HO CHI MINH CITY NATIONAL UNIVERSITY
UNIVERSITY OF LAW AND ECONOMICS

FINAL REPORT ON CREDIT RISK MODEL IN R/PYTHON 2023

Subject: Forecasting credit repayment ability of customers

Study program: K20414C_ Fintech

Class ID: 222CN1001

Student: Nguyễn Văn Thành

TT	Full name	Student code	Class	Role	Phone number	Email
1.	Nguyễn Văn Thành	K204140640	K20414 C	Thành viên	0343984998	Thanhnv20414c@st.uel.edu.vn
2.						
3.						
4.						
5.						

HO CHI MINH CITY NATIONAL UNIVERSITY

UNIVERSITY OF LAW AND ECONOMICS

FINAL REPORT ON CREDIT RISK MODEL IN R/PYTHON 2023

Subject: Forecasting credit repayment ability of customers

ASSURANCE

I hereby declare that the report " Forecasting credit repayment ability of customers " is the result of my work under the guidance of PhD. Pham Thi Thanh Xuan, within the framework of the module "Credit Risk Modeling in R/Python".

Contents

LIST OF TABLES

LIST OF CHARTS

LIBRARIES USED IN THE REPORT

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
from sklearn.tree import plot_tree
from sklearn.utils import resample


from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import cross_val_score, KFold, cross_validate,
cross_val_predict
from sklearn.metrics import classification_report, f1_score, precision_score,
recall_score, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,
AdaBoostClassifier, BaggingClassifier
from sklearn.neural_network import MLPClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
import time
```

I. Introduction

1.1. Request

Building a model to predict the creditworthiness of customers using Decision Tree and Random Forest based on the data provided in the class "Credit risk modeling in R/Python".

1.2. Models and tools used in the report

The main tool used in the report is Python used in the Visual Studio Code environment, in addition, Excel and Word are supporting tools to view data and write reports.

The model used in the report is Decision Tree and Random Forest will be the model selected as the matching tool.

Decision Tree is a key tool in predicting the creditworthiness of customers because it can classify customers into groups based on their characteristics, helping to make predictions about the repayment capacity of customers. customers become easier. The Decision Tree can also display the different decisions and choices that lead to the predicted outcome, making it easier for users to understand and interpret. In addition, Decision Tree allows to add new rules or modify existing rules easily, making the forecasting process more flexible and efficient in improving prediction accuracy.

Random Forest was chosen as a tool to match Decision Tree in predicting credit solvency of customers because it has many advantages compared to Decision Tree. While Decision Trees can be prone to overfitting and cannot generalize to data, Random Forest uses multiple decision trees to create a predictive model, which minimizes the effect of noise and increases the generality of the model. Decision trees in Random Forest are built on sub-data sets, which helps to avoid overfitting and increase the generality of the model. In addition, Random Forest also allows to calculate the importance of features, helping users to understand which features are important in predicting the repayment ability of customers. Finally, Random Forest has high accuracy and good predictability, helping users to make more accurate and effective decisions than Decision Tree.

In addition, in the study, other models are used including K-Nearest Neighbors (KNN), Naive Bayes, Support Vector Machine (SVM), Gradient Boosting, Extreme Gradient Boosting (XGBoost), LightGBM, CatBoost, AdaBoost, Bagging, Artificial Neural Networks (ANN) to compare with the main model used in the model to evaluate the results and make the best judgment.

1.3. Data Description

Target variable: “Khả năng trả nợ”, where 0: is a customer paying on time, 1 is a customer not paying on time.

Variable 1: ‘ID khách hàng’, provide customer code.

Variable 2: ‘Giới tính’, 1: male, 0: female.

Variable 3: ‘Quốc tịch’, 1: Việt Nam, 2: Foreign.

Variable 4: ‘Hóa đơn tiền điện’, showing the amount of electricity to be paid by the customer.

Variable 5: ‘Số tiền vay’, represents the customer's loan amount.

Variable 6: ‘Mục đích vay’, represents the customer's loan purpose. Where 1: is for consumption, 2: is to buy a house, 3: is to buy a car, 4: is to study, 5: is to invest in securities.

Variable 7: ‘Gia đình’, shows the marital status of the client. Where 1: Married, 2: Single, 3: Divorced.

Variable 8: ‘Thời gian tại công việc hiện tại’, where 1: is 1 year, 2: is 2 years, 3: is 3 years, 4: is 4 years, 5: is more than 5 years.

Variable 9: ‘Tuổi’, shows the age of the customer.

Vaiable 10: ‘Tài sản đảm bảo’, where 0: is not owned by the borrower, 1: is owned by the borrower.

Variable 11: ‘Nợ xấu’, where 0: is no bad debt, 1: is bad debt.

Variable 12: ‘Khoảng cách đến điểm giao dịch’, in km, showing the customer's distance to the nearest transaction point. Where 1: is less than 10km, 2: is from 10 to 20 km, 3: is from 20 to 30 km, 4 is more than 30 km.

Variable 13: ‘Học vấn’, showing the level of education of the customer. Where 1: is a graduate school, 2: is a university, 3: is a high school, 4: is other.

Variable 14: ‘Thời gian vay’, in monthly, showing the loan period of the customer.

1.4. The process

Chart 1.1. Process

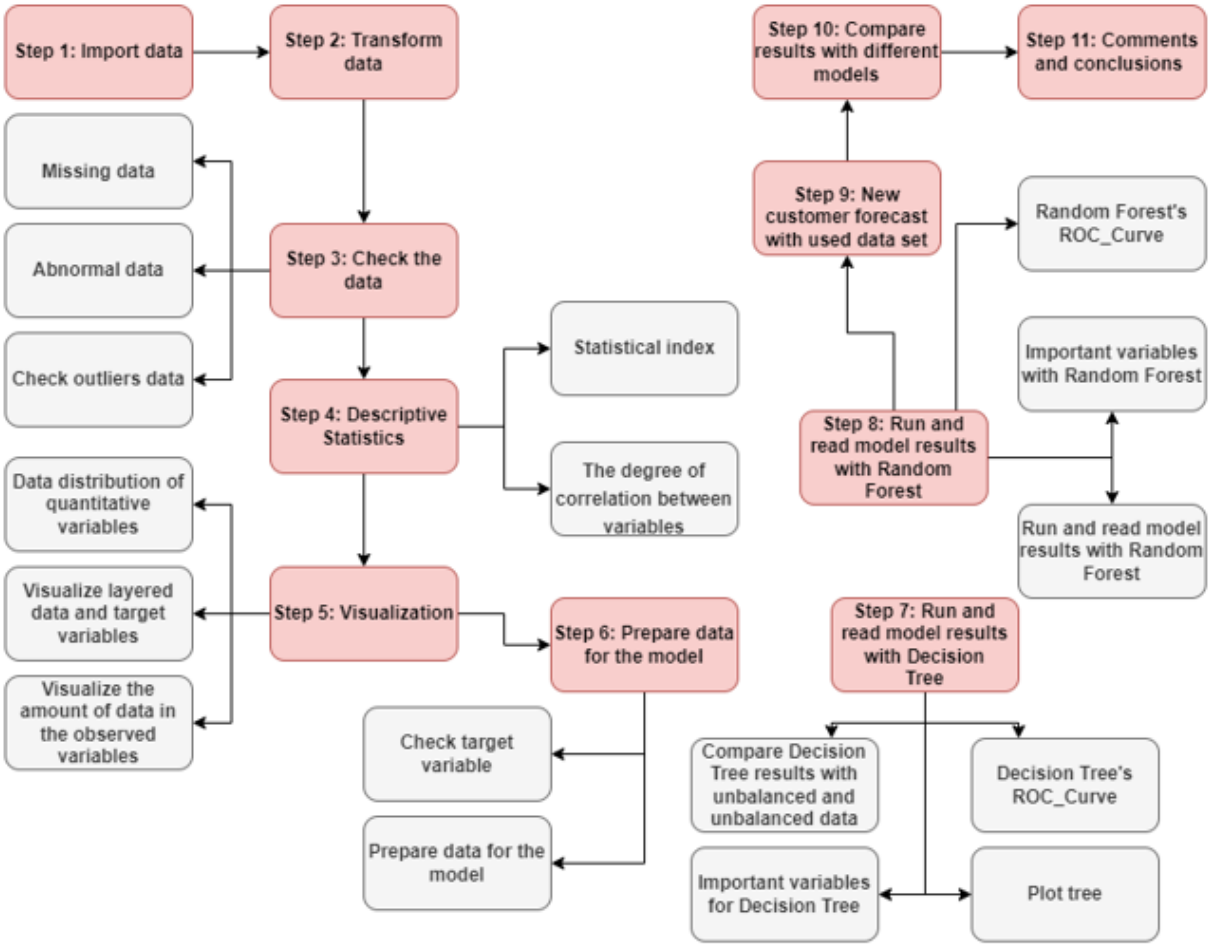


Figure 1 Process

II. Content

2.1. Step 1: Import data

Code box 2.1. Load data

Table 1 Code box 2.1. Load data

```
# Read the data and display the first 5 lines:
df = pd.read_excel('test.xlsx')
df.head()

