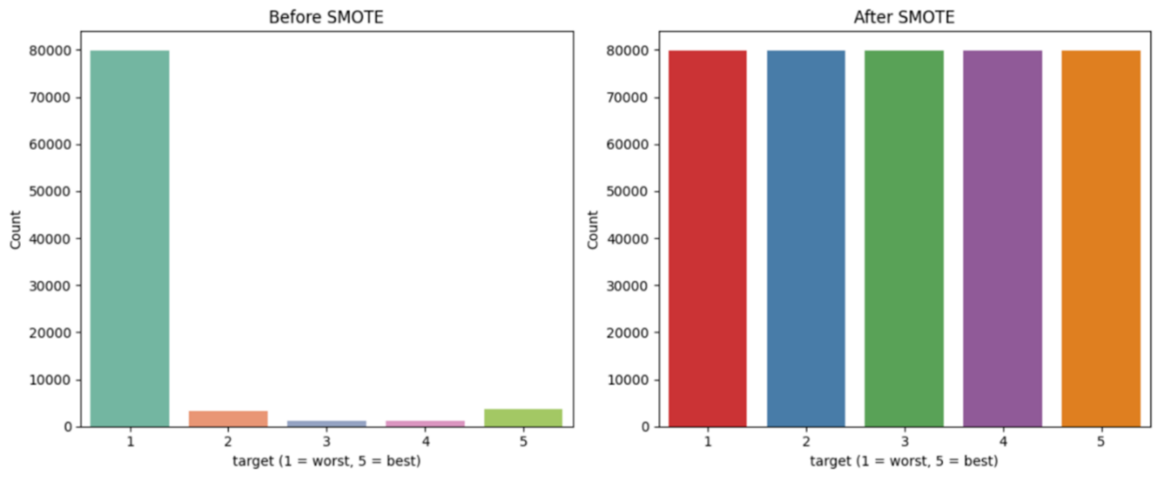


Figure 6.1: Class distribution before SMOTE (left) and after SMOTE (right).

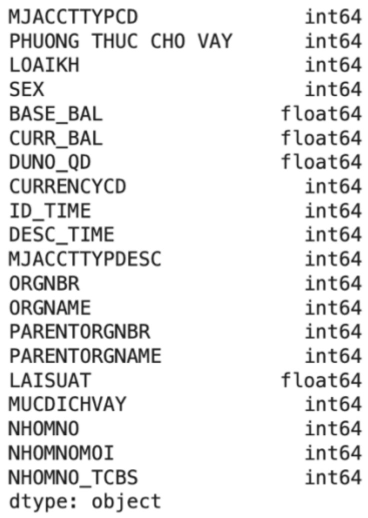
7. Encode the data

To prepare the dataset for machine learning models, all categorical features were encoded into numeric values. First, we identified columns with the data type object using X.select_dtypes(include=['object']). These columns typically contain categorical or string data.

Since our categorical features contain a moderate number of unique values, we used **Label encoding** to convert each category to a unique integer. This method is efficient and prevents the dataset from becoming excessively wide, as would occur with One-Hot Encoding when there are many unique categories.

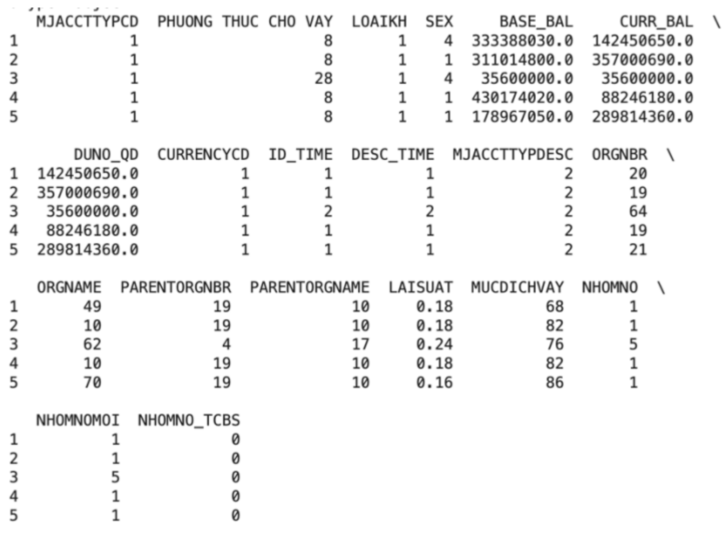
After encoding, all features are numeric, as shown below:

- All columns are of type **int64** or **float64**.
- No **object** type columns remain.
- This ensures that our data is compatible with a wide range of machine learning algorithms.



```
MJACCTTYPCD             int64
PHUONG THUC CHO VAY     int64
LOAIKH                  int64
SEX                     int64
BASE_BAL              float64
CURR_BAL              float64
DUNO_QD               float64
CURRENCYCD              int64
ID_TIME                 int64
DESC_TIME               int64
MJACCTTYPDESC           int64
ORGNBR                  int64
ORGNAME                 int64
PARENTORGNBR            int64
PARENTORGNAME           int64
LAISUAT               float64
MUCDICHVAY              int64
NHOMNO                  int64
NHOMNOMOI               int64
NHOMNO_TCBS             int64
dtype: object
```

Figure 7.1: Feature data types and sample records after categorical encoding



```
   MJACCTTYPCD  PHUONG THUC CHO VAY  LOAIKH  SEX     BASE_BAL     CURR_BAL  \
1            1                    8       1    4  333388030.0  142450650.0
2            1                    8       1    1  311014800.0  357000690.0
3            1                   28       1    4   35600000.0   35600000.0
4            1                    8       1    1  430174020.0   88246180.0
5            1                    8       1    1  178967050.0  289814360.0

       DUNO_QD  CURRENCYCD  ID_TIME  DESC_TIME  MJACCTTYPDESC  ORGNBR  \
1  142450650.0           1        1          1              2      20
2  357000690.0           1        1          1              2      19
3   35600000.0           1        2          2              2      64
4   88246180.0           1        1          1              2      19
5  289814360.0           1        1          1              2      21

   ORGNAME  PARENTORGNBR  PARENTORGNAME  LAISUAT  MUCDICHVAY  NHOMNO  \
1       49            19             10     0.18          68       1
2       10            19             10     0.18          82       1
3       62             4             17     0.24          76       5
4       10            19             10     0.18          82       1
5       70            19             10     0.16          86       1

   NHOMNOMOI  NHOMNO_TCBS
1          1            0
2          1            0
3          5            0
4          1            0
5          1            0
```

Figure 7.2: Dataset preview after label encoding – all columns are numeric

8. Correlation

To ensure robust modeling and avoid issues with multicollinearity, we analyzed the correlation matrix of all features after encoding. Using Pearson correlation, we plotted a heatmap to visualize relationships between variables (see Figure 8.1). "We used a threshold of 0.8, as correlations above this value often indicate problematic multicollinearity that can affect model reliability. In our dataset, the columns **'CURR_BAL', 'DUNO_QD',** and **'NHOMNOMOI**' were dropped as they were strongly correlated with others. After this process, the remaining features had no absolute correlation values exceeding the chosen threshold. This step helps stabilize subsequent models and improves overall interpretability. The code snippet below illustrates the procedure:

