ABSTRACT

Part 1

Amortization models are essential tools for understanding and managing loan repayments, helping both borrowers and lending institutions to design effective repayment plans. The first part explores and implements five common amortization models—Italian, French, American, Geometric, and Lump-Sum. We introduce the basic principles of each model and implements them through programming to simulate the actual loan repayment processes. We also allow users to select different amortization models and visually compare their performance based on key parameters such as balance, interest, and principal. In conclusion, this part highlights the advantages and disadvantages of each amortization method, suggesting that the choice of method depends on the borrower's financial situation and needs.

Part 2

When determining loan approvals, banks typically evaluate an applicant's profile based on specific criteria. Traditional manual approval processes are often inefficient, time-consuming, and prone to human errors. To address this, we aim to develop a predictive model to enhance decision-making accuracy. Using a loan dataset from Kaggle, we identified that XGBoost, LightGBM, and CatBoost consistently outperform other commonly used models. Consequently, we combined these models into a stacked meta-model to further improve prediction accuracy. Our analysis of feature contributions revealed that banks primarily assess applicants based on their financial status, credit rating, and repayment stability when deciding loan approvals.

TABLE OF CONTENT

PART I - AMORTIZATION METHODS

INTRODUCTION

In the financial sector, loan repayment is a complex process involving the time value of money and the calculation of instalment payments. Amortization models are key tools for understanding and managing loan repayments; they help borrowers and lending institutions forecast future cash flows and devise reasonable repayment plans. Different countries and regions may employ different amortization methods, which vary in calculating periodic payments and remaining principal.

This report will first introduce five common amortization methods: Italian, French, American, Geometric and Lump-Sum. Each method has its unique calculation and applicable scenarios. Subsequently, we implement these models in the Python programming language, enabling users to select different amortization methods as needed and to visually compare the repayment situations among different methods.

In the final section, by comparing the advantages and disadvantages of each amortization method, we conclude that: the choice of amortization method depends on the borrower’ s financial situation and needs. For those with cash flow constraints, the Chinese and American methods (interest-first) provide flexibility by reducing early financial pressure. Borrowers with stable income and long-term repayment goals benefit from the Italian method (equal principal and interest) for its stability and ease of planning. For short-term, small loans, the Chinese and American methods are ideal, offering immediate relief and flexibility for urgent financial demands.

MODELS

1. Italian Amortisation Method

1.1 Overview

In the Italian amortisation (equal principal) method of loan repayment, the borrower is required to repay a fixed amount of principal in each instalment throughout the loan term.

1.2 Key parameters

P: The principal amount of the loan.	r: The interest rate per period.
n: The total number of periods.	A_k: The repayment amount for the k period.
P_k: The principal remaining at the beginning of the k period.	

1.3 Annuity calculation formula

$$A_k = \left(\frac{P}{n}\right) + \left(P - \sum_{i=1}^{k-1} \frac{P}{n}\right) \times r$$

This formula indicates that the amount to be repaid in each instalment consists of two parts: the fixed principal portion $\frac{P}{n}$ and the interest portion of the remaining principal.

1.4 Details of each installment

- Interest I_k for the k period:

$$I_k = \left(P - \sum_{i=1}^{k-1} \frac{P}{n}\right) \times r$$

- Principal repayment for the k period:

$$\frac{P}{n}$$

- The remaining principal P_{k+1} at the end of the k period:

$$P_{k+1} = P_k - \frac{P}{n}$$

2. French amortisation method

2.1 Overview

The French amortization method is a loan repayment approach that features equal principal and interest payments. In this method, each installment consists of both principal and interest components, with the interest portion gradually decreasing and the principal portion increasing over time.

2.2 Key parameters

P: The principal amount of the loan	r: The interest rate per period
n: The total number of periods	A: The annuity payment per period

P_k: The principal remaining at the beginning of the k period

2.3 Annuity calculation formula

$$A = P \times \frac{r(1+r)^n}{(1+r)^n - 1}$$

The repayment amount per instalment A consists of both principal and interest components, with the interest component gradually decreasing and the principal component gradually increasing over time.

2.4 Details of each installment

- For each instalment, we have Interest I_k for the k period.

$$I_k = P_k \times r$$

- The principal to be repaid in the k period.

$$A - I_k$$

- Remaining principal P_{k+1} at the end of period k.

$$P_{k+1} = P_k - (A - I_k)$$

3. United States Amortisation Method

3.1 Overview

The amortization by the American method is characterized by the payment of only interests during the loan period, the whole capital is paid at the end of the loan.

3.2 Key parameters

P: The principal amount of the loan.	r: The interest rate per period.
n: The total number of periods.	I_k: The interest for the k period.

A: The final instalment amount, including all remaining principal and the interest for the final instalment.

3.3 Payment schedule

- Interest payment per instalment:

$$I_k = P \times r$$

- Final instalment:

$$A = P + I_k$$

4. Geometric Amortisation Method

4.1 Overview

The Geometric Amortization Method is a loan repayment approach where the payment amount increases or decreases at a steady rate (geometrically) over the life of the loan. Unlike traditional methods where payments stay the same or follow a linear pattern, the geometric method introduces a factor that determines how the payment amount changes over time. This allows for flexibility in situations where the borrower's income or the financial situation is expected to change at a consistent rate.

4.2 Key parameters

P: The principal amount of the loan. n: The total number of periods.

r: The interst rate per period. I_k: The interest for the k period.

P_k: The principal remaining at the beginning of the k period.

A_k: The repayment amount for the k period.

factor: Value of the progression (for a progression of t% it would be expressed as 1.0t)

4.3 Payment schedule

$$A_1 = \left(\frac{\mathrm{P} \times (\mathrm{r} - \mathrm{factor})}{1 - \left(\frac{1+factor}{1+\mathrm{r}}\right)^{\mathrm{n}}} \right)$$

- Interest for each period:

$$I_k = P_k \times r$$

- Repayment for each period:

$$A_k = A_{\text{k-1}} \times q$$

- Principal for each period:

$$P_k = A_k - I_k$$

- Last year of repayment:

For the last year of repayment, the principal should be equal to the balance of the previous period so that the final balance can be set to zero to fully pay off the loan:

$$P_n = \mathrm{Balance}_{n-1} = P - \sum_{k=1}^{n-1} P_k$$

5. Chinese Amortization

5.1 Overview

The Lump-Sum method is a straightforward and commonly used loan repayment approach in which the borrower repays the loan principal and accumulated interest at the end of the loan term in a single payment. According to Article 6, Loan Repayment, of the Personal Loan Contract (2024 Edition) issued by the Bank of China.

5.2 Key parameters

P: The principal amount of the loan. n: The total number of periods.

r: The interst rate per period

5.3 Payment schedule

- Total repayments：

$$Total\ Repayment = P + Interest$$

This formula reflects the simplicity of the method, as no intermediate payments are made during the loan term.

- Calculation of interest:

$$\text{Interest} = P \times n \times r$$

TEST CASE

The input values provided are for calculating the amortization schedule using four different methods: Italian, French, American, and Lump-Sum. The loan amount is $1,000,000, with an annual interest rate of 5% and a repayment frequency of once per year. The amortization term is set to 10 years. Additionally, for the geometric method, a factor of 0.98 is provided to adjust the payment schedule.Please make sure that the factor cannot be greater than one plus the interest rate when you set it by yourself . These parameters will be all used to compute the loan repayment amounts and timelines based on the selected method.

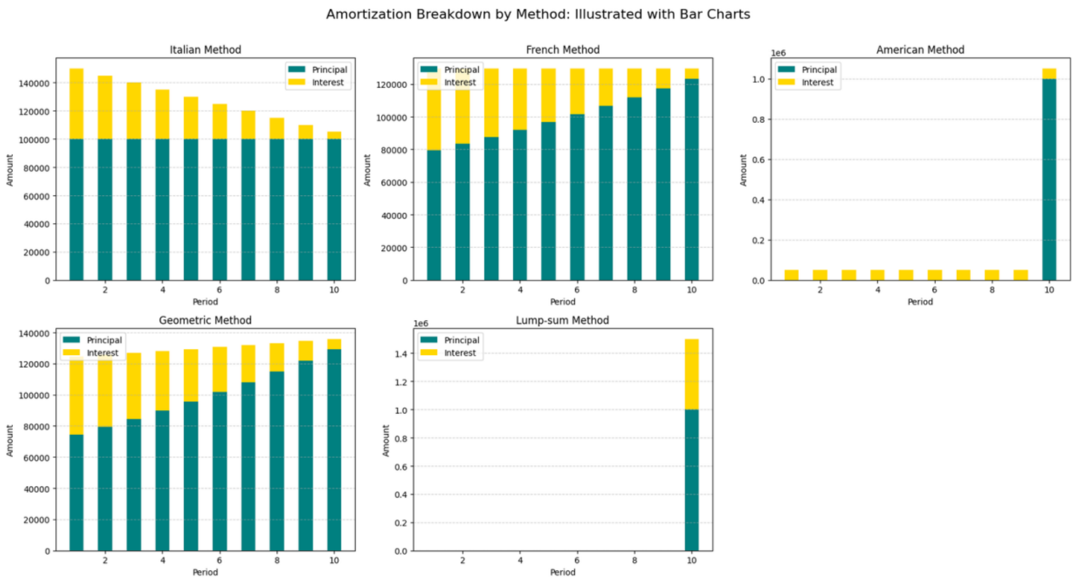


Figure 1: Amortization breakdown by methods

These five images illustrate the breakdown of five different amortization methods using stacked bar charts. Each chart shows the distribution of Principal and Interest payments across multiple periods.