# Check information and data format:
df.info()
```

Code box results table 2.1: Load data

Table 2 Code box result table 2.1:: Load data

	ID khách hàng	Giới tính	Quốc tịch	Hóa đơn tiền điện	Số tiền vay	Mục đích vay	Gia đình	Thời gian tại công việc hiện tại	Tuổi	Tài sản đảm bảo	Nợ xấu	Khoảng cách đến điểm giao dịch gần nhất (km)	Học vấn	Thời gian vay (tháng)	Khả năng trả nợ
0	1	1. nam	1. Việt Nam	610000	600000000	2. mua nhà	1. Đã kết hôn	3. 3 năm	30	1. Thuộc sở hữu người vay	0. Không có nợ xấu	1. x<10	4. Khác	12	0. Trả nợ đúng hạn
1	2	2. nữ	1. Việt Nam	150000	50000000	4. học tập	2. Độc thân	1. 1 năm	23	0. Không thuộc sở hữu người vay	0. Không có nợ xấu	2. 10<x<20	2. Đại học	6	0. Trả nợ đúng hạn
2	3	2. nữ	2. Nước ngoài	300000	200000000	3. mua xe	2. Độc thân	2. 2 năm	28	1. Thuộc sở hữu người vay	0. Không có nợ xấu	3. 20<x<30	3. Trung học phổ thông	10	0. Trả nợ đúng hạn
3	4	2. nữ	1. Việt Nam	700000	150000000	5. đầu tư chứng khoán	1. Đã kết hôn	5. Trên 5 năm	32	1. Thuộc sở hữu người vay	0. Không có nợ xấu	3. 20<x<30	1. Cao học	12	0. Trả nợ đúng hạn
4	5	1. nam	1. Việt Nam	140000	300000000	3. mua xe	2. Độc thân	1. 1 năm	17	0. Không thuộc sở hữu người vay	1. Có nợ xấu	3. 20<x<30	4. Khác	4	1. Không trả nợ đúng hạn

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 15 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   ID khách hàng                                 3000 non-null   int64
 1   Giới tính                                     3000 non-null   object
 2   Quốc tịch                                     3000 non-null   object
 3   Hóa đơn tiền điện                             3000 non-null   int64
 4   Số tiền vay                                   3000 non-null   int64
 5   Mục đích vay                                  3000 non-null   object
 6   Gia đình                                      3000 non-null   object
 7   Thời gian tại công việc hiện tại              3000 non-null   object
 8   Tuổi                                          3000 non-null   int64
 9   Tài sản đảm bảo                               3000 non-null   object
 10  Nợ xấu                                        3000 non-null   object
 11  Khoảng cách đến điểm giao dịch gần nhất (  km )  3000 non-null   object
 12  Học vấn                                       3000 non-null   object
 13  Thời gian vay (tháng)                         3000 non-null   int64
 14  Khả năng trả nợ                               3000 non-null   object
dtypes: int64(5), object(10)
```

Comment: The data set used in the article includes 15 observation columns and 3000 rows. In which, there are 5 observation columns with integer data type and 10 observation columns with object data type. No missing data is found in the dataset.

2.2. Step 2: Transform data

Code box 2.2. Transform data

Table 3 Code box 2.2. Transform data

```
# Convert data to numeric form
for i in df.columns:
    df[i] = df[i].astype(str).apply(lambda x: x.split('.')[0])
    df[i] = df[i].astype(str).apply(lambda x: x.split(':')[0])

# Convert data to integer
df = df.astype('int64')

# Check information and data format
df.info()
```

Code box result table 2.2: Data conversion

Table 4 Code box result table 2.2. Data conversion

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 15 columns):
 #   Column                                          Non-Null Count  Dtype
---  ------                                          --------------  -----
 0   ID khách hàng                                   3000 non-null   int64
 1   Giới tính                                       3000 non-null   int64
 2   Quốc tịch                                       3000 non-null   int64
 3   Hóa đơn tiền điện                               3000 non-null   int64
 4   Số tiền vay                                     3000 non-null   int64
 5   Mục đích vay                                    3000 non-null   int64
 6   Gia đình                                        3000 non-null   int64
 7   Thời gian tại công việc hiện tại                3000 non-null   int64
 8   Tuổi                                            3000 non-null   int64
 9   Tài sản đảm bảo                                 3000 non-null   int64
 10  Nợ xấu                                          3000 non-null   int64
 11  Khoảng cách đến điểm giao dịch gần nhất (  km )  3000 non-null   int64
 12  Học vấn                                         3000 non-null   int64
 13  Thời gian vay (tháng)                           3000 non-null   int64
 14  Khả năng trả nợ                                 3000 non-null   int64
dtypes: int64(15)
                              memory usage: 351.7 KB
```

Comment: The data has been successfully converted and formatted as integer data.

2.3. Step 3: Check the data

2.3.1. Missing data

Based on the data table in step 1 and step 2, the dataset used in the report has no missing data. However, just to be sure, check again.

Code box 2.3. Check for missing data

Table 5 Code box 2.3. Check for missing data

```
# Calculate percentage of missing data
total = df.isnull().sum().sort_values(ascending=False)
percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)

# Missing percentage= number of missing lines / total number of lines.
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data
```

Code box result table 2.3: Check for missing data

Table 6 Code box result table 2.3: Check for missing data

	Total	Percent
ID khách hàng	0	0.0
Giới tính	0	0.0
Quốc tịch	0	0.0
Hóa đơn tiền điện	0	0.0
Số tiền vay	0	0.0
Mục đích vay	0	0.0
Gia đình	0	0.0
Thời gian tại công việc hiện tại	0	0.0
Tuổi	0	0.0
Tài sản đảm bảo	0	0.0
Nợ xấu	0	0.0
Khoảng cách đến điểm giao dịch gần nhất (km)	0	0.0
Học vấn	0	0.0
Thời gian vay (tháng)	0	0.0
Khả năng trả nợ	0	0.0

Comment: The dataset used in the article certainly has no missing data.

2.3.2. Abnormal data

According to the regulations of the State Bank of Vietnam, the minimum age to borrow credit is 18 years old for each individual, so it will be checked if there are customers under 18 years old in the data set used. or not.

Code box 2.4. Check for invalid data

Table 7 Code box 2.4. Check for invalid data

```
# Check if the dataset has customers under 18 years old
df_18 = df[df['Tuổi'] <= 18]
df_18
```

Code box results box 2.4: Check invalid data

Table 8 Code box results box 2.4: Check invalid data

	ID khách hàng	Giới tính	Quốc tịch	Hóa đơn tiền điện	Số tiền vay	Mục đích vay	Gia đình	Thời gian tại công việc hiện tại	Tuổi	Tài sản đảm bảo	Nợ xấu	Khoảng cách đến điểm giao dịch gần nhất (km)	Học vấn	Thời gian vay (tháng)	Khả năng trả nợ
4	5	1	1	140000	300000000	3	2	1	17	0	1	3	4	4	1
52	53	1	1	264000	167400000	3	2	1	17	0	1	3	4	4	1
89	90	1	1	608000	396400000	3	2	1	17	0	1	3	4	4	1
137	138	1	1	416000	172000000	3	2	1	17	0	1	3	4	4	1
204	205	1	1	592000	462900000	3	2	1	17	0	1	3	4	4	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2782	2783	1	1	504000	376100000	3	2	1	17	0	1	3	4	4	0
2830	2831	1	1	592000	485100000	3	2	1	17	0	1	3	4	12	1
2865	2866	1	1	432000	460600000	5	2	1	17	0	1	3	2	4	1
2913	2914	1	1	648000	324900000	3	2	1	17	0	0	4	4	4	0
2980	2981	1	1	304000	135500000	3	2	1	17	0	0	2	4	6	0

66 rows × 15 columns

Comment: There are 66 customers under the age of 18, not in accordance with regulations. So the decision was made to remove these customers from the data.

Code box 2.5. Get valid data

Table 9 Code box 2.5. Get valid data

```
# Get qualified customer data
df = df[df['Tuổi'] > 18]
df
# Check remaining data percentage
print('Dữ liệu khách hàng còn lại sau khi loại bỏ những khách hàng không đủ điều kiện là:',
df.shape[0]/df_test.shape[0]*100 ,'%')
```

Code box result 2.5: Get valid data

Table 10 Code box result box 2.5: Get valid data

Kết quả:

```
Dữ liệu khách hàng còn lại sau khi loại bỏ những khách hàng không đủ điều kiện
```