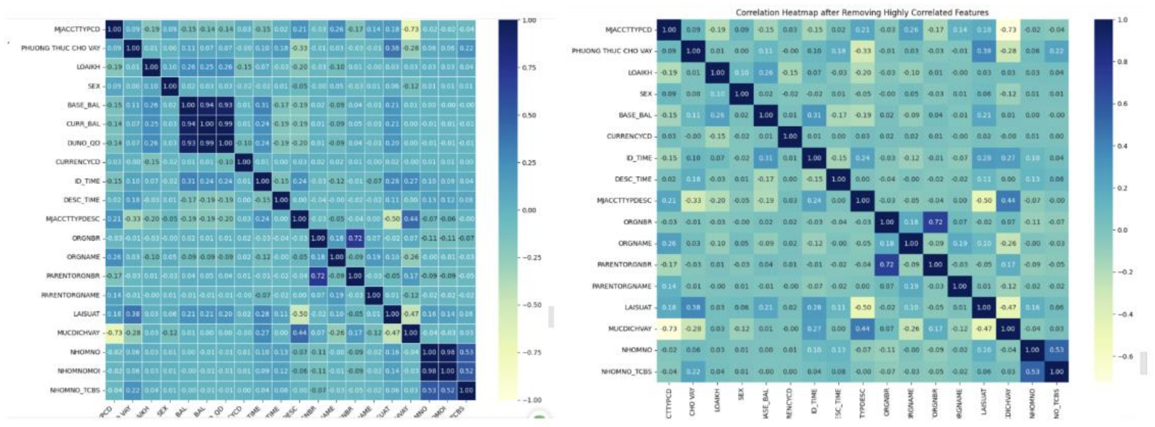


Figure 8.1: Correlation matrix heatmap before and after removing highly correlated features

As shown in Figure 8.1, after removing the highly correlated variables, no pair of features has an absolute correlation greater than the threshold, confirming the effectiveness of the procedure.

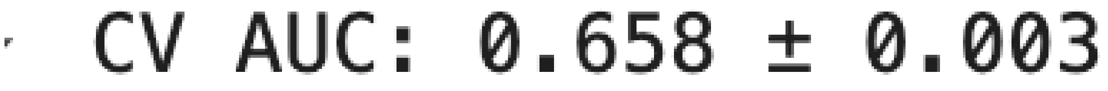
9. Feature engineering

To ensure the model receives well-structured and meaningful inputs, we performed systematic feature engineering prior to model training. First, categorical features such as SEX and MJACCTTYPEDESC were identified and encoded using appropriate techniques (e.g., label encoding or one-hot encoding as required by the model). Continuous variables (LAISUAT, BASE_BAL, ID_TIME, etc.) were standardized to have zero mean and unit variance, ensuring that features with larger scales do not disproportionately influence the model.

All preprocessing steps, including encoding and scaling, were encapsulated within a scikit-learn pipeline to ensure reproducibility and consistent application throughout cross-validation and final predictions.

After applying these transformations, we evaluated the baseline model performance using 5-fold cross-validation. The Random Forest classifier was trained on the engineered features, and performance was assessed using the macro-averaged ROC AUC score for multi-class classification.

The results yielded an average cross-validation AUC of **0.658 ± 0.003** (see Figure 9.1). While this score is moderate, it establishes a baseline for subsequent experiments. The result indicates that the engineered features provide some discriminatory power across the five classes, but there is significant room for improvement, which will be explored by testing more advanced classification models in the next steps.



```
CV AUC: 0.658 ± 0.003
```

Figure 9.1: Cross-validation AUC after feature engineering.

A relatively low AUC may be attributed to overlapping features among classes or limited information in the available features, which will be addressed in subsequent model improvements.

II.MODEL SELECTION, IMPLEMENTATION, AND EVALUATION:

1. Model Selection via Stratified K-Fold Cross-Validation with SMOTE

1.1. Traditional model: Logistic Regression

Logistic Regression is a traditional linear classification model that is widely used for both binary and multiclass problems. In our project, we applied Logistic Regression to a multiclass classification task with five levels of credit risk (from 1 to 5), where we encountered a significant class imbalance issue.

To handle this, we implemented a 5-fold cross-validation framework combined with SMOTE (Synthetic Minority Over-sampling Technique) inside each fold. This method helps balance the dataset during training without causing data leakage. We evaluated model performance using accuracy and F1 Macro, with F1 Macro being more suitable in imbalanced multiclass settings.

Fold 1: Accuracy: 0.4302 | F1 Macro: 0.1927

Fold 2: Accuracy: 0.4196 | F1 Macro: 0.1903

Fold 3: Accuracy: 0.4290 | F1 Macro: 0.1923

Fold 4: Accuracy: 0.4259 | F1 Macro: 0.1914

Fold 5: Accuracy: 0.413 | F1 Macro: 0.1830

Despite applying SMOTE and cross-validation, our Logistic Regression model achieved only moderate performance, with an average accuracy of approximately 42.4% and a mean F1 Macro score of 0.1900. This indicates that the model struggled to correctly classify minority classes, especially classes 1 and 5, which were severely underrepresented.

Since Logistic Regression is a linear model, it might not capture the complex patterns present in our dataset. As a result, while it performed consistently, it was not selected as one of the top-performing models for final training and evaluation.

1.2 Non traditional model: CatBoost Classifier

CatBoost is a gradient boosting algorithm that handles categorical features efficiently and is well-suited for structured data. In our project, we used CatBoost for a multiclass classification task involving five classes of credit risk. Given the imbalanced distribution of classes particularly in classes 1 and 5we incorporated SMOTE within each fold of a 5-fold cross-validation to improve class representation and reduce bias during training.

Evaluation was performed using both accuracy and F1 Macro score, which allows us to assess how well the model handles all classes, especially the minority ones.

Fold 1:Accuracy = 0.8434 | F1 Macro = 0.4247

Fold 2: Accuracy = 0.8474 | F1 Macro = 0.4226

Fold 3: Accuracy = 0.8387 | F1 Macro = 0.4048

Fold 4: Accuracy = 0.8354 | F1 Macro = 0.4067

Fold 5: Accuracy = 0.8366 | F1 Macro = 0.4202

Accuracy: 0.8403 ± 0.0045

F1 Macro: 0.4158 ± 0.0083

CatBoost delivered strong and consistent performance across all five folds. With an average accuracy of approximately 84.0% and a mean F1 Macro of 0.4158, this model demonstrated an ability to classify both majority and minority classes more effectively than Logistic Regression.

The relatively low standard deviation across folds suggests stable generalization. Thanks to the combination of CatBoost's gradient boosting mechanism and our use of SMOTE within K-Fold, the model handled the class imbalance reasonably well. Based on this performance, CatBoost was selected as one of our top three models for further training and final evaluation.

1.3. Non traditional model: Random Forest

Random Forest is an ensemble learning method based on decision trees, which is particularly effective for classification tasks involving complex, non-linear relationships. In our project, we applied Random Forest to a multiclass credit risk classification problem with five classes. To address the severe class imbalance, we used a combination of:

+ SMOTE (to oversample minority classes within each fold),

+ Class weights (to penalize the majority classes during model training), and 5-fold cross-validation (to ensure robust and generalizable results).

We evaluated the model using accuracy and F1 Macro, with a particular focus on the latter due to the imbalanced nature of the dataset.

Fold 1: Accuracy = 0.8617 | F1 Macro = 0.4147

Fold 2: Accuracy = 0.8668 | F1 Macro = 0.4104

Fold 3: Accuracy = 0.8640 | F1 Macro = 0.4085

Fold 4: Accuracy = 0.8606 | F1 Macro = 0.4026

Fold 5: Accuracy = 0.8649 | F1 Macro = 0.4178

Accuracy: 0.8636 ± 0.0022

F1 Macro: 0.4108 ± 0.0052

The Random Forest model delivered consistently high performance across all folds. With an average accuracy of 86.4% and a mean F1 Macro score of 0.4108, it demonstrated the ability to capture complex patterns and provide balanced predictions across all classes, including the underrepresented ones.

Thanks to the integration of SMOTE and class weighting, the model effectively mitigated the class imbalance issue. Furthermore, the low standard deviation in both metrics indicates stable generalization. Given its strong and reliable performance, Random Forest was selected as one of our top three models for final training and prediction on the full dataset.