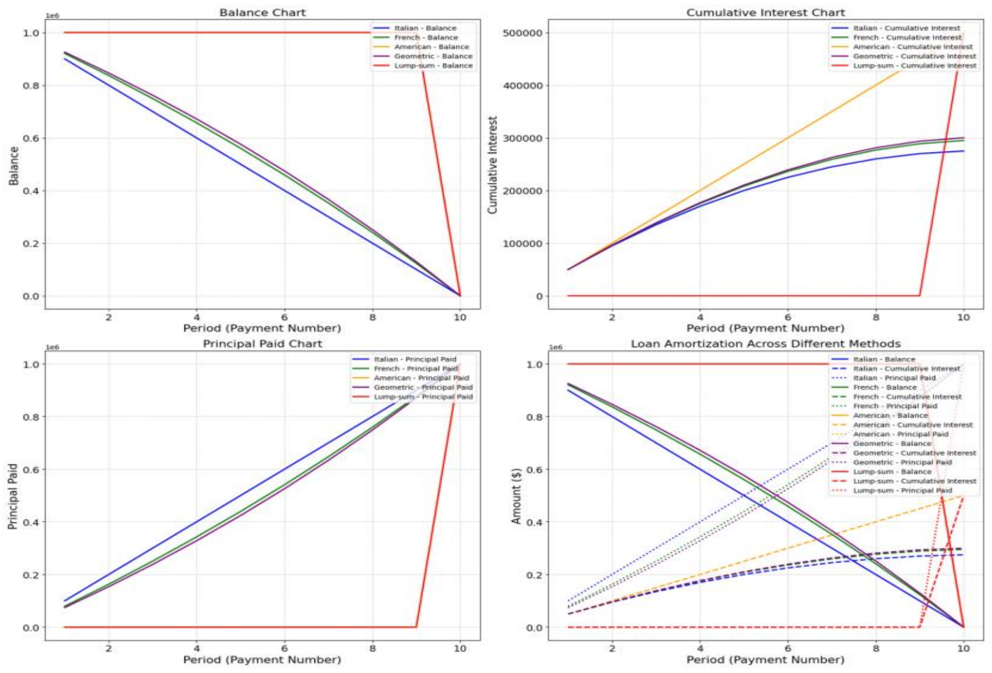


Figure 2: Comparation across methods

This image compares loan amortization methods using four charts. The Balance Chart shows that the Italian, French, and Geometric methods steadily reduce the loan balance, while the American and Lump-sum methods have same curves and maintain the balance until the final period. The Cumulative Interest Chart reveals that the Lump-sum and American methods accumulate the most interest due to delayed principal payments, while the other methods have a slower accumulation. The Principal Paid Chart indicates steady repayment for Italian, French, and Geometric methods, while the American and Lump-sum methods which have same curves trend repay all principal at the end. Lastly, the Loan Amortization Chart provides an overview, showing smoother repayment for Italian and French methods, a declining trend for Geometric, and sharp end-period changes for American and Lump-sum methods.

CONCLUSION

Different amortization methods are suitable for different individuals and situations. Below, we analyze these methods from various perspectives to determine which is most suitable under specific circumstances.

1. For borrowers facing cash flow constraints, the Chinese amortization method and the American amortization method (interest-first) are ideal choices. These methods significantly reduce financial pressure in the early stages of the loan, as borrowers only need to pay interest without repaying the principal during the loan term. This not only alleviates cash flow challenges but also provides greater financial flexibility for daily operations and other investments, maximizing the efficiency of fund utilization.

2. For borrowers with stable income, no cash flow constraints, and a preference for consistent repayments, the Italian amortization method (equal principal and interest) is ideal. It ensures fixed monthly payments, simplifying budgeting and long-term financial planning while protecting against interest rate risks in fixed-rate loans, offering stability and certainty.
3. For short-term, small-scale loans, the Chinese amortization method or the American amortization method (interest-first) is often recommended. These methods reduce financial pressure by requiring only interest payments during the loan term, leaving the principal repayment until the end. This minimizes immediate financial burden and allows borrowers greater flexibility in using funds, which is particularly valuable for addressing urgent or short-term needs.

PART 2 – LOAN APPROVAL QUESTION

INTRODUCTION

This study aims to address two key questions: (1) Which machine learning models can improve classification performance in predicting loan approvals? (2) What factors significantly influence a bank's decision to approve or reject a loan application?

The experiment begins by training baseline models to observe their individual performance and the contribution of features to these models. The classification effectiveness of the models will be evaluated using the Receiver Operating Characteristic (ROC) curve to ensure robust and interpretable results. Additionally, feature importance analysis will be conducted in feature selection and to identify the key factors affecting loan approval decisions. Based on the analysis, LightGBM, XGBoost, and CatBoost, known for their strong performance in structured data tasks, are selected for stacking. A total of 26 features that significantly contribute to these three models are retained as the meta-model's features.

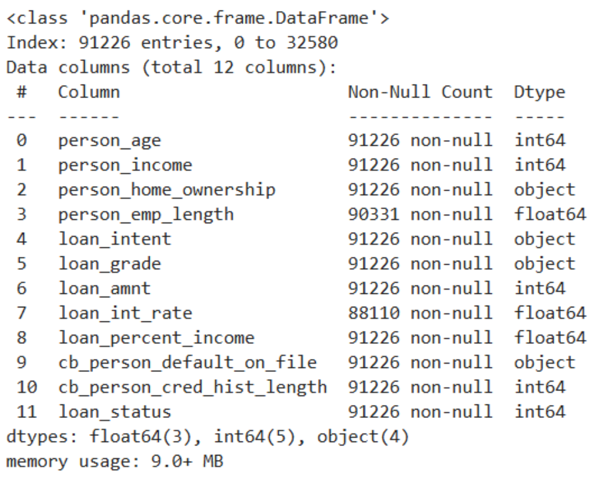
RAW DATA

In this experiment, the following three datasets will be used: train.csv; test.csv; credit_risk_dataset.csv. Among them, credit_risk_dataset.csv is the original dataset, and the train.csv and test.csv are trained by the deep learning model on the basis of the original dataset by contest organizer, and the data features are the same as the original version. In order to make full use of the data and improve the prediction accuracy, we combined credit_risk_dataset.csv with the train.csv. The following is the process of processing the merged dataset.

PROCESS

1. Data exploration

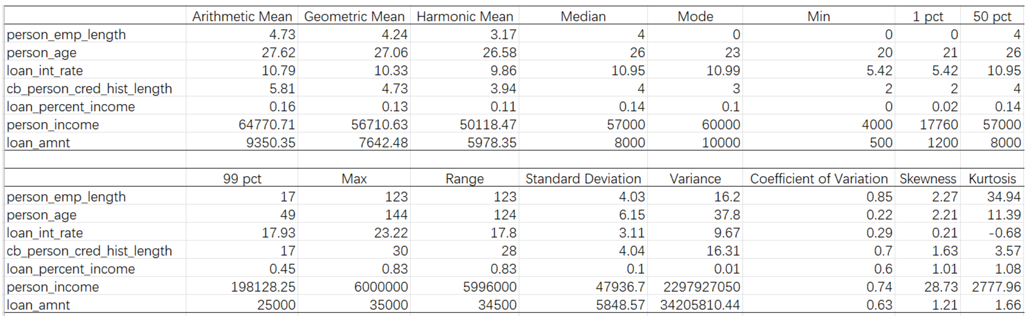
First, get a preliminary understanding of the data and use info() to view the relevant information. Here you can see the dataset size, a total of 91226 pieces of data, 12 features; In terms of missing values, we can see that "person_emp_length" and "loan_int_rate" have missing values, because the number of non-NULL is less than the total; In terms of data types, the dataset has not only numeric types, but also object types.



```
<class 'pandas.core.frame.DataFrame'>
Index: 91226 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   person_age                  91226 non-null  int64  
 1   person_income               91226 non-null  int64  
 2   person_home_ownership       91226 non-null  object 
 3   person_emp_length           90331 non-null  float64
 4   loan_intent                 91226 non-null  object 
 5   loan_grade                  91226 non-null  object 
 6   loan_amnt                   91226 non-null  int64  
 7   loan_int_rate               88110 non-null  float64
 8   loan_percent_income         91226 non-null  float64
 9   cb_person_default_on_file   91226 non-null  object 
 10  cb_person_cred_hist_length  91226 non-null  int64  
 11  loan_status                 91226 non-null  int64  
dtypes: float64(3), int64(5), object(4)
memory usage: 9.0+ MB
```

Figure 3: df dataset

Here we used the indicators taught in the class to observe the data distribution. As shown in the picture, the majority of the features exhibit a right-skewed distribution, with 'person_income' showing an extremely high skewness and kurtosis, indicating a significant presence of outliers with very high income values. The 'person_emp_length' and 'cb_person_cred_hist_length' also display considerable variability , suggesting diverse work and credit histories among individuals. In contrast, 'loan_int_rate' and 'loan_percent_income' demonstrate more moderate variability and near-normal distributions, implying relative stability in loan interest rates and the proportion of income allocated to loan repayments. The average 'person_age' is relatively young at 27.62 years, with a narrow spread, while 'loan_amnt' has a substantial range but a moderate average, reflecting a wide disparity in loan amounts.