```
là: 97.8 %
```

2.3.3. Check outliers data

Outliers need to be identified and eliminated as this is an extremely important step in data processing. The processing of outliers will greatly increase the accuracy of predictive models or business reports. If outliers are detected in the dataset, it will depend on the distribution of the dataset to use a reasonable method (IQR, Z-score, STD ...). In the dataset, the observed variables "Hóa đơn tiền điện", Số tiền vay", "Tuổi" và "Thời gian vay" are in the form of quantitative data, the remaining data is classified or binary data, because so will only check the outliers of the above four observations using boxplot.

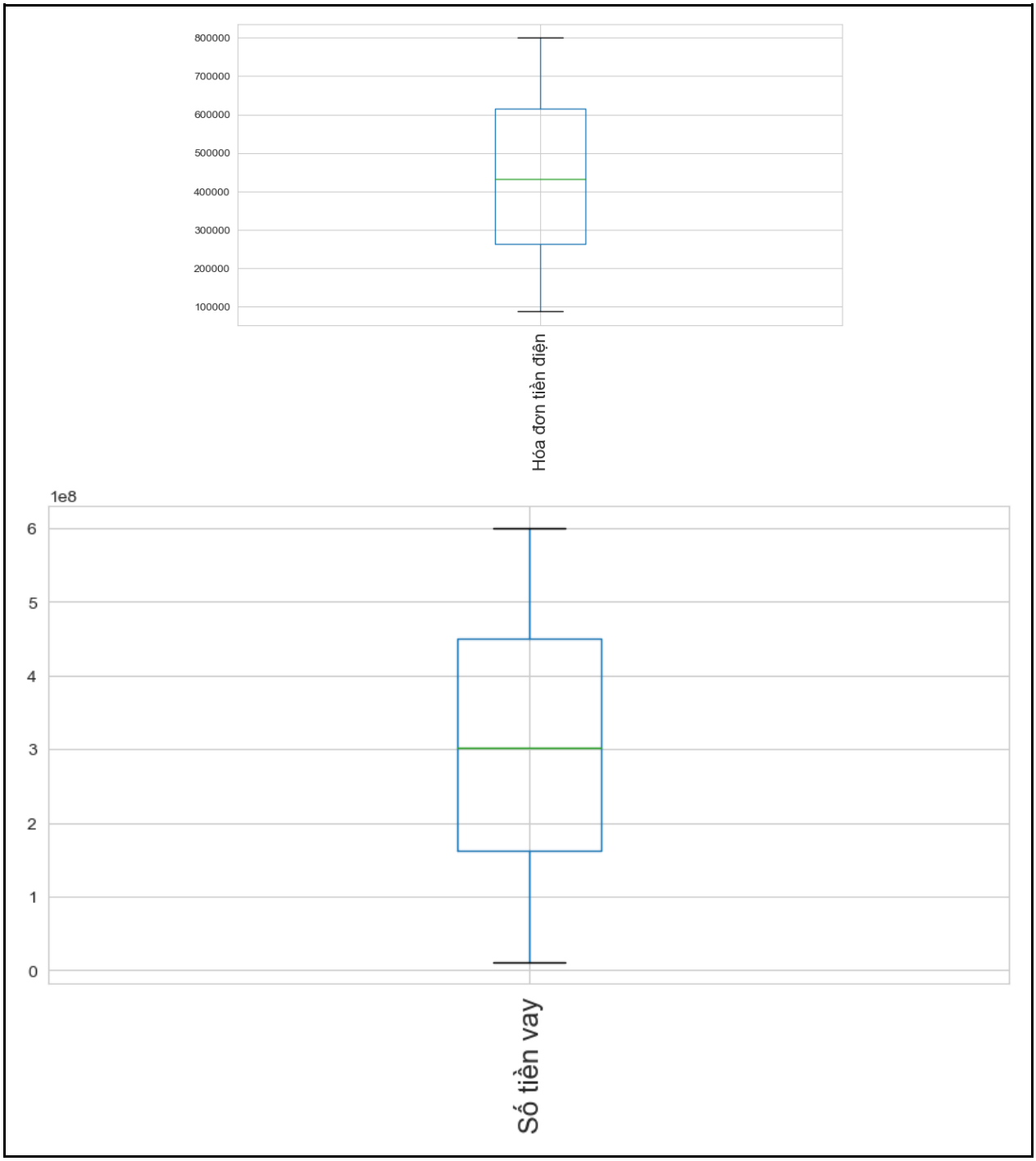
Code box 2.6. Check outliers data

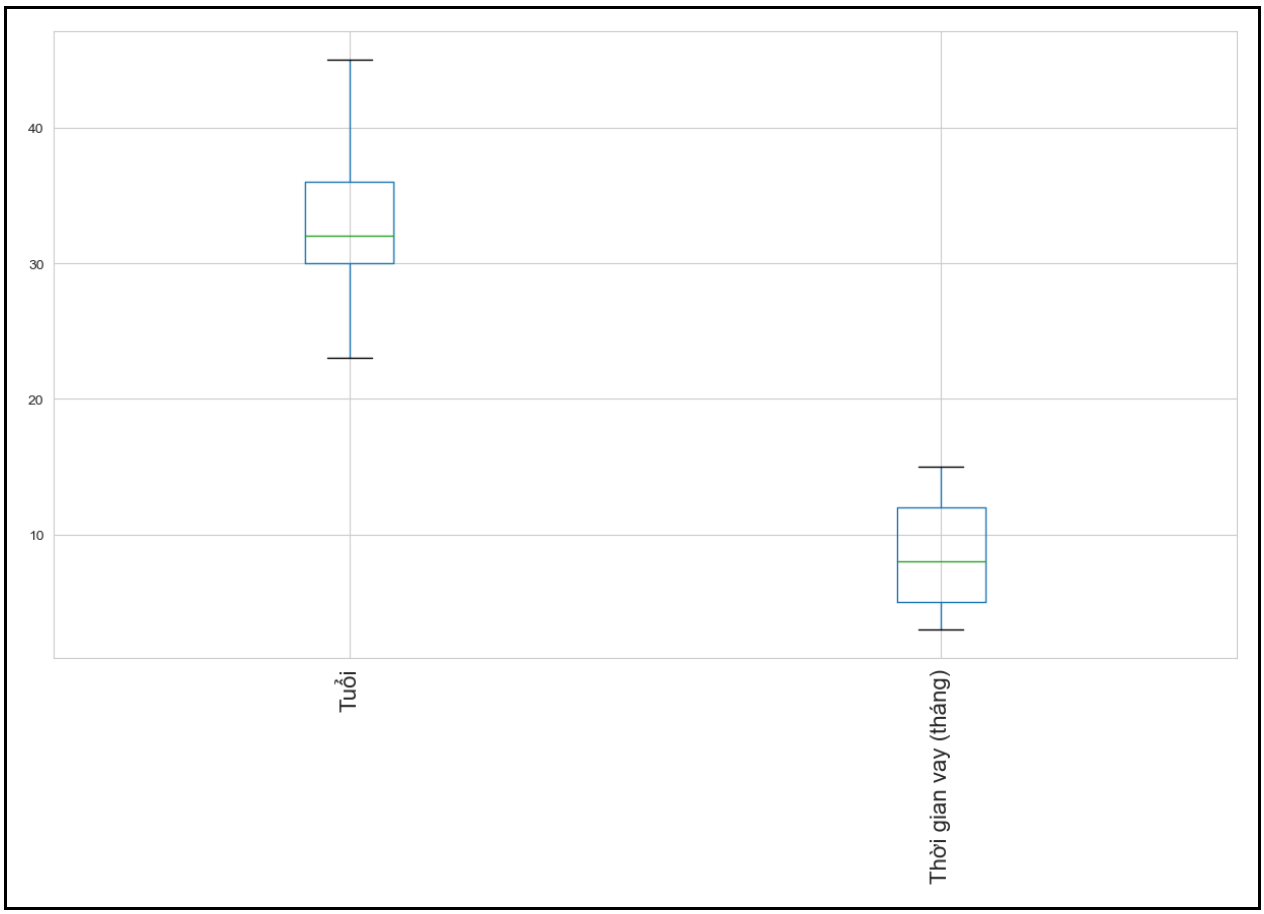
Table 11 Code box 2.6. Check outliers data

```
# Boxplot
# Hóa đơn tiền điện
boxplot = df.boxplot(column=['Hóa đơn tiền điện'], figsize=(10,5), rot=90)
boxplot.tick_params(axis='x', labelsize=16)
# Số tiền vay
boxplot = df.boxplot(column=['Số tiền vay'],figsize=(10,5), rot=90)
boxplot.tick_params(axis='x', labelsize=16)
# Tuổi và Thời gian vay
boxplot = df.boxplot(column=['Tuổi','Thời gian vay (tháng)'], figsize=(15,8), rot=90)
boxplot.tick_params(axis='x', labelsize=16)
```

Result table of code box 2.6: Check outlier data

Table 12 Result table of code box 2.6: Check outlier data





Comment: No outliers appear in the dataset.

2.4. Step 4: Descriptive Statistics

2.4.1. Statistical index

Code box 2.7. Statistical index for continuous variable

Table 13 Code box 2.7. Statistical index for continuous variable