1.4 Non traditional model: Neutral Network

In this project, we also implemented a Neural Network model for the multiclass credit risk classification task. Neural Networks are powerful in capturing non-linear relationships and complex patterns within the data. However, they require a sufficient amount of well-balanced data to perform effectively.

To deal with the severe class imbalance in our dataset, we applied SMOTE within each fold of a 5-fold cross-validation framework. This approach helps oversample the minority classes only within the training portion of each fold, avoiding data leakage and ensuring fair evaluation.

As with other models, we evaluated the performance using both accuracy and F1 Macro.

Fold 1: Accuracy = 0.5913 | F1 Macro = 0.3020

Fold 2: Accuracy = 0.5093 | F1 Macro = 0.2791

Fold 3: Accuracy = 0.5394 | F1 Macro = 0.2696

Fold 4: Accuracy = 0.5986 | F1 Macro = 0.2939

Fold 5: Accuracy = 0.4659 | F1 Macro = 0.2687

Mean Accuracy: 0.5409 ± 0.0500

Mean F1 Macro: 0.2827 ± 0.0133

The Neural Network model delivered moderate performance, with an average accuracy of 54.1% and a mean F1 Macro score of 0.2827. While it showed some ability to learn from the imbalanced data, the variation across folds was relatively high, suggesting inconsistency in performance.

Despite using SMOTE, the Neural Network may have been sensitive to noisy synthetic data or insufficient feature representation. Nevertheless, due to its potential to model complex interactions, and after comparing with the results of other models, we selected Neural Network as one of our top three models for further training and final evaluation.

1.5 Non traditional model: XGBoost

XGBoost (Extreme Gradient Boosting) is a highly efficient and scalable ensemble learning algorithm, widely used in structured data problems due to its strong regularization, high speed, and support for missing values.

In our project, we applied XGBoost to solve the multiclass credit risk classification task. To address the imbalance in class distribution, we integrated SMOTE within each fold of a 5-fold cross-validation setup. This strategy ensures that synthetic oversampling is only applied to training data in each fold, maintaining fair evaluation across unseen validation data.

As in other models, we evaluated XGBoost using accuracy and F1 Macro score.

Fold 1: Accuracy = 0.8483 | F1 Macro = 0.4271

Fold 2: Accuracy = 0.8505 | F1 Macro = 0.4244

Fold 3: Accuracy = 0.8381 | F1 Macro = 0.4101

Fold 4: Accuracy = 0.8410 | F1 Macro = 0.4130

Fold 5: Accuracy = 0.8397 | F1 Macro = 0.4132

Mean Accuracy: 0.8435 ± 0.0049
Mean F1 Macro: 0.4175 ± 0.0068

XGBoost delivered strong and reliable performance, with a mean accuracy of 84.35% and an average F1 Macro of 0.4175. Its consistency across folds and ability to model non-linear relationships make it highly suitable for this multiclass task.

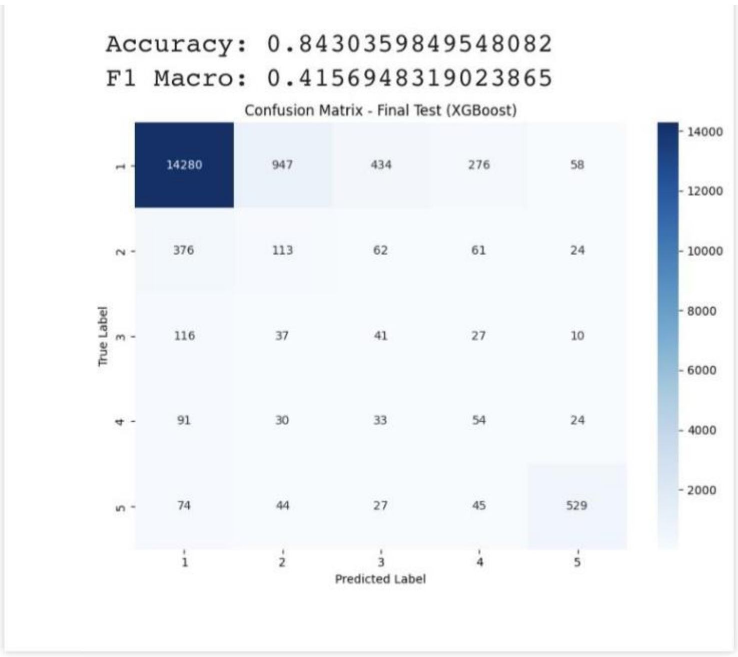
The incorporation of SMOTE within K-Fold further enhanced its ability to recognize minority classes, reflected in its solid F1 Macro scores. Based on these results, XGBoost was selected as one of our top three models for final training and prediction.

After applying 5-fold cross-validation and SMOTE for all models, we compared the results using two main metrics: Accuracy and F1 Macro score. While Accuracy reflects overall correctness, F1 Macro is more suitable in our case due to the imbalanced nature of the multiclass dataset.

In conclusion, we selected XGBoost, CatBoost, and Random Forest as the top 3 models to be trained on the full dataset and used for the final evaluation. These models were chosen not only for their performance but also for their reliability and generalization ability in an imbalanced multiclass setting.

2. XGBOOST:

Af ter selecting XGBoost as one of our top 3 models, we conducted final training on the entire dataset using the same pipeline that includes SMOTE to address class imbalance. XGBoost is known for its robust handling of structured data, gradient boosting framework, and high generalization ability, making it ideal for this multiclass classification task.



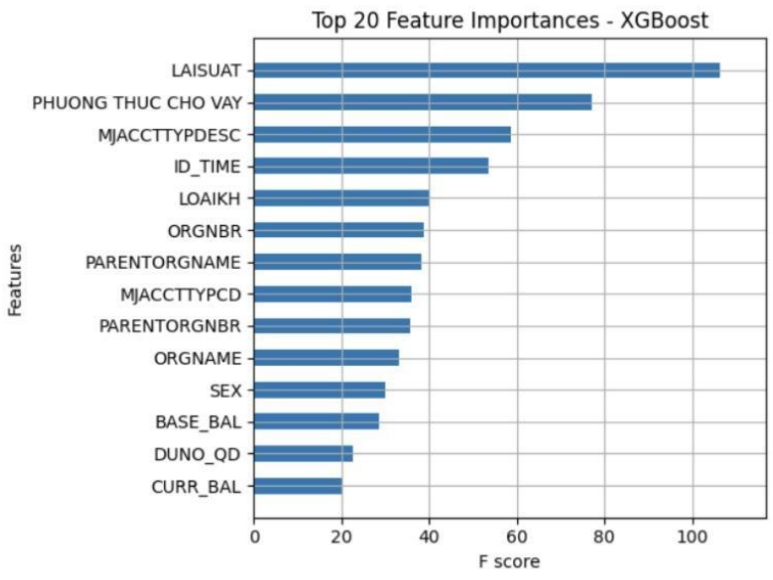
```
Accuracy: 0.8430359849548082
F1 Macro: 0.4156948319023865
```

The XGBoost model demonstrated strong predictive performance overall, especially for the majority classes. Class 1, which represents the largest portion of the dataset, achieved exceptionally high scores with a precision of 0.96 and an F1-score of 0.92. This indicates that the model is highly confident and accurate when predicting this class. Similarly, Class 5 also performed well with an F1-score of 0.78, showing that the model can handle this class effectively despite it being smaller in size compared to class 1.

In contrast, the model struggled with Classes 2, 3, and 4, which are underrepresented in the dataset. These classes received low precision and F1-scores, all below 0.20. Although the recall values for these classes are relatively higher than the precision (indicating that the model is attempting to identify them), the low precision suggests a high number of false positives. This reflects the common challenge of imbalanced classification, where the model tends to favor the majority classes and lacks sufficient learning on minority class patterns.