	Arithmetic Mean	Geometric Mean	Harmonic Mean	Median	Mode	Min	1 pct	50 pct
person_emp_length	4.73	4.24	3.17	4	0	0	0	4
person_age	27.62	27.06	26.58	26	23	20	21	26
loan_int_rate	10.79	10.33	9.86	10.95	10.99	5.42	5.42	10.95
cb_person_cred_hist_length	5.81	4.73	3.94	4	3	2	2	4
loan_percent_income	0.16	0.13	0.11	0.14	0.1	0	0.02	0.14
person_income	64770.71	56710.63	50118.47	57000	60000	4000	17760	57000
loan_amnt	9350.35	7642.48	5978.35	8000	10000	500	1200	8000

	99 pct	Max	Range	Standard Deviation	Variance	Coefficient of Variation	Skewness	Kurtosis
person_emp_length	17	123	123	4.03	16.2	0.85	2.27	34.94
person_age	49	144	124	6.15	37.8	0.22	2.21	11.39
loan_int_rate	17.93	23.22	17.8	3.11	9.67	0.29	0.21	-0.68
cb_person_cred_hist_length	17	30	28	4.04	16.31	0.7	1.63	3.57
loan_percent_income	0.45	0.83	0.83	0.1	0.01	0.6	1.01	1.08
person_income	198128.25	6000000	5996000	47936.7	2297927050	0.74	28.73	2777.96
loan_amnt	25000	35000	34500	5848.57	34205810.44	0.63	1.21	1.66

Figure 4: Non-object values distribution

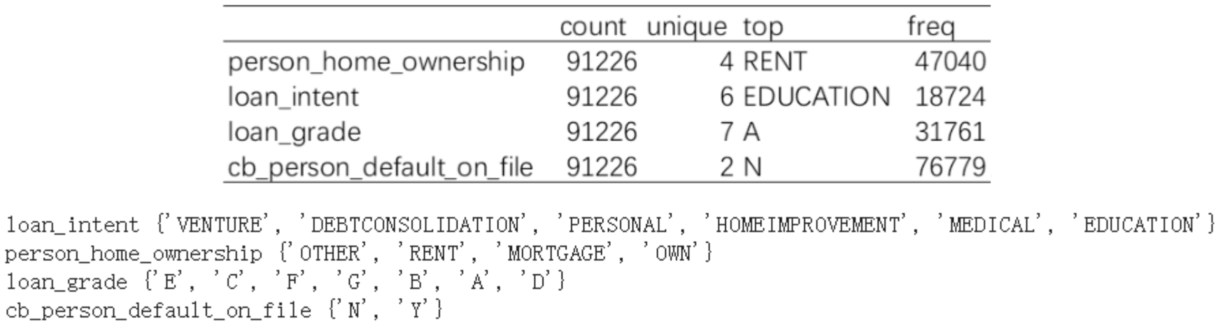
As the information shown here：

Home Ownership (person_home_ownership): The majority of individuals in the dataset are renters, with 47040 out of 91226 records falling under the 'RENT' category.

Loan Intent (loan_intent): The most common reason for seeking a loan is 'EDUCATION', accounting for 18724 records, indicating a significant number of applicants are looking to finance their education.

Loan Grade (loan_grade): The most frequently occurring loan grade is 'A', with 31761 records, suggesting that a substantial portion of applicants have a high credit rating.

Default Status (cb_person_default_on_file): A significant majority, 76779 out of 91226, have no record of default, which is a positive indicator of creditworthiness.



	count	unique	top	freq
person_home_ownership	91226	4	RENT	47040
loan_intent	91226	6	EDUCATION	18724
loan_grade	91226	7	A	31761
cb_person_default_on_file	91226	2	N	76779

```
loan_intent {'VENTURE', 'DEBTCONSOLIDATION', 'PERSONAL', 'HOMEIMPROVEMENT', 'MEDICAL', 'EDUCATION'}
person_home_ownership {'OTHER', 'RENT', 'MORTGAGE', 'OWN'}
loan_grade {'E', 'C', 'F', 'G', 'B', 'A', 'D'}
cb_person_default_on_file {'N', 'Y'}
```

Figure 5: Object value distribution

In order to directly observe the relationship between the target value and the eigenvalue, we first used the kernel density estimation graph to explore the distribution of each non-object feature with the target value. The image reveals that there are significant distribution differences in ‘person_income’, ‘loan_amnt’, ‘loan_int_rate’, and ‘loan_percent_income’ between the ‘Approve’ and ‘Non-Approve’ groups. In some value ranges, these variables show a clear correlation with the target variable. For example, the higher the loan interest rate, the greater the probability of not being approved; However, the distribution of person_age, person_emp_length, cb_person_cred_hist_length, and these characteristics were not significantly different, and it was difficult to identify the correlation with the target value.

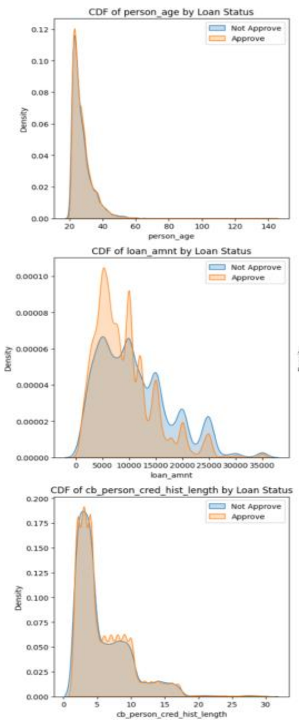
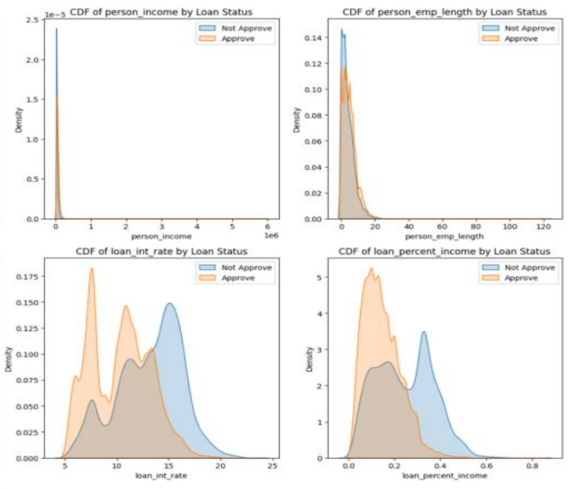



Figure 6: Kernel density distribution of non-object features and target states

For the features of the object class, the histogram is used to observe, and it can be observed that the ‘loan_grade’ feature has a pronounced effect in distinguishing the target values, and the other three features also have a certain distinguishing effect. The feature here is the discrete value, which is easier to classify in decision trees because of its limited range of values. However, the above features are all continuous values, and the boundary between approval and non-approval is not as clear as that of discrete features, which makes the decision tree more susceptible to noise and boundary samples when the continuous values are split.