```
continuous_variable = df[['Hóa đơn tiền điện','Số tiền vay','Thời gian tại công
việc hiện tại','Thời gian vay (tháng)','Tuổi']]
continuous_variable.describe()
```

Code box result table 2.7: Statistical index for continuous variable

Table 14 Code box result table 2.7: Statistical index for continuous variable

Kết quả:

	Hóa đơn tiền điện	Số tiền vay	Thời gian tại công việc hiện tại	Thời gian vay (tháng)	Tuổi
count	2934.000000	2.934000e+03	2934.000000	2934.000000	2934.000000
mean	439477.164281	3.058056e+08	3.758010	7.882754	33.294479
std	205596.496872	1.692276e+08	1.171631	3.899595	5.350312
min	88000.000000	1.120000e+07	1.000000	3.000000	23.000000
25%	264000.000000	1.615500e+08	3.000000	5.000000	30.000000
50%	432000.000000	3.021500e+08	4.000000	8.000000	32.000000
75%	616000.000000	4.500000e+08	5.000000	12.000000	36.000000
max	800000.000000	6.000000e+08	5.000000	15.000000	45.000000

Comment: From the above parameters, we can see that the data has a fairly even distribution and the mean values of the attributes do not differ too much from the minimum and maximum values, showing that there is not a big difference between the parameters. data samples, and datasets that can be used to train or build statistical models.

Code box 2.7. Statistical index for discrete variable

Table 15 Code box 2.7. Statistical index

```
discrete_variable = df[['Giới tính','Quốc tịch','Mục đích vay','Gia đình','Tài
sản đảm bảo','Nợ xấu', 'Khoảng cách đến điểm giao dịch gần nhất (  km )','Học
vấn','Khả năng trả nợ']]

for column in discrete_variable.columns:
    value_counts = discrete_variable[column].value_counts()
    value_counts_df = pd.DataFrame({'Value': value_counts.index, 'Count':
value_counts.values})
    value_counts_df['Frequency'] = value_counts_df['Count'] /
len(discrete_variable[column])
    value_counts_df['Percentage'] = value_counts_df['Frequency'] * 100
    print(f"{column}:")
    print(value_counts_df)
    print()
```

Code box result table 2.7: Statistical index for discrete variable

Table 16 Code box result table 2.7: Statistical index

Kết quả:

```
Giới tính:

   Value  Count  Frequency  Percentage
0      2   1539    0.52454   52.453988
```

```
1      1   1395    0.47546   47.546012

Quốc tịch:
   Value  Count  Frequency  Percentage
0      1   2533   0.863327   86.332652
1      2    401   0.136673   13.667348

Mục đích vay:
   Value  Count  Frequency  Percentage
0      3    949   0.323449   32.344922
1      2    759   0.258691   25.869121
2      5    674   0.229721   22.972052
3      1    321   0.109407   10.940695
4      4    231   0.078732    7.873211

Gia đình:
   Value  Count  Frequency  Percentage
0      1   1124   0.383095   38.309475
1      2   1035   0.352761   35.276074
2      3    775   0.264145   26.414451

Tài sản đảm bảo:
...
   Value  Count  Frequency  Percentage
0      0   2079   0.708589   70.858896
1      1    855   0.291411   29.141104
```

2.4.2. The degree of correlation between variables

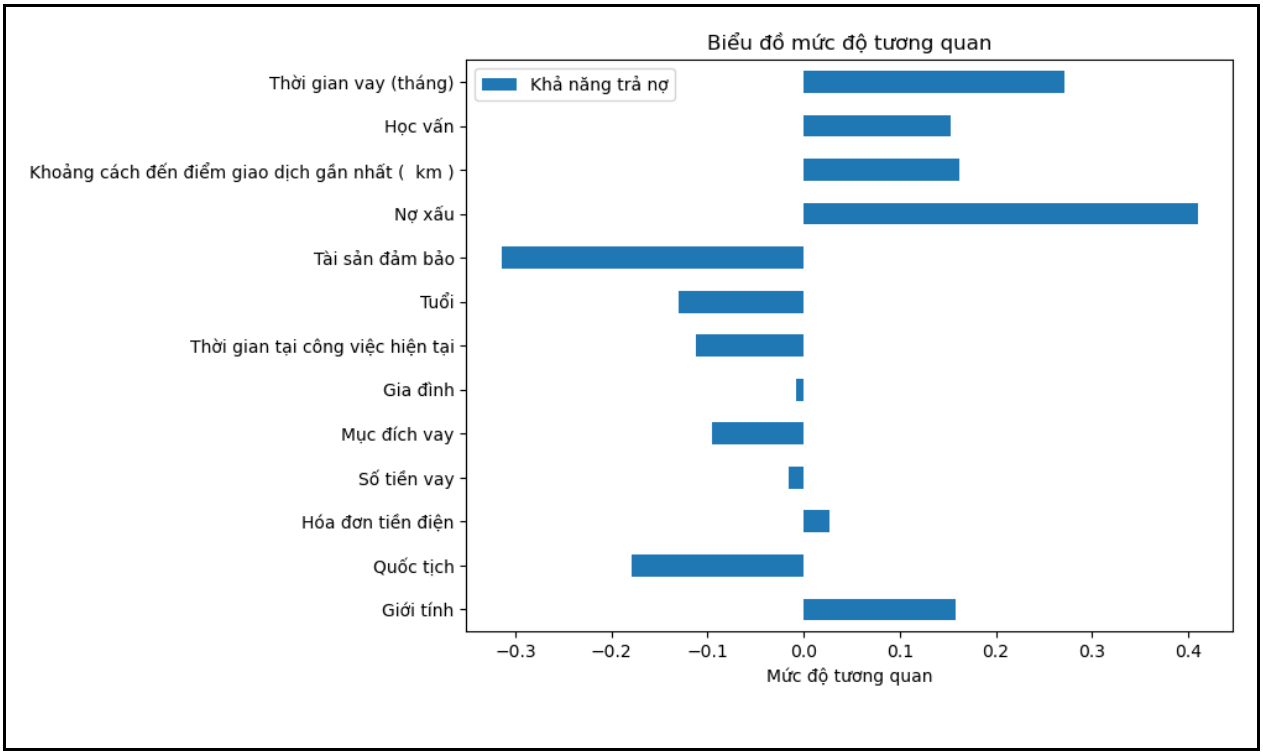
Code box 2.8. Check the degree of correlation between variables

Table 17 Code box 2.8. Check the degree of correlation between variables

```
# correlation
df_corr = pd.DataFrame(df.corr().iloc[1:,-1]).drop(index=['Khả năng trả nợ'])
df_corr
# Plot
ax = df_corr.plot.barh(figsize=(8, 6))
ax.set_xlabel('Mức độ tương quan')
ax.set_title('Biểu đồ mức độ tương quan')
plt.show()
```

Code box result table 2.8: Check the degree of correlation between variables

Table 18 Code box result table 2.8: Check the degree of correlation between variables



Comment: The variables that have the most correlation with the target variable are Loan duration, Bad debt (positive correlation) and collateral (negative correlation), in addition, at least family, loan amount and electricity bills.

2.5. Step 5: Visualization

2.5.1. Data distribution of quantitative variables

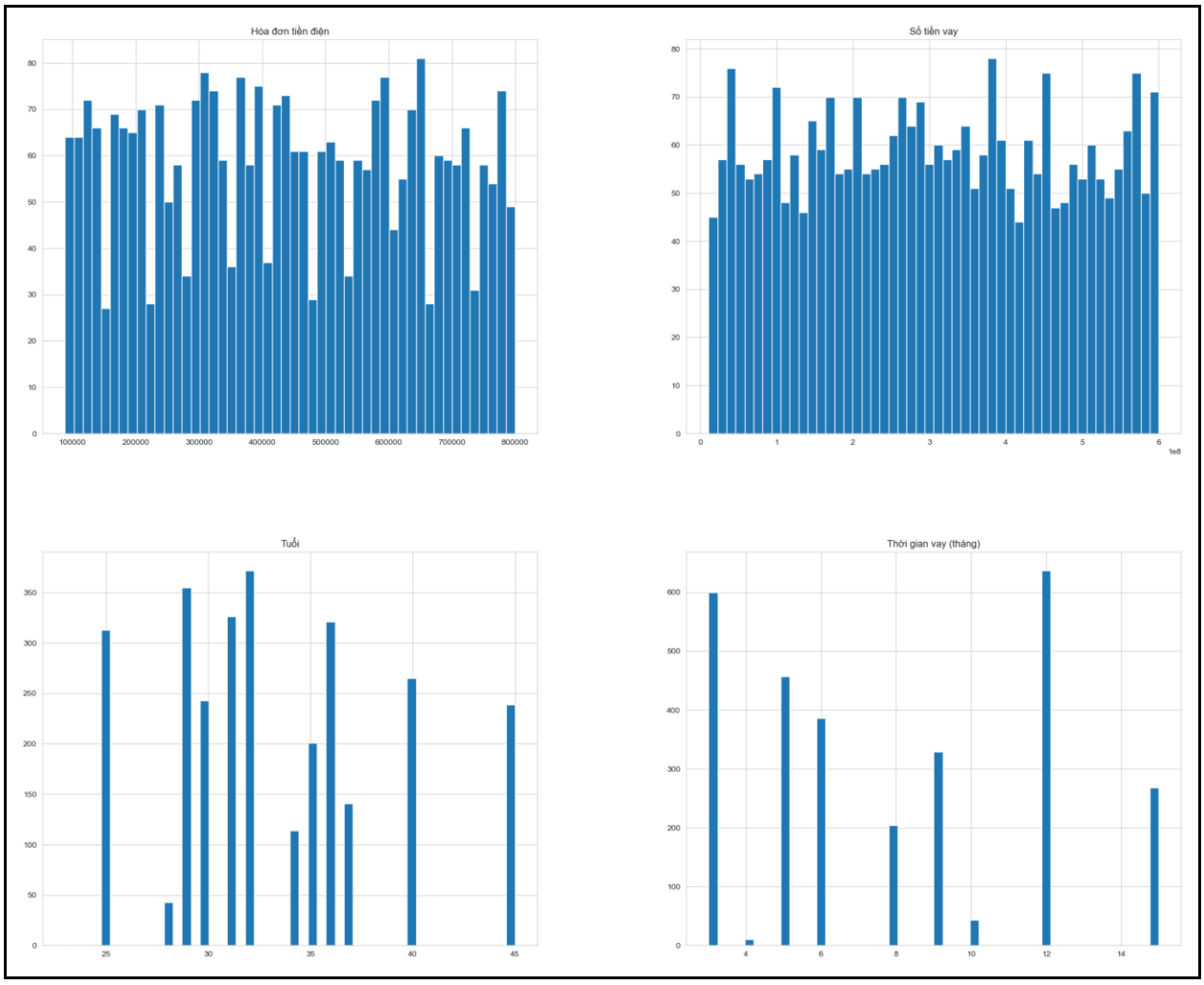
Code box 2.9. Data distribution

Table 19 Code box 2.9. Data distribution