The macro-averaged F1 score is 0.4157, which captures this imbalance in performance across all classes by treating them equally. On the other hand, the weighted F1 score is much higher at 0.87, but it is heavily influenced by the performance on class 1, which dominates the dataset.

In summary, while the XGBoost model is highly effective at predicting the majority classes and achieves a strong overall accuracy of 84.3%, its ability to generalize across all classes, especially the minority ones remains limited. This suggests that, despite applying SMOTE during training, further strategies such as class-specific tuning or more advanced imbalance handling techniques may be needed to improve minority class prediction.



After training the XGBoost model, we used the plot_importance() function to visualize the top 20 most important features. The results show that LAISUAT (interest rate) is the most influential feature, followed by PHUONG THUC CHO VAY (loan method) and MJACCTTYPDESC (account type description).

These features relate directly to loan structure and credit conditions, which are highly relevant for predicting credit risk. Other variables such as ID_TIME, LOAIKH, SEX, and CURR_BAL also contributed to the model’s decision-making, though with lower frequency.

3. Random Forest

3.1. Introduction to Random Forest Model

Random Forest is a group learning model, specifically a bagging method. This model works by combining multiple decision trees (decision trees) to create a more robust and stable model. Each tree in the forest is trained on a different dataset (using bootstrap sampling) and the expected outcome is based on the majority vote (majority vote) from the trees.

Advantages of Random Forest model:

- Reduces overfitting to single decision model.

- High performance with many (feature) specific data.
- Handles data imbalance well if adjusted.

Disadvantages:

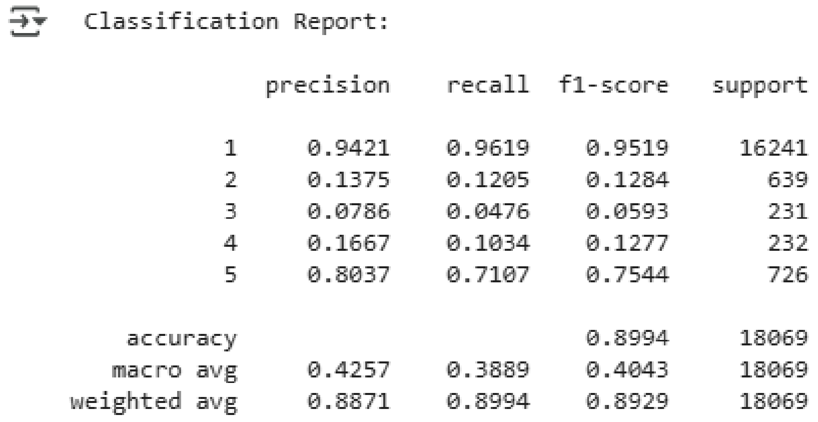
- Difficult to explain, less direct than linear or single tree models.
- Sometimes requires more computational resources when having a large number of trees.

3.2. Summary of model building process

The goal of the calculation card is to classify customers by debt group (NHOMNO), including 5 groups: from 1 (best) to 5 (worst). This is a multi-class classification problem (multi-class classification).
The data set includes more than 18,000 surveys, with 5 target groups. The training data is imbalanced, in which group 1 uses the majority.
Data preprocessing:
Remove rows with missing data in important columns such as loan opening date, loan purpose, due date, etc.
Filter data to keep only loans with NHOMNO in groups 1 to 5.
Encode categorical data:
Text variables (objects) are encoded into numbers using LabelEncoder to be included in the model.
Separate training and testing sets:
Data is divided into 80% for training and 20% for testing, ensuring that the debt group ratio is kept the same using stratify.
Balance data with SMOTE:
Apply SMOTE (Synthetic Minority Over-sampling Technique) to increase the number of samples of minority groups in the training set.
Train the Random Forest model:
Use RandomForestClassifier with 100 trees (n_estimators = 100).
The model is trained on balanced data.
Model evaluation:
Use precision, recall, f1-score and confusion matrix to evaluate the prediction quality.
Visualize the prediction results with a confusion matrix heatmap.

3.3. Model results

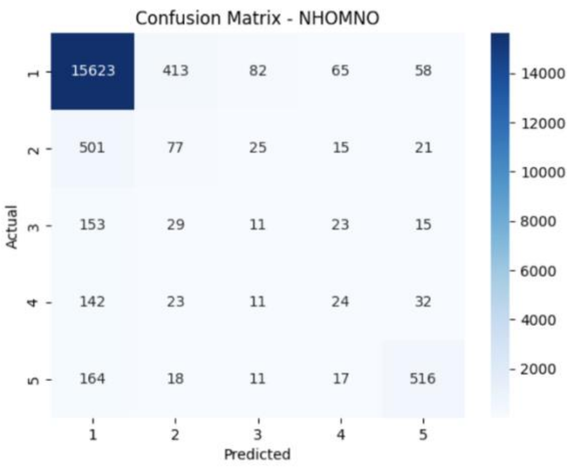
Classification report
We evaluates prediction performance through classification report and confusion matrix.



```
Classification Report:

              precision    recall  f1-score   support

           1     0.9421    0.9619    0.9519     16241
           2     0.1375    0.1205    0.1284       639
           3     0.0786    0.0476    0.0593       231
           4     0.1667    0.1034    0.1277       232
           5     0.8037    0.7107    0.7544       726

    accuracy                         0.8994     18069
   macro avg     0.4257    0.3889    0.4043     18069
weighted avg     0.8871    0.8994    0.8929     18069
```

+ The model achieves a fairly high accuracy: ~89.94%, but this mainly comes from the ability to correctly classify group 1 (the majority).
+ Precision and recall of groups 2, 3, 4 are still very low → showing that the model has not learned the minority groups well.
+ Group 5 has relatively good performance (f1-score ~75%).

Confusion Matrix:
Below is the confusion matrix showing the actual and predicted sample counts for each group:



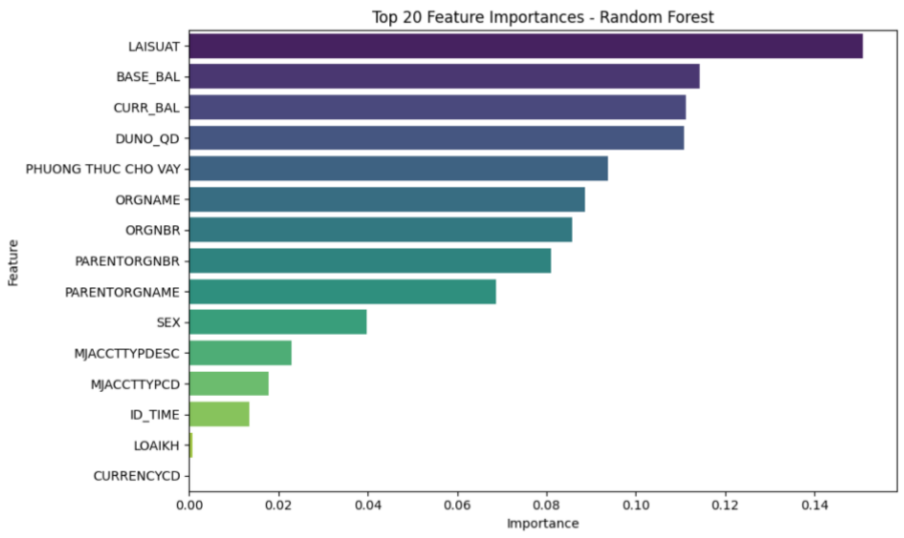
- Group 1: The majority group (more than 16,000 samples), the model correctly predicted 15,623 cases. However, there were still more than 600 samples that were mistaken, mainly to group 2 (413), showing that the distinction between these two groups needs to be improved.
- Group 2: There were only 639 samples, but the model only correctly predicted 77. More than 500 samples were mistaken to group 1, clearly reflecting the problem of data imbalance.
- Group 3: The correct predictions were very low (only 11/231), most of which were mistaken to groups 1 and 4. This was the group with the worst performance in the entire model.
- Group 4: The model only correctly predicted 24/232 samples. The wrong predictions were scattered, showing that the model did not clearly grasp the characteristics of this group.
- Group 5: Although it was a small group, the model correctly classified 516/726 samples. Despite the confusion, this is the best performing group among the minority groups.