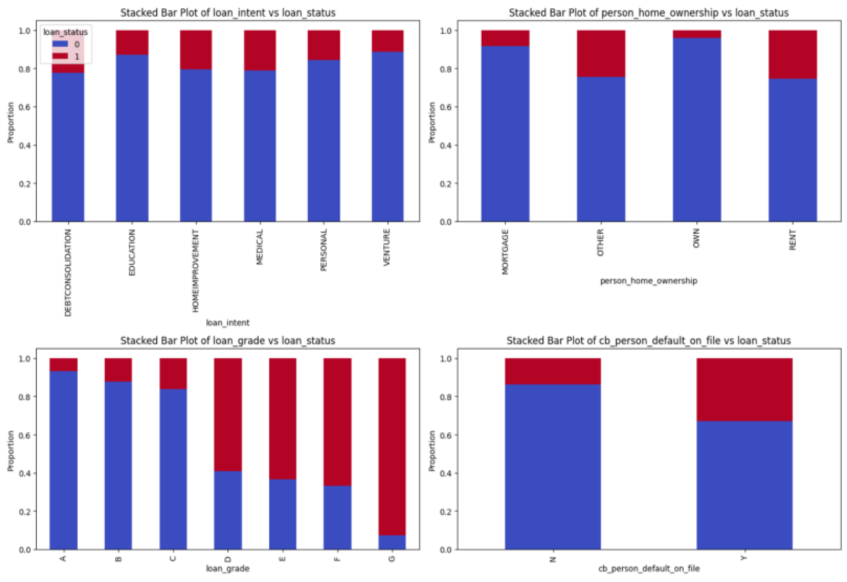
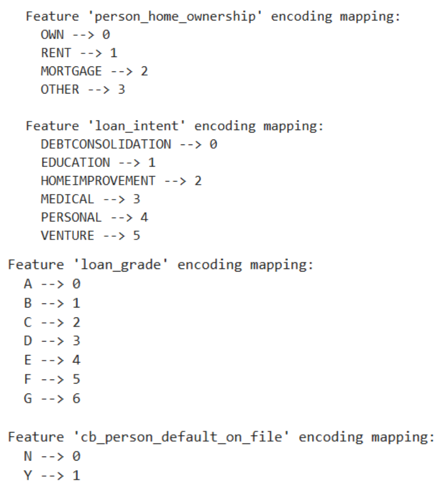


Figure 7: Distribution of object data

2. Data preprocessing

Therefore due to the existence of object types, it is necessary to convert this kind of data into numeric types to facilitate subsequent model training and build new features, which are encoded by Labelencoding.



```
  Feature 'person_home_ownership' encoding mapping:
    OWN --> 0
    RENT --> 1
    MORTGAGE --> 2
    OTHER --> 3

  Feature 'loan_intent' encoding mapping:
    DEBTCONSOLIDATION --> 0
    EDUCATION --> 1
    HOMEIMPROVEMENT --> 2
    MEDICAL --> 3
    PERSONAL --> 4
    VENTURE --> 5
Feature 'loan_grade' encoding mapping:
  A --> 0
  B --> 1
  C --> 2
  D --> 3
  E --> 4
  F --> 5
  G --> 6

Feature 'cb_person_default_on_file' encoding mapping:
  N --> 0
  Y --> 1
```

Figure 8.Object column numeric encoding conversion

After that, we started to check for outliers, where we calculated the IQR of the numeric features in the dataset and plotted a boxplot for each numeric feature, so that we could clearly see the outliers in the data for each feature.

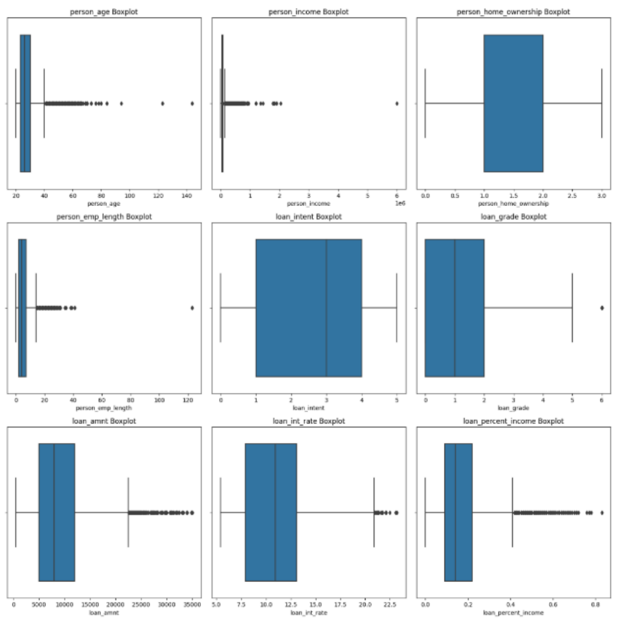


Figure 9: Outliers in the dataset

We initially just used truncation to deal with outliers but finally found that the effect of truncation after removing a very few of the most distant outliers is the best. So in the end we took that approach to dealing with outliers.

The only two columns of missing values in the dataset then begin. After learning and understanding, we chose the KNNimputer method to fill in the missing values.

3. Feature engineering

There are only 11 features in the dataset, which is not enough for model training, so new features need to be expanded. Since this question is the question of whether to approve the loan to the lender which features can influence the obtention of loan, it is necessary to consider the relevant risk issues, and after combining the relevant information such as the rules found in the exploration stage and some national regulations, we will write new characteristics from seven risk directions.

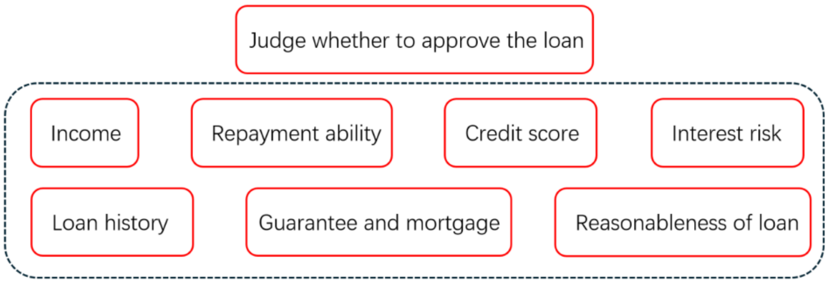


Figure 10: 7 perspectives for judging whether to approve a loan application

Combining the information gained from the previous data exploration section and the approval of the loan consideration, we used the original features to divide, multiply, and judge to create new features, exploring the relationship between the original features and the extent to which they contributed to the final decision. After creating a new feature, the total number of features is 36.

Table 1.Classification of newly constructed features(segmental)

Income	Customize the income to loan ratio	loan_amnt / person_income
	Personal income	log(person_income)
Loan history	Categorized based on the number of historical loans	It is divided into less than 5 times, 5-10 times, and more than 10 times
	Determine whether it is a new borrower	cb_person_cred_hist_length < 2
Reputation score	The relationship between the loan amount and the loan history	loan_amnt / cb_person_cred_hist_length
	Relationship between loan grade and loan history	loan_grade * cb_person_cred_hist_length
Guarantees and mortgages	The loan amount is related to the condition of the house	loan_amnt / home_value_index
Source of repayment and ability to repay	The loan amount is related to the working hours	loan_amnt / (person_emp_length + 1)
	Classified according to working hours	There are three types of working hours: short, medium and long
Reasonableness of the loan application	There is a mismatch between high income and renting	(person_income > person_income.quantile(0.8)) & (person_home_ownership == 'RENT')
	Low income does not match buying a home	(person_income < person_income.quantile(0.2)) & (person_home_ownership == 'OWN')
Interest rate risk	Loan interest rate and loan history	loan_int_rate / cb_person_cred_hist_length
	Interest rate differentials	loan_int_rate - mean(loan_int_rate)

Finally, all the data were normalized. Because the dimensions vary greatly between the features, for example, the income can reach 60 million, but the loan interest rate is only 0.4%, so the data set needs to be normalized.

MODEL

1. Basic model training

Since we want to know which tree model have nice performance, we used common decision tree models for classification, including Randomforest, XGBClassifier, GradientBoostingClassifier, AdaBoostClassifier, BaggingClassifier, LGBMClassifier, CatBoostClassifier。

After each model was given an accuracy, ROC score, training time, and feature contribution, we made a table to compare the effects of the models. It can be observed that XGBClassifier, LGBMClassifier and CatBoostClassifier perform the best, with high accuracy, ROC scores of more than 0.95, and no more than 4 seconds of training time. As a result, we select these three models to be our based model for stacking.

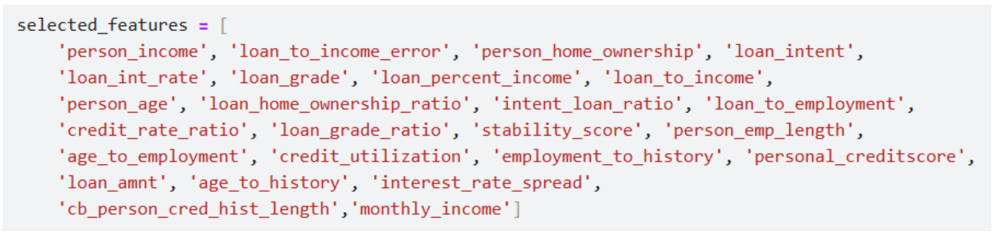
Table 2 Training results of each model

	Accuracy	ROC score	Training time
Randomforest	0.9417	0.936	22.72
XGBClassifier	0.9464	0.9553	2.98
GradientBoostingClassifier	0.9398	0.9364	28.86
AdaBoostClassifier	0.9171	0.9196	14.53
BaggingClassifier	0.9437	0.9419	148.57
LGBMClassifier	0.9479	0.9559	3.08
CatBoostClassifier	0.9465	0.9536	1.97

Based on the comprehensive contribution assessment of each feature by various models, we ultimately retained only 26 features that are beneficial to the stacked model.