```
%matplotlib inline
# Plot
df.hist(column=['Hóa đơn tiền điện','Số tiền vay','Tuổi','Thời gian vay (tháng)'],bins=50,
figsize=(25,20))
plt.show()
```

Code box result table 2.9: Data distribution

Table 20 Code box result table 2.9: Data distribution



Comment: In the observation of the electricity bill and the loan amount of the customer, the distribution of the amount does not skew towards more or less, in which the amount of electricity bill is in the range (100 - 800 thousand VND), and the loan amount is in the range (100 - 600 million VND). The age group of customers is concentrated mainly in the range of 30 to 40 years old, and the credit loan period ranges from 2 to 15 months, with the largest number of customers taking out loans for 12 months. All observations are not normally distributed.

2.5.2. Visualize layered data and target variables

Code box 2.10. Checking customer's ability to repay debt by gender

Table 21 Code box 2.10. Checking customer's ability to repay debt by gender

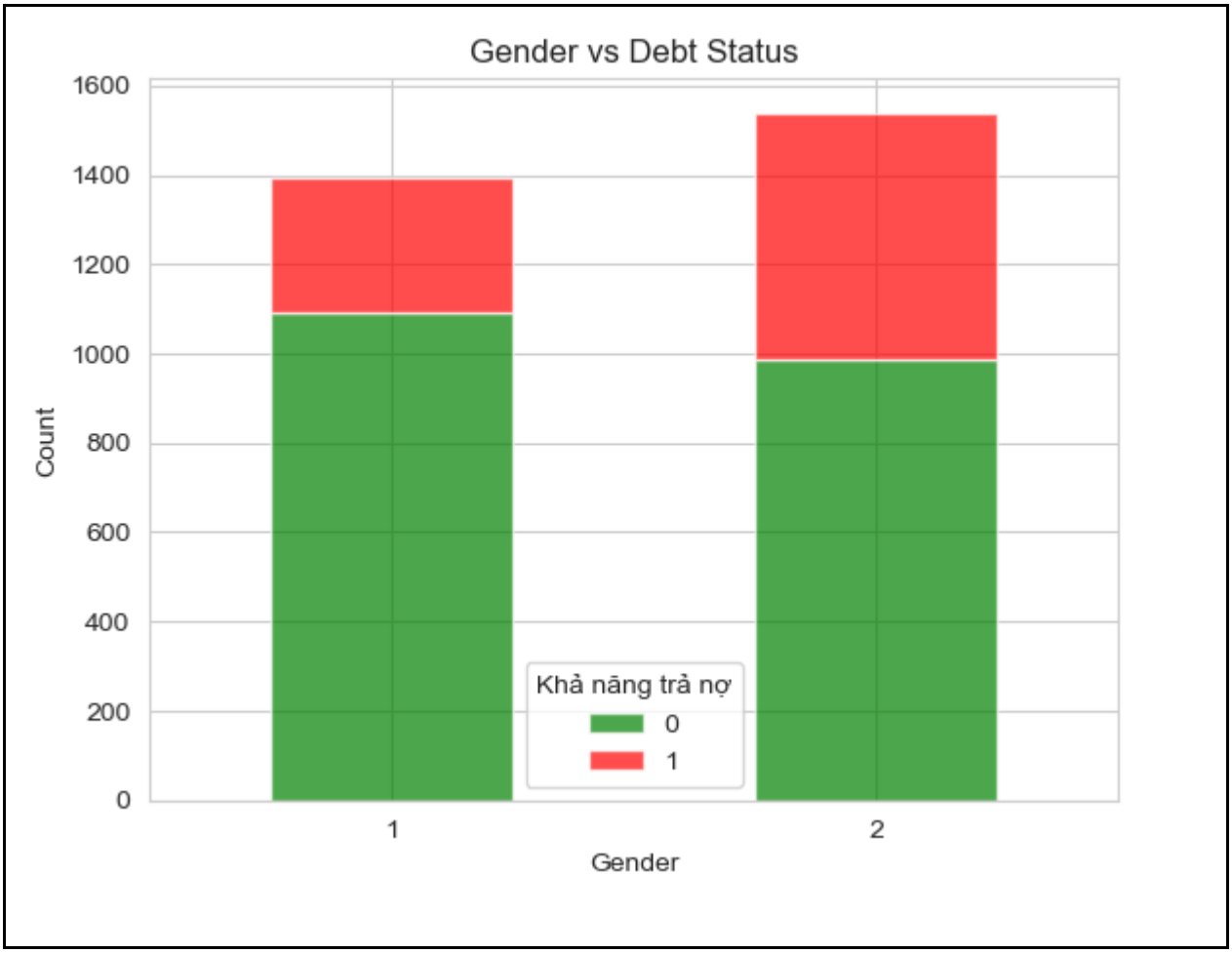
```
# Groupby data by condition
```

```
gender_debt = df.groupby(['Khả năng trả nợ','Giới tính']).agg( Count = ( 'Giới tính','count'
)).reset_index()
gender_debt

# Create pivot table to reformat data
pivot = pd.pivot_table(gender_debt, values='Count', index='Giới tính', columns='Khả năng trả
nợ')