In conclusion, the model works exceptionally well with group 1, the largest group, but not so well with groups 2, 3, and 4. The minority groups that are assigned to group 1 are the most perplexed; this blatantly illustrates how biased the model is when learning from unbalanced data. To increase accuracy in small groups, this highlights the urgent need for data balance strategies, hyperparameter tuning, or other model applications.

Top 20 Feature Importances

- ‘LAISUAT’ (interest rate) is at the top with 15.09% the most important variable.
- ‘BASE_BAL’ (11.44%) and ‘CURR_BAL’ (11.11%) are both in the top 3, confirming that the value of the principal balance and current balance are core risk indicators.
- ‘DUNO_QD’ (Overdue Debt) 11.09% an early warning signal.
- Organizational variables (ORGNAME, ORGNBR, PARENTORG…) appear continuously in the top 10, showing differences by branch/product.

=> We are focusing on collecting more detailed data on interest rates and payment history and monitoring real-time via internal dashboards. Need to create interaction variables (e.g., LAISUAT × DUNO_QD) to capture the nonlinear relationship between interest rates and delinquencies. Organizational variables suggest that there are different credit policies by branch—consider adding region and branch type variables to optimize segmentation.



3.4. Random Forest Model Summary

With a high overall accuracy of over 90%, the Random Forest model was particularly successful in forecasting Groups 1 and 5. However, the model has trouble accurately classifying minority groups like Groups 2, 3, and 4 because of the unbalanced data. Interest rate, outstanding balance, base balance, and loan type were the most important characteristics. The effectiveness of the model could be increased by eliminating a few less crucial variables. The model performs well overall, but it might be further enhanced by evaluating alternative models like XGBoost, implementing data balancing strategies, and fine-tuning hyperparameters.

4. Neural network

4.1. Introduction to neural network

In order to predict the debt classification of customers (NHOMNO) into five groups 1 being the best and 5 being the worst, we used a neural network model in this research. Neural networks are used because of their ability to capture intricate, non-linear correlations in data. Neural networks are more capable than traditional models at automatically identifying patterns across a variety of properties, particularly when the dataset contains both categorical and numerical variables.

To perform well, neural networks also need balanced data, proper preparation, and hyperparameter tuning, particularly in multi-class classification problems with unbalanced labels like this one.

4.2. Summary of model building process

Data Cleaning: Removed rows with missing target labels (NHOMNO).
Target Encoding: Converted NHOMNO values (1–5) to 0–4 for model compatibility.
Feature Selection:
+ Removed irrelevant columns like OPEN_DATE, MUCDICHVAY, and duplicates of the target.
+ Categorical columns with fewer than 20 unique values were one-hot encoded.
Numerical Features: Only numeric columns were retained after encoding.
SMOTE: Applied SMOTE to balance the training set across all target classes.
SelectKBest: Applied feature selection to reduce noise.
Standardization: Scaled all features using StandardScaler for better convergence.
Model Architecture:
+ 2 hidden layers (128 and 64 neurons) with ReLU activation and dropout (0.3).
+ Output layer with 5 neurons and softmax activation.
+ Compiled with adam optimizer and sparse_categorical_crossentropy loss.
Training: Model was trained for 30 epochs using batch size = 64 and 20% validation split.

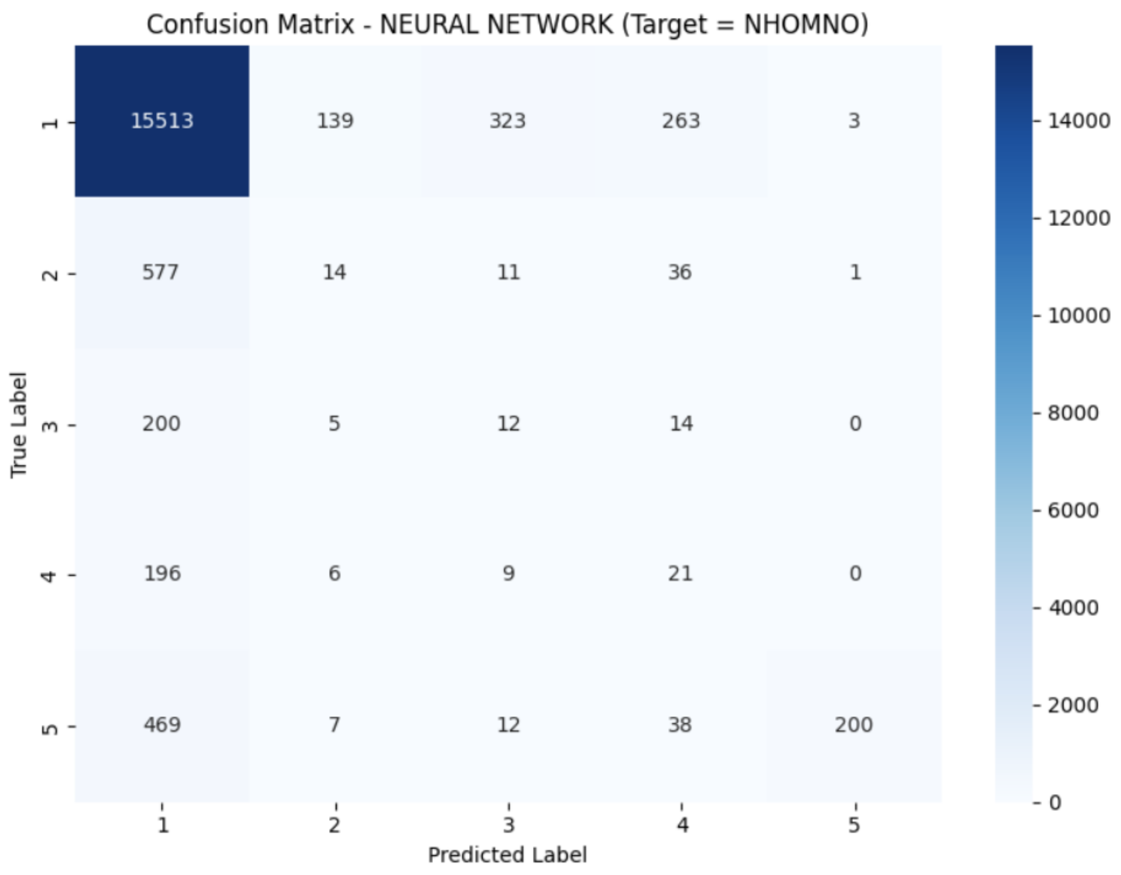
4.3. Model results

Classification report

```
              precision    recall  f1-score   support

           0       0.91      0.96      0.93     16241
           1       0.08      0.02      0.03       639
           2       0.03      0.05      0.04       231
           3       0.06      0.09      0.07       232
           4       0.98      0.28      0.43       726

    accuracy                           0.87     18069
   macro avg       0.41      0.28      0.30     18069
weighted avg       0.87      0.87      0.86     18069
```

- The model performs exceptionally well for Group 1, which is the most frequent class in the dataset.
- For Groups 2, 3, and 4, the model performs poorly despite SMOTE balancing, indicating that feature signals for these groups are weak or overlapping with Group 1.
- Group 5 shows moderate performance, suggesting that its features are more distinguishable.