Table 3 The top 5 characteristics of comprehensive contribution and the retained features

	AB	GB	LGB	XGB	RF	CB	Mean_Importance
person_income	0.13	0.05	577	0.04	0.05	8.87	97.69
loan_to_income_error	0.19	0.02	270	0.01	0.04	5.34	45.93
person_home_ownership	0.05	0.17	251	0.16	0.06	16.79	44.71
loan_intent	0.08	0.04	252	0.06	0.03	11.97	44.03
loan_int_rate	0.03	0.00	170	0.02	0.05	1.19	28.55



```
selected_features = [
    'person_income', 'loan_to_income_error', 'person_home_ownership', 'loan_intent',
    'loan_int_rate', 'loan_grade', 'loan_percent_income', 'loan_to_income',
    'person_age', 'loan_home_ownership_ratio', 'intent_loan_ratio', 'loan_to_employment',
    'credit_rate_ratio', 'loan_grade_ratio', 'stability_score', 'person_emp_length',
    'age_to_employment', 'credit_utilization', 'employment_to_history', 'personal_creditscore',
    'loan_amnt', 'age_to_history', 'interest_rate_spread',
    'cb_person_cred_hist_length','monthly_income']
```

2. Metamodel training

Here, we optimized the model's hyperparameters using Bayesian search and K-fold cross-validation. Bayesian search builds a probabilistic model to intelligently select hyperparameters, requiring fewer iterations and being more efficient than grid and random search. K-fold cross-validation splits the data into subsets, ensuring each data point is used for both training and validation, providing more reliable evaluation results. Combining these two methods improves the efficiency and accuracy of hyperparameter tuning.

Then, the three models were stacked using StackingClassifier, the model performance was evaluated by 5-fold cross-validation, and finally the test effect was trained on the divided test set. The final training time was about 20 minutes, and the ROC score was 0.9581, which was higher than the original score for each base model.

```
Stacked Model Test ROC AUC Score: 0.9581
```

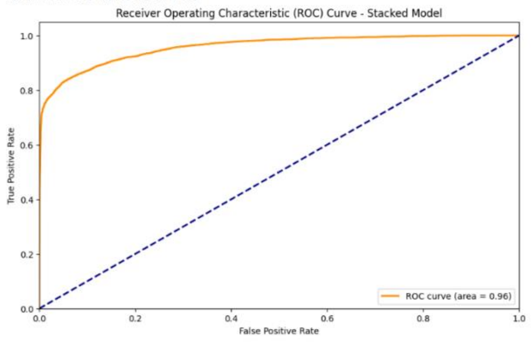
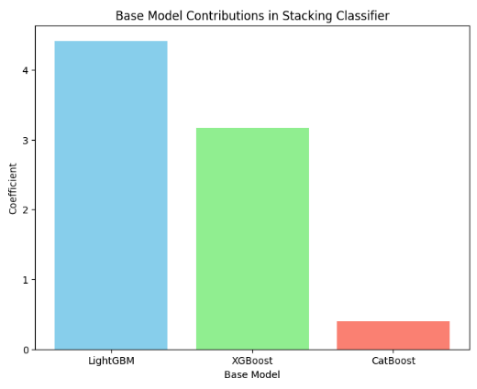


Figure 11.Metamodel training results

Result

Finally, the contribution of each model in the meta-model is observed, and it can be found that LGB contributes the most, while Catboost contributes less. But we also tried to just stack LGB with XGB and it didn't work out any better than this model. we also learned from the data that Catboost is very suitable for processing data with object types, but since the four object types in the dataset have been encoded to construct new features in this experiment, the stacked model here does not reflect the contribution of Catboost well.



```
  Base Model  Coefficient
0   LightGBM     4.414606
1    XGBoost     3.164391
2   CatBoost     0.401089
```

Figure 12. The contribution of each model

The main factor influencing loan approval is **loan grade**, reflecting creditworthiness. **Person income** and **home ownership** follow, showing financial stability and repayment ability. **Loan intent** and **loan percent income** indicate purpose and affordability. **Loan amount** and **interest rate** capture key loan terms and risks. Moderate factors like **monthly income** and **employment** support financial reliability. Overall, borrower traits and loan details strongly impact approval decisions.

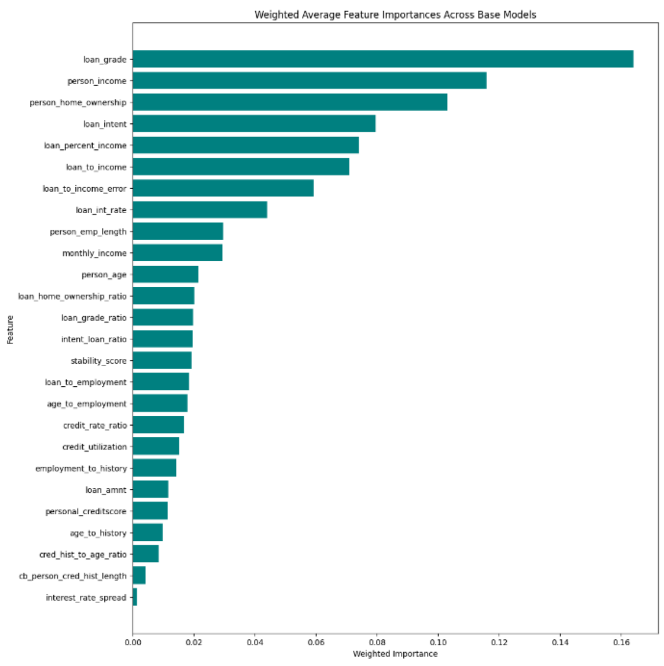


Figure 13.Composite contribution of features

CONCLUSION

Combined with the final comprehensive characteristic score, the question set at the beginning of the experiment can be answered:

1. **Lightgbm**, **Xgboost**, and **Catboost** are the top three best model to be used in this classification question. The stacking model can improve the overall accuracy and LGB contributed the most in it.

2. The bank will mainly judge whether to approve the loan to the applicant based on the borrower's personal **financial situation**, **credit rating**, and **repayment stability**.

BIBLIOGRAPHY

1. Bank of China. (2024). *Personal Loan Contract (2024 Edition)*. Article 6: Loan Repayment,https://pic.bankofchina.com/bocappd/pbservice/202406/P020240628562328417229.pdf
2. Jackson, S. B. (2008), “The Effect of Firms' Depreciation Method Choice on Managers' Capital Investment Decisions,” *The Accounting Review*, 83(2): 351–376.
3. Kirli, M.,(2018), “Comparison of Depreciation Methods in ‘International Accounting Standard 16 Property, Plant and Equipment’ and an Application,” *Economics and Applied Informatics*, 3: 91-98.
4. Malde, P.(2020), “A Comparative Study on Depreciation as Per Companies Act and Income Tax Act in Indian Context,” *Renaissance Journal*, Vol. I, Dec, pp. 50-65.
5. Mark Williams.(2021),"What banks look for when reviewing a loan application",https://www.wolterskluwer.com/en/expert-insights/what-banks-look-for-when-reviewing-a-loan-application
6. Thalassinos, E., Xu, Y.(2021), “Fixed Capital Depreciation Across OECD Countries: A Panel Data Study,” *SpringerLink Journal of Economics Research*, 58(2): 342-358.
7. Tsamis, A., Liapis, K.(2013), “Fair Value and Cost Accounting: Depreciation Methods, Recognition, and Measurement for Fixed Assets,” *International Journal of Economics and Business Administration*, 1(3): 115-123.
8. Yunze Li.(2024),"Order of the State Financial Regulatory Administration(NO.3 of 2024)",https://www.gov.cn/gongbao/2024/issue_11306/202404/content_6947720.html
9. Yunze Li.(2024),"Order of the State Financial Regulatory Administration(NO.2 of 2024)",https://www.gov.cn/gongbao/2024/issue_11306/202404/content_6947721.html

APPENDIX A: CODES

Part1

```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
```

Codes for Italian Method

```
'''Italian Method: Pay the same loan amount each time, interest gets smaller.'''
'''This method is commonly used in China（provided by China Bnak）'''
def italian_method(loan, rate, periods, period_type):
    principal = loan / periods
    balance = loan
    data = []
    for i in range(1, periods + 1):
        interest = balance * rate
        payment = principal + interest
        balance -= principal
        data.append([i, payment, principal, interest, balance])
    return pd.DataFrame(data, columns=[f'Period ({period_type})', 'Payment', 'Principal', 'Interest', 'Balance']).round(4)
```