# Barplot
pivot.plot(kind='bar', stacked=True, color=['green', 'red'], alpha=0.7)

# Set title, axis label
plt.title('Gender vs Debt Status')
plt.xlabel('Gender')
plt.ylabel('Count')

# Format x-axis
plt.xticks(np.arange(len(pivot.index)), pivot.index, rotation=0)

# Show chart
plt.show()
```

Code box result table 2.10: Checking customer's ability to repay debt by gender

Table 22 Code box result table 2.10: Checking customer's ability to repay debt by gender

Khả năng trả nợ	0	1
Giới tính		
1	1094	301
2	985	554



Comment: The number of male and female customers is not much different, the number of male customers is less than female customers. The number of male customers paying on time is more than the number of female customers, namely 1094 male customers compared to 985 female customers. In addition, the number of female customers who do not pay on time is more than the number of male customers, namely 554 female customers compared to 301 male customers.

2.5.3. Visualize the amount of data in the observed variables

Code box 2.11. Amount of data in observed variables

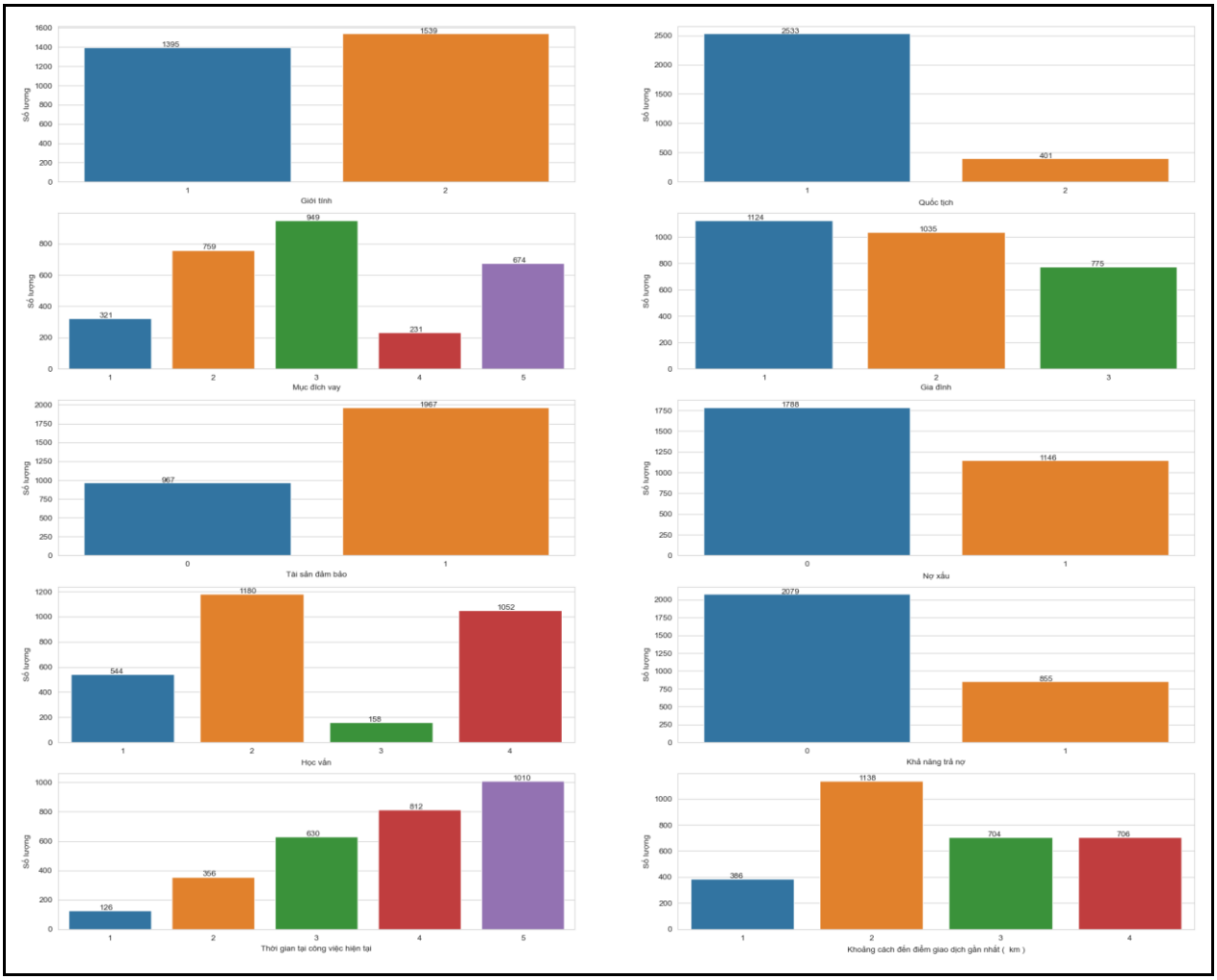
Table 23 Code box 2.11. Amount of data in observed variables

```
# countplot
sns.set_style('whitegrid') # set style for plot

fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(25, 20)) # create a grid 4x2 plot
sns.countplot(x='Giới tính', data=df, ax=axes[0,0])
axes[0,0].set_ylabel('Số lượng')
```

```
for p in axes[0,0].patches:
    axes[0,0].annotate(p.get_height(), (p.get_x()+0.3, p.get_height()+5))

sns.countplot(x='Quốc tịch', data=df, ax=axes[0,1])
axes[0,1].set_ylabel('Số lượng')
for p in axes[0,1].patches:
    axes[0,1].annotate(p.get_height(), (p.get_x()+0.3, p.get_height()+5))
……….
plt.show()
```

Code box result table 2.11: Number of data in observed variables

Table 24 Code box result table 2.11: Number of data in observed variables



Comment: The data shows that out of a total of 2934 customers, there are 1395 male customers and 1539 female customers, of which 401 customers are foreign nationals and 2533 customers are Vietnamese. The main purpose of the customer's loan is to buy personal property. The total number of single customers is 1124, married is 1035 and the rest is divorced with 775 customers. Most of

the client's collateral is owned by themselves, and most of the clients have no bad debt, with 1788 clients having no bad debt. The most common client education is a university degree, and most clients have been in their current job for more than 4 years.

2.6. Step 6: Prepare data for the model

2.6.1. Check target variable

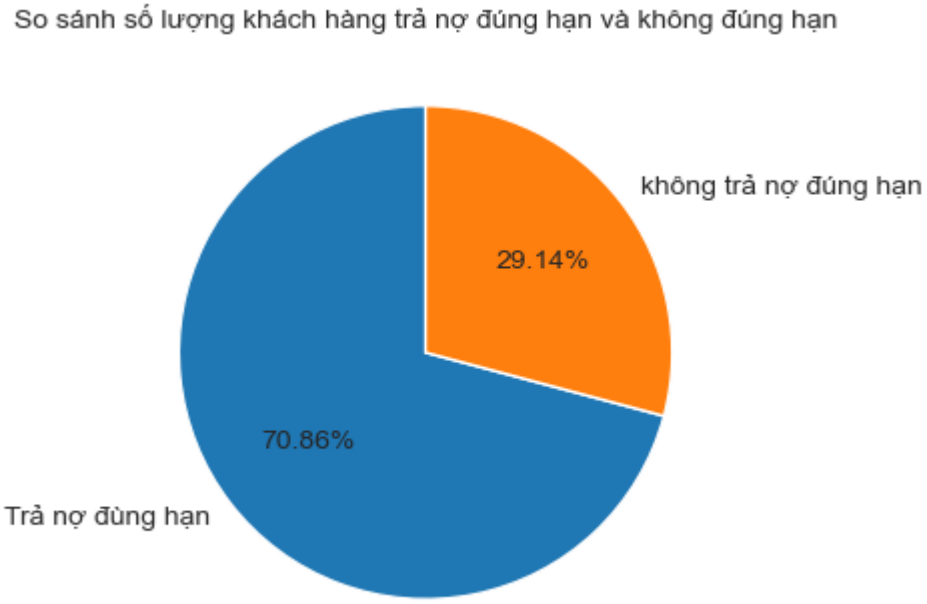
Code box 2.12. Check target variable

Table 25 Code box 2.12. Check target variable

```
# Debt repayment comparison chart
df_dept = df['Khả năng trả nợ'].value_counts()

fig, ax = plt.subplots(figsize=(6, 4))
ax.pie(df_dept, labels=['Trả nợ đùng hạn','không trả nợ đúng hạn' ], autopct='%2.2f%%',
startangle=90)
plt.title("So sánh số lượng khách hàng trả nợ đúng hạn và không đúng hạn", size =10)
plt.show()
```

Code result table 2.12: Check target variable

Table 26 Code result table 2.12: Check target variable



Comment: The current data are unbalanced, since the number of samples of class 1 is significantly less than that of class 0 (70.86% and 29.14%). So, to solve this problem, oversampling method is suitable to solve this problem. Oversampling is used to increase the sample size for the minority class.

Code box 2.13. Data balance

Table 27 Code box 2.13. Data balance

```
# Rebalance data
class_0 = df[df['Khả năng trả nợ'] == 0]
class_1 = df[df['Khả năng trả nợ'] == 1]
class_1_oversampled = resample(class_1,
                        replace=True,
                        n_samples=len(class_0),random_state=42)
# merge
balanced_data = pd.concat([class_0, class_1_oversampled])
# Double check the number of observations of each class
print(balanced_data['Khả năng trả nợ'].value_counts())
```

Code box result table 2.13: Data balance

Table 28 Code box result table 2.13: Data balance

```
0    2079
1    2079
Name: Khả năng trả nợ, dtype: int64
```

Comment: The data has been balanced with a count of 2079 for both forecast classes.

2.6.2. Prepare data for the model

Code box 2.14. Prepare forecast data for the model

Table 29 Code box result table 2.14. Prepare forecast data for the model

```
# Select target variable
target = ['Khả năng trả nợ']
features = list(set(list(balanced_data.columns)) - set(target))

# Balance data
balanced_data = balanced_data.drop(columns=['ID khách hàng'])
X = balanced_data[features].values
y = balanced_data[target].values
```

```
# Unbalanced data
df = df.drop(columns=['ID khách hàng'])

X_ = balanced_data[features].values
y_ = balanced_data[target].values
```

Code box 2.15: Normalize data

Table 30 Code box 2.15: Normalize data

```
# Balanced data normalization
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Unbalanced data normalization
scaler = StandardScaler()
X_ = scaler.fit_transform(X_)
```

2.7. Step 7: Run and read model results with Decision Tree

2.7.1. Compare Decision Tree results with unbalanced and unbalanced data

Code box 2.16. Run and read model results with Decision Tree (unbalanced data)

Table 31 Code box 2.16. Run and read model results with Decision Tree (unbalanced data)

```
# Chia tập dữ liệu
X_ = df[features].values
y_ = df[target].values
X_train, X_test, y_train, y_test = train_test_split(X_, y_, test_size = 0.1, random_state=42)

# Chạy mô hình
DT_classifier = DecisionTreeClassifier()
DT_classifier.fit(X_train, y_train.ravel())

y_pred = DT_classifier.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print('Decision Tree accuracy: ', accuracy_score(y_test, y_pred))
```

Code box results table 2.16: Run and read model results with Decision Tree (unbalanced data)

Table 32 Code box results table 2.16: Run and read model results with Decision Tree

```
[[183  26]
 [ 23  62]]
              precision    recall  f1-score   support

           0       0.89      0.88      0.88       209
           1       0.70      0.73      0.72        85

    accuracy                           0.83       294
   macro avg       0.80      0.80      0.80       294
weighted avg       0.84      0.83      0.83       294