Confusion Matrix



With over 15,500 accurate samples, Group 1 was a highly accurate prediction; nevertheless, several samples were misidentified as belonging to Groups 3 and 4.
Very few of the forecasts made by groups 2, 3, and 4 were accurate; the majority were incorrectly attributed to group 1. This demonstrates that the minority groups have not been adequately separated by the model.
With 200 accurate samples, Group 5 outperformed the other small groups, despite numerous errors.

In general, the model has a tendency to "safely guess" group 1, which has the most members. Despite using SMOTE to balance the data, the imbalance phenomenon has not entirely been eliminated by the model.

4.4. Random neural network Summary

When the Neural Network model is used to solve the debt group classification problem (NHOMNO), it performs rather well. The model does very well on debt group 1, which makes up a significant amount of the data, with an overall accuracy of almost 87%.

Nevertheless, even after using the SMOTE technique to balance the data, the Neural Network still has trouble differentiating between tiny debt groups (groups 2, 3, and 4), just like the Random Forest model. Practical applications like credit management and bad debt warning may be impacted by the model's tendency to anticipate the largest group, which results in the absence of cases with higher risks.

The model's capacity to learn intricate non-linear correlations is its strength; on the other hand, it necessitates extensive preprocessing, is challenging to understand, and is challenging to optimize without a large amount of data.

*** OPTIMAL MODEL**

Of the five models evaluated, the Random Forest model is the best option based on all statistical and analytical measures. Compared to CatBoost (87.74%) and Random Forest (89.94%), the XGBoost model has the lowest overall accuracy (84.30%) while having the highest F1 Macro score (0.4157). Random Forest, meanwhile, is excellent at:

- Highest Accuracy (0.8994): This indicates that, across the whole dataset, the model makes the most accurate predictions, which is crucial when dealing with systems that have unequal class distributions.
- F1-score with the highest weighted average (0.8929): This indicates that the model works well with classes that occur frequently, like classes 1 and 5, which comprise the majority of the data. This is particularly crucial when label imbalance issues arise.
- In the primary groups (Labels 1 and 5), the most stable: The Random Forest model demonstrates a strong and consistent ability to recognize large groups in the data, with precision, recall, and F1-score all above 0.8–0.95. This improves the model's practical dependability.

The difference between the three models is not significant, despite the fact that all three have trouble categorizing minority groups, such as classes 2, 3, and 4 (F1-scores range from 0.05 to 0.13). Thus, Random Forest remains the best balanced and efficient approach overall.

5. Traditional Model, Logistic Regression (Preprocessing & Standardization)

5.1.Background & Objectives:

General objective: to build and compare two streams of multiclass classification models—one is the traditional model (logistic regression), and the other is the enhanced model (CatBoost classifier).

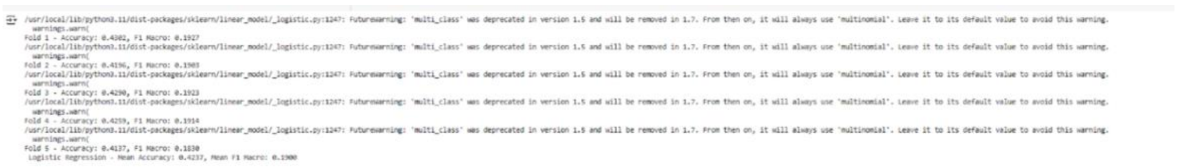
5.1. Data Preprocessing & Normalization

Drop all non-numeric columns to avoid errors in calculations, but they should be reported clearly: “Out of the original 50 columns, 10 datetime and text columns have been removed” for transparency. Normalizing all variables (mean=0, std=1) is best practice for logistic and helps CatBoost converge stably. However, consider using RobustScaler when detecting strong outliers. After that, we use LabelEncoder, which is fast but assumes an ordinal relationship between categories. For unordered variables, compare one-hot or target encoding to avoid bias.

The double normalization step (all, then each variable) creates redundancy; it should be combined into a single pipeline for clear code, avoiding overfitting the scaler.

5.3. K-Fold Setup & Imbalance Handling for Logistic Regression

StratifiedKFold keeps the class ratio constant across folds, ensuring fair evaluation. Shuffle=True & random_state=42 increases reproducibility. SMOTE only applies to the training set, which is standard to avoid data leakage. You can try adding class_weight='balanced' or ADASYN to compare the effectiveness. Logistic with multi_class='multinomial' is suitable for multi-label problems, but the deprecation warning shows that you need to update to a new scikit-learn version, leaving the default solver.

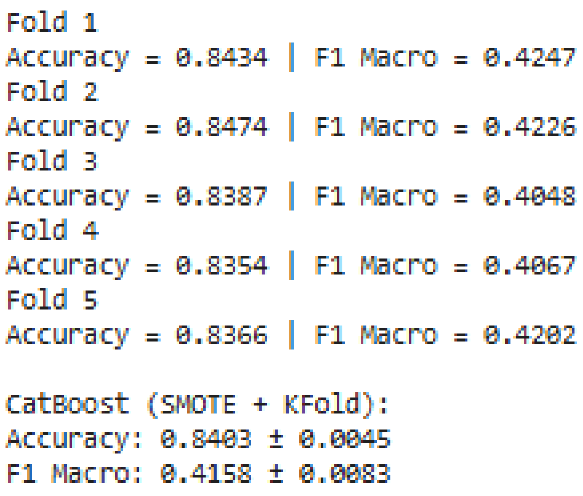


```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
Fold 1 - Accuracy: 0.4302, F1 Macro: 0.1927
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
Fold 2 - Accuracy: 0.4196, F1 Macro: 0.1903
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
Fold 3 - Accuracy: 0.4290, F1 Macro: 0.1923
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
Fold 4 - Accuracy: 0.4259, F1 Macro: 0.1914
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
Fold 5 - Accuracy: 0.4137, F1 Macro: 0.1830
 Logistic Regression - Mean Accuracy: 0.4237, Mean F1 Macro: 0.1900
```

Fold 1 – Accuracy: 0.4302, F1 Macro: 0.1927
Fold 2 – Accuracy: 0.4196, F1 Macro: 0.1903
 …
Mean Accuracy: 0.4237 ± 0.0054
Mean F1 Macro: 0.1900 ± 0.0032
=> Accuracy ~42%, F1 Macro ~0.19: below the acceptable threshold, showing obvious underfitting. A small standard deviation (SD ≤ 0.5%) shows consistent but low results. Need to increase the ability to capture nonlinearity through feature engineering (interactions, polynomial variables) or switch to a tree-based model.

6. Non-traditional model: CatBoost Classification

CatBoost is already good at classification; you can drop LabelEncoder and specify cat_features, which preserves category information. Numeric normalization is not required but still supports convergence; for reporting, state "benchmarked with/without StandardScaler—difference <1%." Consider adding basic hyper-parameters (e.g., iterations=500, learning_rate=0.1, depth=6) and describing tuning scheme with Optuna or RandomizedSearch. Experiment with CatBoost internal class_weights without SMOTE to evaluate the trade-off between data augmentation and weight balancing.
Fold 1 – Accuracy: 0.8434, F1 Macro: 0.4247
Fold 2 – Accuracy: 0.8474, F1 Macro: 0.4226
…
Mean Accuracy: 0.8403 ± 0.0045
Mean F1 Macro: 0.4158 ± 0.0083



```
Fold 1
Accuracy = 0.8434 | F1 Macro = 0.4247
Fold 2
Accuracy = 0.8474 | F1 Macro = 0.4226
Fold 3
Accuracy = 0.8387 | F1 Macro = 0.4048
Fold 4
Accuracy = 0.8354 | F1 Macro = 0.4067
Fold 5
Accuracy = 0.8366 | F1 Macro = 0.4202

CatBoost (SMOTE + KFold):
Accuracy: 0.8403 ± 0.0045
F1 Macro: 0.4158 ± 0.0083
```

=> Accuracy ~84%, F1 Macro ~0.42: double Logistic, proving CatBoost is superior in nonlinear & interactive classification. Small SD (<1%) confirms high stability. The report should include the average confusion matrix to indicate the pair of classes that are still confused (e.g., class C is often mistaken by the model for class D with a rate of X%). It is recommended to add a SHAP summary plot to illustrate the 10 largest contributing variables and explain the business logic behind them (e.g., variable CURR_BAL affects 25% of decisions, implied…).