Codes for French Method

```
'''French Method: Fixed total payment with varying interest and principal portions.'''
'''This method is commonly used in China（provided by China Bnak）'''
def french_method(loan, rate, periods, period_type):
    payment = loan * (rate * (1 + rate) ** periods) / ((1 + rate) ** periods - 1)
    balance = loan
    data = []
    for i in range(1, periods + 1):
        interest = balance * rate
        principal = payment - interest
        balance -= principal
        data.append([i, payment, principal, interest, balance])
    return pd.DataFrame(data, columns=[f'Period ({period_type})', 'Payment', 'Principal', 'Interest', 'Balance']).round(4)
```

Codes for American Method

```
'''American Method: Pay only interest first, then all the loan at once.'''
'''This method is commonly used in China（provided by China Bnak）'''
def american_method(loan, rate, periods, period_type):
    balance = loan
    interest_only_payment = loan * rate
    data = []
    for i in range(1, periods):
        data.append([i, interest_only_payment, 0, interest_only_payment, loan])
    data.append([periods, loan + interest_only_payment, loan, interest_only_payment, 0])
    return pd.DataFrame(data, columns=[f'Period ({period_type})', 'Payment', 'Principal', 'Interest',
'Balance']).round(4)
```

Codes for Geometric Method

```
'''Geometric Method: Payments go up or down at a steady rate.'''
def geometric_method(loan, rate, periods, factor, period_type):
    factor=factor/100
    if period_type == 'monthly':
      factor = (1 + factor) ** (1 / 12) - 1
    elif period_type == 'biannual':
      factor = (1 + factor) ** (1 / 2) - 1
    elif period_type == 'quarterly':
      factor = (1 + factor) ** (1 / 4) - 1
    elif period_type == 'four months':
      factor = (1 + factor) ** (1 / 3) - 1
    elif period_type == 'annual':
        factor = factor
    principal = loan
    q = 1 + factor
    payment = (loan * (rate - factor)) / (1 - (q / (1 + rate)) ** periods)
    balance = principal
    data = []
    for i in range(1, periods + 1):
      if i > 1:
        payment *= q
      interest = balance * rate
      principal = payment - interest
      if i == periods:
          principal = balance
          payment = principal + interest
          balance = 0
      else:
          balance -= principal
      data.append([i, payment, principal, interest, balance])
    return pd.DataFrame(data, columns=[f'Period ({period_type})', 'Payment', 'Principal', 'Interest', 'Balance']).round(4)
```

Codes for Lump-sum Method

```
'''Maturity Lump-Sum Repayment: Pays the entire interest and principal balance in a single lump sum at maturity.'''

'''This method is is commonly used in China（provided by China Bnak）'''

def maturity_lump_sum(loan, rate, periods, period_type='Period'):

    total_interest = loan * rate * periods

    total_payment = loan + total_interest

    data = []

    for i in range(1, periods):

        principal=0

        payment=0

        interest=0

        balance=loan

        data.append([i, payment, principal, interest, balance])

    data.append([periods , total_payment, loan, total_interest, 0])

    return pd.DataFrame(data, columns=[f'Period ({period_type})', 'Payment', 'Principal', 'Interest', 'Balance']).round(4)
```

Main routine

```
loan = float(input("Enter loan amount: "))

rate = float(input("Enter annual interest rate (as a decimal): "))

period_type = input("Please enter the payment frequency (annual, biannual, four months, quarterly, monthly): ").lower()

periods = int(input("Please enter the amortization term (years): "))

method_choice = input("Choose a method (Italian, French, American, Geometric, Lump-sum): ")

factor=0.98

# Adjust the interest rate and number of periods based on the payment frequency

if  period_type == 'biannual':

    rate = rate / 2

    periods = periods * 2

elif period_type == 'four months':

    rate = rate / 3

    periods = periods * 3

elif period_type == 'quarterly':

    rate = rate / 4

    periods = periods * 4

elif period_type == 'monthly':

    rate = rate / 120

    periods = periods * 12

# The methods people can choose
```

```
methods = {
    'Italian': lambda loan, rate, periods: italian_method(loan, rate, periods, period_type),
    'French': lambda loan, rate, periods: french_method(loan, rate, periods, period_type),
    'American': lambda loan, rate, periods: american_method(loan, rate, periods, period_type),
    'Geometric': lambda loan, rate, periods: geometric_method(loan, rate, periods, 0.98, period_type),
    'Lump-sum': lambda loan, rate, periods: maturity_lump_sum(loan, rate, periods, period_type)
}
# Select the chosen repayment method
selected_method = methods[method_choice]
# Geometric method is a little bit different with others, so if the chosen method is the geometric method, prompt for the factor
if method_choice == "Geometric":
    factor = float(input("Enter the factor of the geometric method(This value cannot be greater than one plus the interest rate): "))
    df = geometric_method(loan,rate,periods,factor,period_type)
else:
    df = selected_method(loan, rate, periods)
#Print the amortization table
print("Amortization Table:")
print(df)
export_to_excel(methods, loan, rate, periods)
#Print the total amount table
total_payment = df['Payment'].sum()
total_principal = df['Principal'].sum()
total_interest = df['Interest'].sum()
totals_df = create_totals_df(total_payment, total_principal, total_interest)
totals_df_transposed = totals_df.T
print("\nTotals Table:")
print(totals_df)
#Visualize the result
visualize_methods_as_bar_charts(methods, loan, rate, periods, period_type)
seperate_visualize(methods, loan, rate, periods)
visualize_amortization(methods, loan, rate, periods)
export_to_excel(methods, loan, rate, periods)
```

Part 2

Codes for dealing with outliers

```
df = df[~(df["person_age"] > 100)]
df = df[~(df["person_income"] > 2000000)]
df = df[~(df["person_emp_length"] > 100)]
target="loan_status"
for column in df.select_dtypes(include=['float64', 'int64']).columns:
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 2.5 * IQR
    upper_bound = Q3 + 2.5 * IQR

    if column != target:
        num_outliers_before = ((df[column] < lower_bound) | (df[column] > upper_bound)).sum()
        print(f"Number of outliers in {column} before clipping: {num_outliers_before}")

        df[column] = df[column].clip(lower=lower_bound, upper=upper_bound)

        num_outliers_after = ((df[column] < lower_bound) | (df[column] > upper_bound)).sum()
        print(f"Number of outliers in {column} after clipping: {num_outliers_after}")
```

Codes for KNN

```
imputer = KNNImputer(n_neighbors=5)
df[['person_emp_length', 'loan_int_rate']] = imputer.fit_transform(df[['person_emp_length', 'loan_int_rate']])
print("\nMissing values after imputation:")
print(df[['person_emp_length', 'loan_int_rate']].isna().sum())
```

Codes for features engineering

```
def feature_engineering(df):
    # Income situation
    income_60th_percentile = df['person_income'].quantile(0.6)
    income_90th_percentile = df['person_income'].quantile(0.9)

    def assign_income_level_numeric(income):
        if income < income_60th_percentile:
            return 1  # Low income
        elif income < income_90th_percentile:
            return 2  # Middle income
        else:
```

```
        return 3  # High income

    df['income_level_numeric'] = df['person_income'].apply(assign_income_level_numeric)
    df['loan_to_income'] = df['loan_amnt'] / df['person_income']
    df['loan_to_income_error'] = (df['loan_amnt'] / df['person_income']) - df['loan_percent_income']
    df['loan_to_income_error'] = df['loan_to_income_error'] * 100
    df['loan_percent_income'] = df['loan_percent_income'] * 100
    df['monthly_income'] = df['person_income'] / 12
    df['person_income'] = np.log(df['person_income'])
    df['monthly_income'] = np.log(df['monthly_income'])

    # Loan history records
    def categorize_past_loans_numeric(n):
        if n < 5:
            return 1  # Less than 5 times
        elif 5 <= n <= 10:
            return 2  # 5-10 times
        else:
            return 3  # More than 10 times

    df['loan_history_category_numeric'] =
df['cb_person_cred_hist_length'].apply(categorize_past_loans_numeric)
    df['is_new_credit_user'] = (df['cb_person_cred_hist_length'] < 2).astype(int)

    df['age_to_history'] = df['person_age'] / (df['cb_person_cred_hist_length'] + 1)

    # Credit score
    df['credit_utilization'] = df['loan_amnt'] / df['cb_person_cred_hist_length']
    df['personal_creditscore'] = df['loan_grade'] * df['cb_person_cred_hist_length']
    df['loan_grade_ratio'] = df['loan_amnt'] / (df['loan_grade'] + 1)
    df['cred_hist_to_age_ratio'] = df['cb_person_cred_hist_length'] / df['person_age']

    # Guarantee and mortgage
    df['loan_home_ownership_ratio'] = df['loan_amnt'] / (df['person_home_ownership'] + 1)

    # Repayment sources and ability
    def assign_employment_stability_numeric(emp_length):
        if emp_length <= 1:
            return 1  # Unstable
        elif emp_length <= 5:
            return 2  # Comparatively stable
        else:
            return 3  # Stable

    df['employment_stability_numeric'] = df['person_emp_length'].apply(assign_employment_stability_numeric)
    df['loan_to_employment'] = df['loan_amnt'] / (df['person_emp_length'] + 1)

    df['age_to_employment'] = df['person_age'] / (df['person_emp_length'] + 1)
    df['employment_to_history'] = df['person_emp_length'] / (df['cb_person_cred_hist_length'] + 1)
    df['stability_score'] = (df['person_emp_length'] * df['person_income']) / (df['loan_amnt'] * (df['cb_person_cred_hist_length'] + 1))

    # Rationality of loan application
    df['intent_loan_ratio'] = df['loan_amnt'] / (df['loan_intent'] + 1)
    df['intent_home_match'] = ((df['loan_intent'] == 'HOMEIMPROVEMENT') & (df['person_home_ownership'] == 'OWN')).astype(int)
    df['income_home_mismatch_rich'] = ((df['person_income'] > df['person_income'].quantile(0.8)) & (df['person_home_ownership'] == 'RENT')).astype(int)
    df['income_home_mismatch_poor'] = ((df['person_income'] < df['person_income'].quantile(0.2)) & (df['person_home_ownership'] == 'OWN')).astype(int)

    # Interest rate risk
    df['credit_rate_ratio'] = df['loan_int_rate'] / df['cb_person_cred_hist_length']
    df['interest_rate_spread'] = df['loan_int_rate'] - df['loan_int_rate'].mean()
    df['high_interest_loan'] = (df['loan_int_rate'] > df['loan_int_rate'].quantile(0.75)).astype(int)

    return df

# Apply feature engineering
df = feature_engineering(df)
test = feature_engineering(test)

# View the data after feature engineering
print("\nTransformed Training DataFrame:")
print(df.head())
```