                 Decision Tree accuracy:  0.8333333333333334
```

Code box 2.17. Run and read model results with Decision Tree (balanced data)

Table 33 Code box 2.17. Run and read model results with Decision Tree (balanced data)

```
# Split dataset
X = balanced_data[features].values
y = balanced_data[target].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1, random_state=42)

# Run the model
DT_classifier = DecisionTreeClassifier()
DT_classifier.fit(X_train, y_train.ravel())

y_pred = DT_classifier.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print('Decision Tree accuracy: ', accuracy_score(y_test, y_pred))
```

Code box results table 2.17: Run and read model results with Decision Tree (balanced data)

Table 34 Code box results table 2.17: Run and read model results with Decision Tree

```
[[202  29]
 [ 7  178]]
              precision    recall  f1-score   support

           0       0.97      0.87      0.92       231
           1       0.86      0.96      0.91       185

    accuracy                           0.91       416
   macro avg       0.91      0.92      0.91       416
weighted avg       0.92      0.91      0.91       416
```

```
Decision Tree accuracy:  0.9134615384515384
```

Comment:

Comparing two confusion matrices on balanced and unbalanced data we have:

The confusion matrix on the unbalanced data shows that the Decision Tree model correctly predicted 246 out of 294 samples, equivalent to an accuracy rate of 83.7%. In which, the number of incorrect predictions is the number of samples belonging to the class of false positives with 25 cases. This can be explained by the lack of balance between the two data classes, making it easy for the model to predict many patterns of the class that are not present in the data set. The confusion matrix on the balanced data shows that the Decision Tree model correctly predicted 380 out of 416 samples, equivalent to an accuracy rate of 91.3%. The correct ratio of the model on balanced data is much higher than on unbalanced data, showing that data balance is very important in model building. The model's most false prediction on balanced data is the number of samples belonging to the false negative class with 7 cases, but this number has fallen sharply compared to the unbalanced data. .

Table 2.1: comparing the results of Decision Tree on 2 data sets

Table 35 Table 2.1: Comparing the results of Decision Tree on 2 data sets

Score	Imbalance	Balance
Accuracy	0.837	0.913
Precision của lớp 0	0.89	0.97
Recall của lớp 0	0.88	0.87
F1-score của lớp 0	0.88	0.92
Precision của lớp 1	0.71	0.86
Recall của lớp 1	0.73	0.96
F1-score của lớp 1	0.72	0.91

On balanced data, the Decision Tree model achieves high accuracy (0.913) and the F1-scores of both classes are high (0.92 and 0.91). Both of these metrics are higher than on unbalanced data (accuracy is 0.837 and F1-scores for classes 0 and 1 are 0.88 and 0.72, respectively).

Precision of class 0 on balanced data achieved very good results (0.97), while on unbalanced data achieved only 0.89. Precision of class 1 on balanced data is also higher (0.86) than on unbalanced data (0.71).

Class 0's recall on balanced and unbalanced data is quite close (0.87 and 0.88), while class 1's recall on balanced data achieves very high results (0.96) and increases compared to lossy data. balanced (0.73). Conclusion: Using balanced data helps the Decision Tree model to achieve higher accuracy and better classification for both classes. However, to evaluate the effectiveness of the model, it is necessary to consider other indicators such as AUC, ROC curve, and Confusion Matrix.

Conclusion: Using balanced data helps Decision Tree model to achieve higher accuracy and better classifier for both classes.

2.7.2. Important variables for Decision Tree

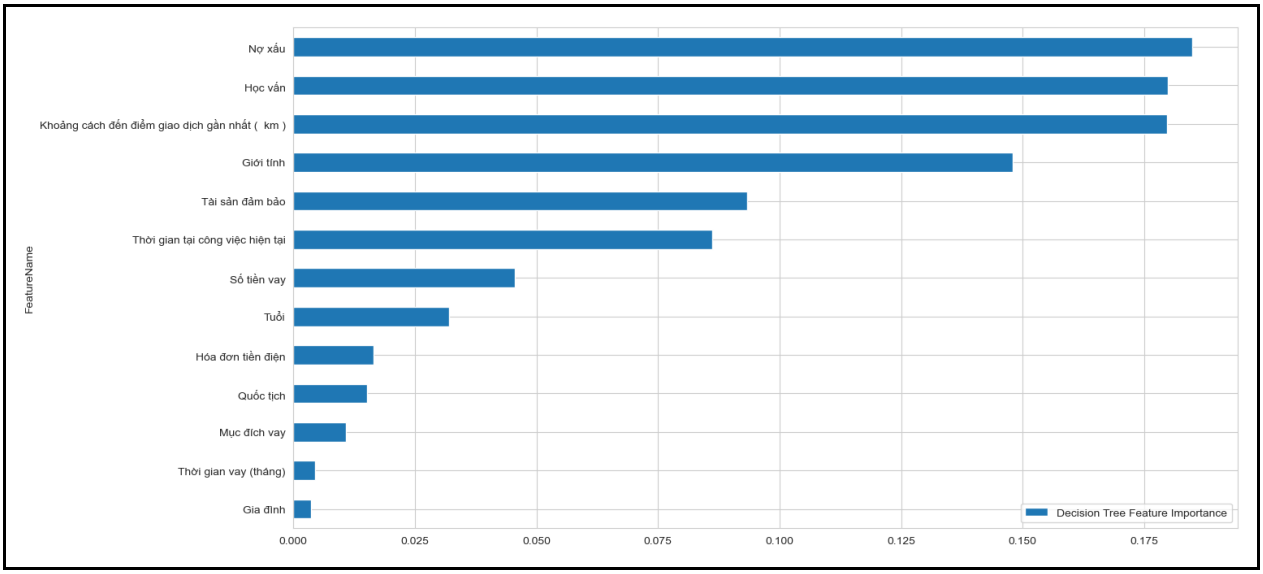
Code box 2.18. Find variables with strong influence on Decision Tree

Table 36 Code box 2.18. Find variables with strong influence on Decision Tree

```
# Find important variables
importance_dt = DT_classifier.feature_importances_
features_importances_dt = pd.DataFrame({'FeatureName':
balanced_data.columns[0:len(balanced_data.columns)-1], 'Decision Tree Feature Importance':
importance_dt})
features_importances_dt.sort_values(by=['Decision Tree Feature Importance'],
ascending=False)
# Plot
features_importances_dt.sort_values("Decision Tree Feature Importance").plot(figsize=(15,8),
x="FeatureName", y=["Decision Tree Feature Importance"], kind="barh")
```

Code box results table 2.18: Run and read model results with Decision Tree (balanced data)

Table 37 Code box results table 2.18: Run and read model results with Decision Tree



Comment: The variables of the data set used in the report that have the most influence on the forecast results of Decision Tree are bad debt, education, distance to the nearest transaction point, gender, financial position, etc. guarantee and time at current job. This means that, if we only use these variables for the Decision Tree model, we can improve the prediction results.

2.7.3. Decision Tree's ROC_Curve

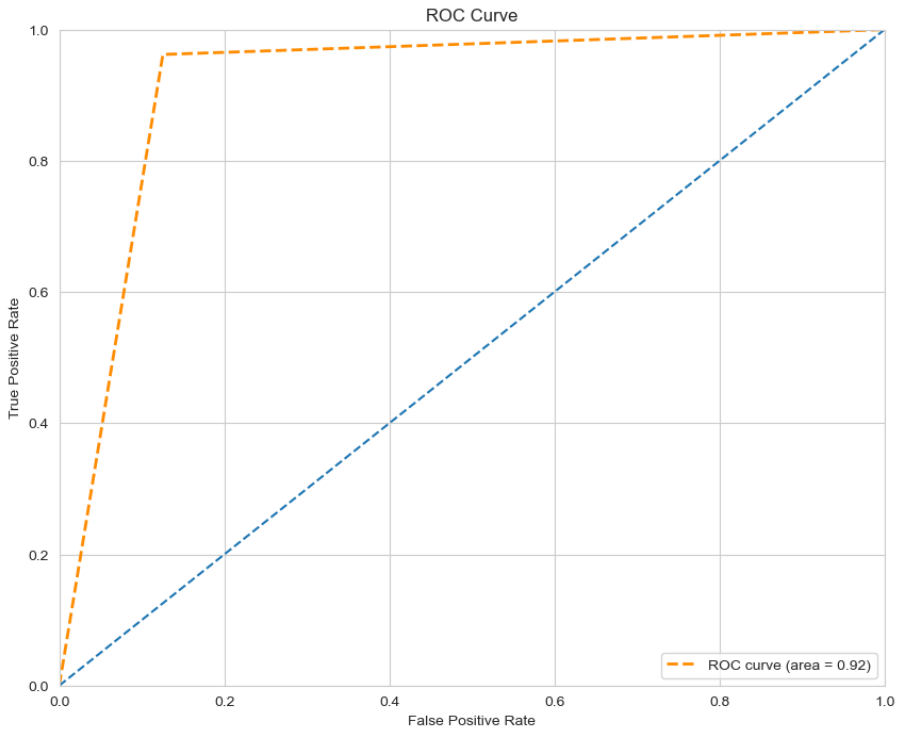
Code box 2.19. Plot Decision Tree's ROC_Curve

Table 38 Code box 2.19. Plot Decision Tree's ROC_Curve