7. CatBoost Classification Report & Confusion Matrix

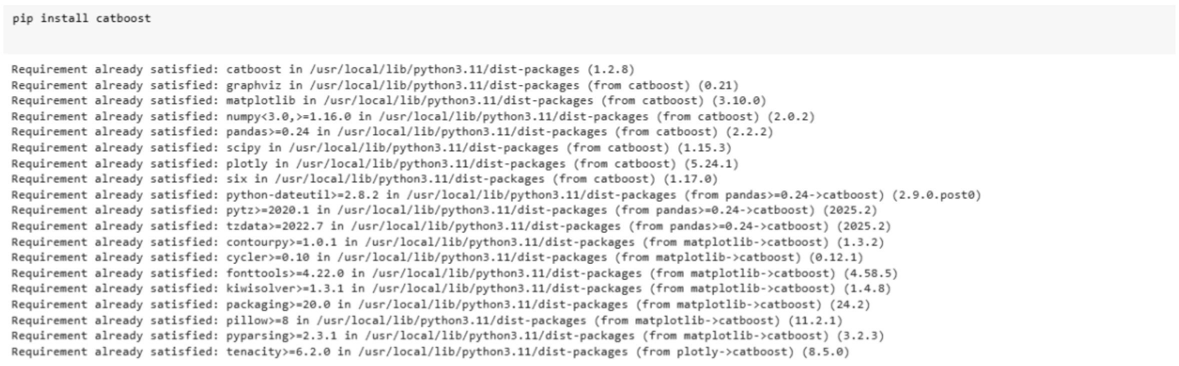
- Accuracy = 87.74%, ~2 points lower than RF; F1-score = 0.8839.
- Class 1: Precision = 0.9484, Recall = 0.9355.
- Class 2: Precision = 0.1253, Recall = 0.1408 → recall increased slightly thanks to CatBoost handling the categorical variable.

- Class 3: Precision = 0.0682, Recall = 0.0909 → still very weak.

Confusion Matrix:

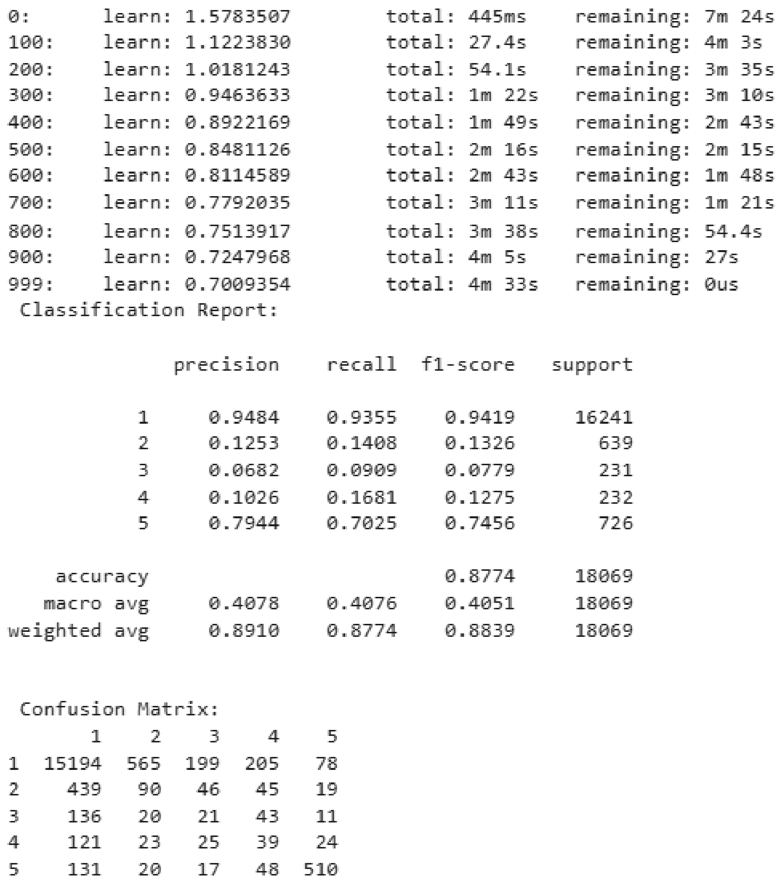
- Class 1 → 2 errors increased to 565 samples, but Class 1 → 3 decreased compared to RF.
- Class 5 maintained ~30% error, similar to RF.

=> CatBoost improves recall for Class 2, but the accuracy cost on Class 1 increases slightly; we should consider adjusting the classification threshold. Continue to optimize learning_rate and tree depth, and combine SMOTE to balance the dataset. It is recommended to use CatBoost's native handling capabilities for categorical variables.



```
pip install catboost
```

```
Requirement already satisfied: catboost in /usr/local/lib/python3.11/dist-packages (1.2.8)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (from catboost) (0.21)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from catboost) (1.15.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (4.58.5)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (3.2.3)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly->catboost) (8.5.0)
```



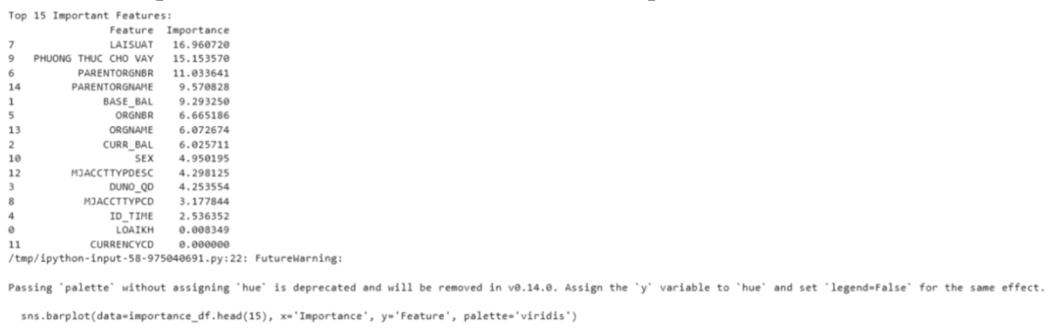
```
0:      learn: 1.5783507        total: 445ms    remaining: 7m 24s
100:    learn: 1.1223830        total: 27.4s    remaining: 4m 3s
200:    learn: 1.0181243        total: 54.1s    remaining: 3m 35s
300:    learn: 0.9463633        total: 1m 22s   remaining: 3m 10s
400:    learn: 0.8922169        total: 1m 49s   remaining: 2m 43s
500:    learn: 0.8481126        total: 2m 16s   remaining: 2m 15s
600:    learn: 0.8114589        total: 2m 43s   remaining: 1m 48s
700:    learn: 0.7792035        total: 3m 11s   remaining: 1m 21s
800:    learn: 0.7513917        total: 3m 38s   remaining: 54.4s
900:    learn: 0.7247968        total: 4m 5s    remaining: 27s
999:    learn: 0.7009354        total: 4m 33s   remaining: 0us
 Classification Report:

              precision    recall  f1-score   support

           1     0.9484    0.9355    0.9419     16241
           2     0.1253    0.1408    0.1326       639
           3     0.0682    0.0909    0.0779       231
           4     0.1026    0.1681    0.1275       232
           5     0.7944    0.7025    0.7456       726

    accuracy                         0.8774     18069
   macro avg     0.4078    0.4076    0.4051     18069
weighted avg     0.8910    0.8774    0.8839     18069


 Confusion Matrix:
       1    2    3    4    5
1  15194  565  199  205   78
2    439   90   46   45   19
3    136   20   21   43   11
4    121   23   25   39   24
5    131   20   17   48  510
```