Codes for stacking model

```
# Data separation (based on the selected features and target variable)
selected_features = [
    'person_income', 'loan_to_income_error', 'person_home_ownership', 'loan_intent',
    'loan_int_rate', 'loan_grade', 'loan_percent_income', 'loan_to_income',
    'person_age', 'loan_home_ownership_ratio', 'intent_loan_ratio', 'loan_to_employment',
    'credit_rate_ratio', 'loan_grade_ratio', 'stability_score', 'person_emp_length',
    'age_to_employment', 'credit_utilization', 'employment_to_history', 'personal_creditscore',
    'loan_amnt', 'age_to_history', 'interest_rate_spread', 'cred_hist_to_age_ratio',
    'cb_person_cred_hist_length','monthly_income']


# Separate features and target variable
X = df[selected_features]
y = df[target]

# Split the data into training set, validation set, and test set
X_train_full, X_test, y_train_full, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_train_full, y_train_full, test_size=0.25,
random_state=42, stratify=y_train_full)  # 0.25 x 0.7 = 0.175

# Define the parameter search space for Bayesian optimization
param_grid_lgb = {
    'n_estimators': (100, 500),
    'learning_rate': (0.01, 0.05, 'log-uniform'),
    'num_leaves': (31, 200),
    'max_depth': (5, 15)
}

param_grid_xgb = {
    'n_estimators': (100, 500),
    'learning_rate': (0.01, 0.2, 'log-uniform'),
    'max_depth': (3, 10)
}

param_grid_cat = {
    'iterations': [100, 200],
    'learning_rate': [0.01, 0.05, 0.1],
    'depth': [3, 5, 7]
}
# Bayesian optimization for base models
# LightGBM
lgb_model = lgb.LGBMClassifier(random_state=42)
lgb_search = BayesSearchCV(
    lgb_model,
    param_grid_lgb,
    n_iter=30,
    cv=5,  # Changed to 5-fold cross-validation
    scoring='roc_auc',
    random_state=42
)
lgb_search.fit(X_train, y_train, eval_set=[(X_val, y_val)],
callbacks=[lgb.early_stopping(stopping_rounds=5), lgb.log_evaluation(period=5)])
optimized_lgb = lgb_search.best_estimator_

# XGBoost
xgb_model = xgb.XGBClassifier(eval_metric='logloss', use_label_encoder=False, random_state=42)
xgb_search = BayesSearchCV(
    xgb_model,
    param_grid_xgb,
    n_iter=30,
    cv=5,  # Changed to 5-fold cross-validation
    scoring='roc_auc',
    random_state=42
)
xgb_search.fit(X_train, y_train)
optimized_xgb = xgb_search.best_estimator_
# CatBoost
cat = CatBoostClassifier(silent=True, random_state=42)
cat_search = GridSearchCV(cat, param_grid=param_grid_cat, scoring='roc_auc', cv=3)
cat_search.fit(X_train, y_train)
optimized_cat = cat_search.best_estimator_
# Build a stacking model using the optimized base models
stacking_clf = StackingClassifier(estimators=[
        ('lgb', optimized_lgb),('xgb',optimized_xgb),('cat',
optimized_cat)],final_estimator=LogisticRegression(),cv=3 )

# Evaluate the performance of the stacking model using 2-fold cross-validation
```

```
stratified_cv = StratifiedKFold(n_splits=2)
cv_scores = cross_val_score(stacking_clf, X_train_full, y_train_full, cv=stratified_cv, scoring='roc_auc')
print("Optimized Stacking Classifier CV ROC AUC scores:", cv_scores)
print("Mean CV ROC AUC score:", cv_scores.mean())

# Final evaluation on the test set
stacking_clf.fit(X_train_full, y_train_full)
y_test_pred = stacking_clf.predict_proba(X_test)[:, 1]
fpr, tpr, _ = roc_curve(y_test, y_test_pred)
roc_auc = auc(fpr, tpr)
print(f"Stacked Model Test ROC AUC Score: {roc_auc:.4f}")

# Plot the ROC curve for the test set
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.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('Receiver Operating Characteristic (ROC) Curve - Stacked Model')
plt.legend(loc='lower right')
plt.show()
sub_test = test[selected_features]
# Make predictions on the test set
y_predict_test = stacking_clf.predict(sub_test)
y_pred_proba = stacking_clf.predict_proba(sub_test)[:, 1]  # Extract the predicted probability scores

#print final result
sub['loan_status'] = y_pred_proba
print(sub.head())
sub.to_csv("try20241118.csv", index=False)
```

APPENDIX B: OTHER

PART 1

These tables are generated from the code, which explain the paymen, principal,interest and balance of each amortization method.

Appendix B.1 Italian Amortizaion Method Chart

Period (annual)	Payment	Principal	Interest	Balance
1	150000	100000	50000	900000
2	145000	100000	45000	800000
3	140000	100000	40000	700000
4	135000	100000	35000	600000
5	130000	100000	30000	500000
6	125000	100000	25000	400000
7	120000	100000	20000	300000
8	115000	100000	15000	200000
9	110000	100000	10000	100000
10	105000	100000	5000	0

Total Payment	1275000
Total Principal	1000000
Total Interest	275000

Appendix B.2 French Amortizaion Method Chart

Period (annual)	Payment	Principal	Interest	Balance
1	129504.6	79504.58	50000	920495.4
2	129504.6	83479.8	46024.77	837015.6
3	129504.6	87653.79	41850.78	749361.8
4	129504.6	92036.48	37468.09	657325.3
5	129504.6	96638.31	32866.27	560687
6	129504.6	101470.2	28034.35	459216.8
7	129504.6	106543.7	22960.84	352673.1
8	129504.6	111870.9	17633.65	240802.2
9	129504.6	117464.5	12040.11	123337.7
10	129504.6	123337.7	6166.885	0

Total Payment	1295046
Total Principal	1000000
Total Interest	295045.7

Appendix B.3 American Amortizaion Method Chart

Period (annual)	Payment	Principal	Interest	Balance
1	50000	0	50000	1000000
2	50000	0	50000	1000000
3	50000	0	50000	1000000
4	50000	0	50000	1000000
5	50000	0	50000	1000000
6	50000	0	50000	1000000
7	50000	0	50000	1000000
8	50000	0	50000	1000000
9	50000	0	50000	1000000
10	1050000	1000000	50000	0

Total Payment	1500000
Total Principal	1000000
Total Interest	500000

Appendix B.4 Geometric Amortizaion Method Chart

Period (annual)	Payment	Principal	Interest	Balance
1	124381.4	74381.38	50000	925618.6
2	125600.3	79319.39	46280.93	846299.2
3	126831.2	84516.24	42314.96	761783
4	128074.1	89985	38089.15	671798
5	129329.3	95739.38	33589.9	576058.6
6	130596.7	101793.8	28802.93	474264.8
7	131876.5	108163.3	23713.24	366101.5
8	133168.9	114863.9	18305.08	251237.7
9	134474	121912.1	12561.88	129325.6
10	135791.8	129325.6	6466.278	0

Total Payment	1300124
Total Principal	1000000
Total Interest	300124.4

Appendix B.5 Lump-sum Amortizaion Method Chart

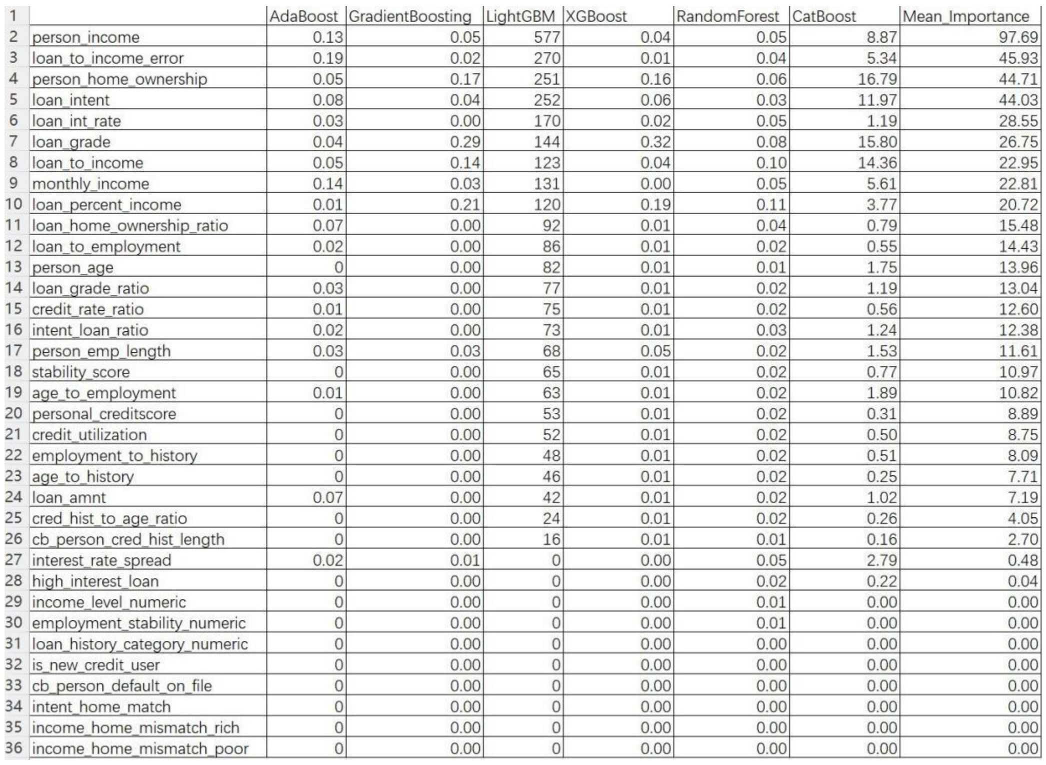
Period (annual)	Payment	Principal	Interest	Balance
1	0	0	0	1000000

2	0	0	0	1000000
3	0	0	0	1000000
4	0	0	0	1000000
5	0	0	0	1000000
6	0	0	0	1000000
7	0	0	0	1000000
8	0	0	0	1000000
9	0	0	0	1000000
10	1500000	1000000	500000	0

Total Payment	1500000
Total Principal	1000000
Total Interest	500000

PART 2

Appendix B.6 Feature importance in models



1		AdaBoost	GradientBoosting	LightGBM	XGBoost	RandomForest	CatBoost	Mean_Importance
2	person_income	0.13	0.05	577	0.04	0.05	8.87	97.69
3	loan_to_income_error	0.19	0.02	270	0.01	0.04	5.34	45.93
4	person_home_ownership	0.05	0.17	251	0.16	0.06	16.79	44.71
5	loan_intent	0.08	0.04	252	0.06	0.03	11.97	44.03
6	loan_int_rate	0.03	0.00	170	0.02	0.05	1.19	28.55
7	loan_grade	0.04	0.29	144	0.32	0.08	15.80	26.75
8	loan_to_income	0.05	0.14	123	0.04	0.10	14.36	22.95
9	monthly_income	0.14	0.03	131	0.00	0.05	5.61	22.81
10	loan_percent_income	0.01	0.21	120	0.19	0.11	3.77	20.72
11	loan_home_ownership_ratio	0.07	0.00	92	0.01	0.04	0.79	15.48
12	loan_to_employment	0.02	0.00	86	0.01	0.02	0.55	14.43
13	person_age	0	0.00	82	0.01	0.01	1.75	13.96
14	loan_grade_ratio	0.03	0.00	77	0.01	0.02	1.19	13.04
15	credit_rate_ratio	0.01	0.00	75	0.01	0.02	0.56	12.60
16	intent_loan_ratio	0.02	0.00	73	0.01	0.03	1.24	12.38
17	person_emp_length	0.03	0.03	68	0.05	0.02	1.53	11.61
18	stability_score	0	0.00	65	0.01	0.02	0.77	10.97
19	age_to_employment	0.01	0.00	63	0.01	0.02	1.89	10.82
20	personal_creditscore	0	0.00	53	0.01	0.02	0.31	8.89
21	credit_utilization	0	0.00	52	0.01	0.02	0.50	8.75
22	employment_to_history	0	0.00	48	0.01	0.02	0.51	8.09
23	age_to_history	0	0.00	46	0.01	0.02	0.25	7.71
24	loan_amnt	0.07	0.00	42	0.01	0.02	1.02	7.19
25	cred_hist_to_age_ratio	0	0.00	24	0.01	0.02	0.26	4.05
26	cb_person_cred_hist_length	0	0.00	16	0.01	0.01	0.16	2.70
27	interest_rate_spread	0.02	0.01	0	0.00	0.05	2.79	0.48
28	high_interest_loan	0	0.00	0	0.00	0.02	0.22	0.04
29	income_level_numeric	0	0.00	0	0.00	0.01	0.00	0.00
30	employment_stability_numeric	0	0.00	0	0.00	0.01	0.00	0.00
31	loan_history_category_numeric	0	0.00	0	0.00	0.00	0.00	0.00
32	is_new_credit_user	0	0.00	0	0.00	0.00	0.00	0.00
33	cb_person_default_on_file	0	0.00	0	0.00	0.00	0.00	0.00
34	intent_home_match	0	0.00	0	0.00	0.00	0.00	0.00
35	income_home_mismatch_rich	0	0.00	0	0.00	0.00	0.00	0.00
36	income_home_mismatch_poor	0	0.00	0	0.00	0.00	0.00	0.00

Appendix B.7 Classification of newly constructed features(full)

Income	Customize the income to loan ratio	loan_amnt / person_income
	Personal income	log(person_income)
	Income Classification	Divided into low, medium, and high income categories
	Deviation Between Original and Custom Ratios	((loan_amnt / person_income) - loan_percent_income) * 100
	Original Ratio Percentage	loan_percent_income * 100
	Average Monthly Income	person_income / 12
Loan history	Categorized based on the number of historical loans	It is divided into less than 5 times, 5-10 times, and more than 10 times
	Determine whether it is a new borrower	cb_person_cred_hist_length < 2
	Age and Loan History Relationship	person_age / (cb_person_cred_hist_length + 1)
Reputation score	The relationship between the loan amount and the loan history	loan_amnt / cb_person_cred_hist_length
	Relationship between loan grade and loan history	loan_grade * cb_person_cred_hist_length
	Income and Loan Grade Relationship	loan_amnt / loan_grade_index
	Loan History and Age Relationship	cb_person_cred_hist_length / person_age
Guarantees and mortgages	The loan amount is related to the condition of the house	loan_amnt / home_value_index
Source of repayment and ability to repay	The loan amount is related to the working hours	loan_amnt / (person_emp_length + 1)
	Classified according to working hours	There are three types of working hours: short, medium and long
	Age and Work Experience Relationship	person_age / (person_emp_length + 1)
	Repayment Stability Coefficient	(person_emp_length * person_income) / (loan_amnt * (cb_person_cred_hist_length + 1))
	Work Experience and Loan History Relationship	person_emp_length / (cb_person_cred_hist_length + 1)
Reasonableness of the loan application	There is a mismatch between high income and renting	(person_income > person_income.quantile(0.8)) & (person_home_ownership == 'RENT')
	Low income does not match buying a home	(person_income < person_income.quantile(0.2)) & (person_home_ownership == 'OWN')
	Loan Amount and Loan Purpose Relationship	loan_amnt / loan_intent
	Loan Purpose and Housing Status Mismatch	(loan_intent == 'HOMEIMPROVEMENT') & (person_home_ownership == 'OWN')
Interest rate risk	Loan interest rate and loan history	loan_int_rate / cb_person_cred_hist_length
	Interest rate differentials	loan_int_rate - mean(loan_int_rate)
	Interest Rate Deviation	loan_int_rate - mean(loan_int_rate)