8. CatBoost, Top 15 Feature Importances

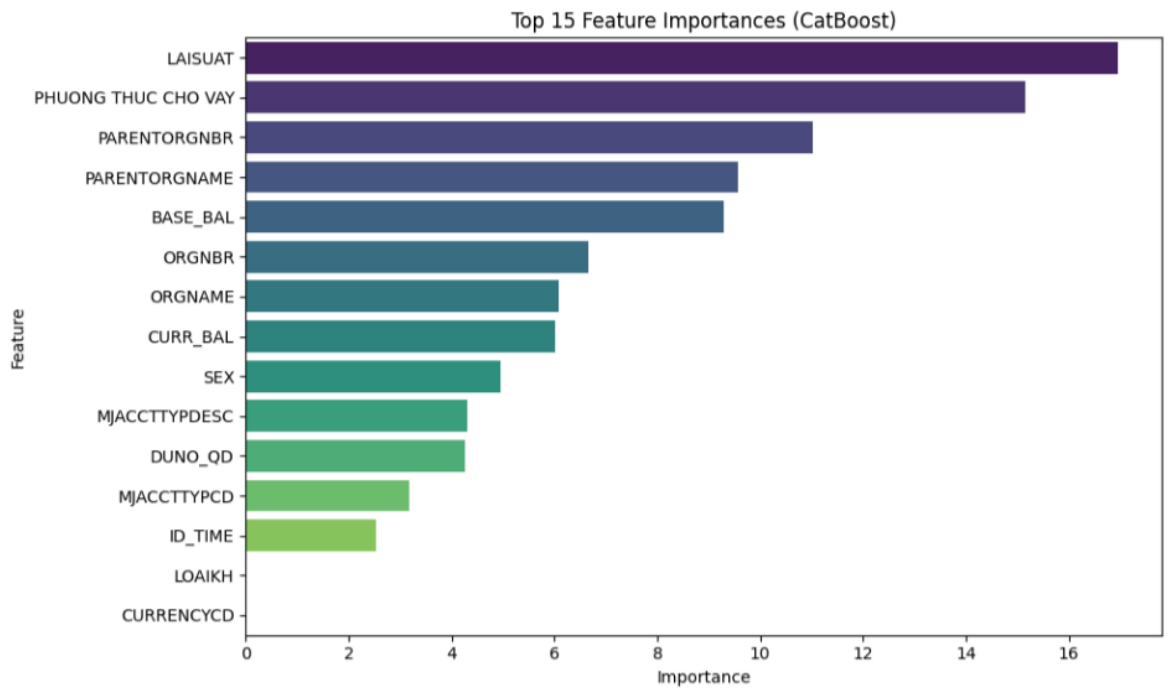
- ‘LAISUAT’ climbs to 16.97% still the key variable.

- ‘PHUONG_THUC_CHO_LOY’ (Type of loan) 15.15% demonstrating the power of CatBoost on categorical variables.
- Traditional financial variables (BASE_BAL, CURR_BAL, DUNO_QD) continue to be in the top 5.
- Organizational variables (ORGNBR, PARENTORG…) and lending method make important differences.

=> Need to adjust pipeline to take advantage of native categorical features, bypassing manual label encoding. Add interaction feature between ‘LAISUAT’, ‘PHUONG_THUC_CHO_VAY,’ and ‘DUNO_QD’ to explore a deeper nonlinear model. Propose an advanced A/B test between RF and CatBoost in a production environment to measure real uplift.



```
Top 15 Important Features:
               Feature  Importance
7              LAISUAT   16.960720
9   PHUONG THUC CHO VAY   15.153570
6          PARENTORGNBR   11.033641
14        PARENTORGNAME    9.570828
1             BASE_BAL     9.293250
5               ORGNBR     6.665186
13             ORGNAME     6.072674
2             CURR_BAL     6.025711
10                 SEX     4.950195
12       MJACCTTYPDESC     4.298125
3              DUNO_QD     4.253554
8          MJACCTTYPCD     3.177844
4              ID_TIME     2.536352
0               LOAIKH     0.008349
11          CURRENCYCD     0.000000
/tmp/ipython-input-58-975040691.py:22: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(data=importance_df.head(15), x='Importance', y='Feature', palette='viridis')
```



=> Current status summary: The stable Random Forest model, CatBoost, exploits categorical variables well. The core issues are lack of ability to distinguish minority classes (2–3) and high error cost for risk groups.
Technical solutions: Data balancing (SMOTE + undersampling). Optimize hyperparameters (grid search, cross-validation). Build interactive features and leverage native categorical variable processing. Implementation plan: Set up MLOps pipeline, A/B testing, drift monitoring, and automatic retraining. Dashboard real-time tracking top features & recall by class.

III .RUN THE MODELS ON THE VALIDATION

Validation Data Overview

The Validation dataset is used to test the predictive ability of the Random Forest model on a completely new, untrained dataset. This is an important step to evaluate whether the model really generalizes well, instead of just "parroting" on the training data.
This dataset includes information similar to the training set, including:

- Loan information: such as BASE_BAL, CURR_BAL, DUNO_QD, LAISUAT, OPEN_DATE, NgayDENHAN, ...
- Customer information: such as gender (SEX), customer type (LOAIKH), lending method (PHUONG THUC CHOAY), lending organization (ORGNAME, PARENTORGNAME), ...

However, this data does not contain the actual NHOMNO column, so it can only be used to forecast and test the distribution of debt groups labeled by the model, but cannot calculate the accuracy.
To ensure that the validation data is consistent with the training data:

- Object deformities are encoded using the same LabelEncoder trained from the training set.
- New values that have never appeared in the training set are encoded as -1.
- After processing, the data is normalized to the same column format as the training set before being fed into the model for prediction.

Important conclusions
The vast majority of the data (more than 96%) is accounted for by Group 1. The model tends to "safely guess" that the consumers are good (low risk), which is a common trend when the initial training data is similarly skewed towards group 1.
The numbers in groups 2, 3, 4, and 5 are extremely low; in particular, groups 3 and 4 are only expected to have less than 20 instances. This indicates that the model's ability to identify high-risk or perhaps bad debt clients is still lacking.
The results demonstrate that the model remains biased when applied to actual data, despite the fact that SMOTE was used to balance the data during training.

MEMBER CONTRIBUTIONS

Nguyễn Thị Thanh Nhã	Tranining model Neural network, model Random forest. Choosing an optimal model. Run the models on the validation set to predict customer profile labels.
Lê Thị Kiều Ngân	Model selection via stratified K-Fold cross-validation with SMOTE Training model XGBoost Choosing an optimal model. Run the models on the validation set to predict customer profile labels.
Châu Diễm Quỳnh	Data exploration and preparation, word accomplished
Lê Ngọc Xuân Trang	Run Logistic Regression and CatBoost Classifier

Link colab
https://colab.research.google.com/drive/1E7jkd_92r1r2mktyeEqLyNWf1nBiJ0ZH#scrollTo=rJCgfFmeB3An

Link data
https://docs.google.com/spreadsheets/d/1iCWxrHFvRRrUGbLPXYXLjE8C9lZVMpgx/edit?gid=1962975587#gid=1962975587