```
from sklearn.metrics import roc_curve, auc

def _plot_roc_curve(fpr, tpr, thres, auc):
    plt.figure(figsize = (10, 8))
    plt.plot(fpr, tpr, 'b-', color='darkorange', lw=2, linestyle='--', label='ROC curve (area = 
%0.2f)'%auc)
    plt.plot([0, 1], [0, 1], '--')
    plt.axis([0, 1, 0, 1])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc='lower right')
    plt.title('ROC Curve')


y_pred_prob_test = DT_classifier.predict_proba(X_test)[:, 1]
fpr, tpr, thres = roc_curve(y_test, y_pred_prob_test)
roc_auc = auc(fpr, tpr)

_plot_roc_curve(fpr, tpr, thres, roc_auc)
```

Code box result table 2.19: Plot Decision Tree's ROC_Curve

Table 39 Code box result table 2.19: Plot Decision Tree's ROC_Curve



Comment: Based on the chart, it can be seen that the ROC curve is plotted and the area under the ROC curve is 0.92. This shows that the classification model has a good performance, as the area under the ROC curve is between 0.5 and 1.0 and the closer to 1, the better the model. So Decision Tree has good classification ability.

2.7.4. Plot tree

Code box 2.20. Plot Tree

Table 40 Code box 2.20. Plot Tree

```
# Check target variable definition
balanced_data['Khả năng trả nợ'].unique()
# Select input variables and output variables
X_tree = balanced_data[['Nợ xấu','Học vấn','Khoảng cách đến điểm giao dịch gần nhất (  km )','Giới tính','Tài sản đảm bảo','Thời gian tại công việc hiện tại']]
y_tree = balanced_data['Khả năng trả nợ']
# Name of input variables
features = ['Nợ xấu','Học vấn','Khoảng cách đến điểm giao dịch gần nhất (  km )','Giới tính','Tài sản đảm bảo','Thời gian tại công việc hiện tại']
# Create a Decision Tree classifier with selected variables
clf = DecisionTreeClassifier(max_depth=3)
# Train the model with input data and output labels
clf.fit(X_tree, y_tree)
# Plot decision tree
plt.figure(figsize=(20,10))
plot_tree(clf, fontsize=10, filled=True, feature_names=features, class_names=["Trả nợ đúng hạn","Không trả nợ đúng hạn"])
plt.show()
```

Code box result table 2.20: Plot Tree

Table 41 Code box result table 2.20: Plot Tree

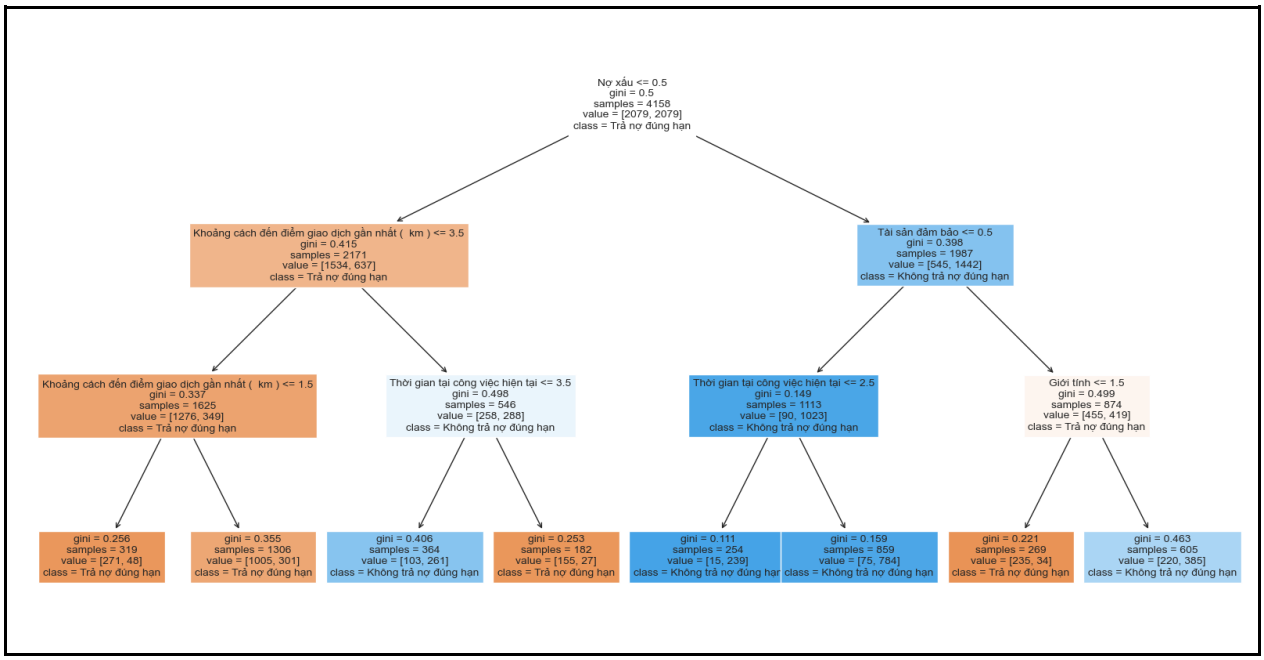


Chart 2.1. Name the branches in the tree

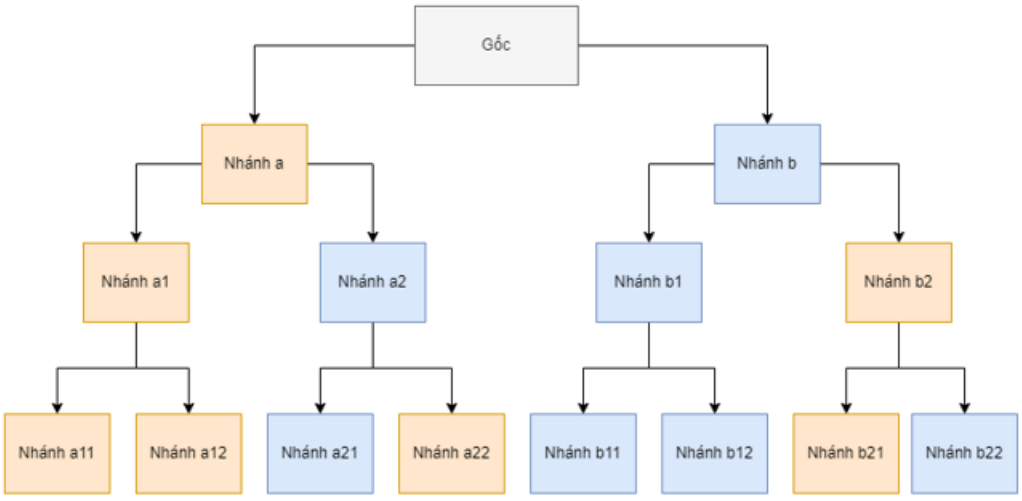


Figure 2 Chart 2.1. Name the branches in the tree

Comment: This decision tree has a root with the following information: Bad debt <= 0.5, gini = 0.5, samples = 4158, values = [2079, 2079].

This root has two branches, branch a and branch b. Branch a is divided from the root with the criteria being Distance to the nearest transaction point <= 3.5, gini = 0.415, samples = 2171, value = [1543, 637], Class = On-time repayment. This branch is again divided into two sub-branches,

branch a1 and a2. Branch a1 is divided into two sub-branches, branch a11 and a12. Branch a2 is also divided into two sub-branches, branch a21 and a22.

Branch b is divided from the root with the criteria as collateral <= 0.5, gini = 0.398, samples =1987, values = [545, 1442], class = default. This branch is again divided into two sub-branches, branch b1 and b2. Branch b1 is again divided into two sub-branches, branch b11 and b12. Branch b2 is divided into two sub-branches, branch b21 and b22.

Each sub-branch will have a gini index to evaluate the categorization of the data in that sub-branch. The lower the gini index, the better the classification of the data in that sub-branch. For branching, the decision tree uses different criteria.

Example of tree operation: At the root, the decision tree is branched into two sub-branches a and b based on the bad debt criterion <= 0.5. Branch a is further branched into two sub-branches a1 and a2 based on the criteria of distance to the nearest transaction point and gini index value. Branch a1 has a lower gini index value than branch a2, indicating a more accurate classification for the data group in branch a1. Therefore, the decision tree will continue to branch at a1 to create two sub-branches a11 and a12 based on the criteria of distance to nearest transaction point and gini index value. In branch a11, the gini index value is even lower than in branch a12, indicating a more accurate classification for the data group in branch a11. Therefore, the decision tree will continue to branch at a11 to produce further sub-branches. So, from root to a1 and from a1 to a11 are both based on the gini index value and the distance criterion to the nearest transaction point to optimize the ability to classify correctly for data groups.

2.8. Step 8: Run and read model results with Random Forest

2.8.1. Run and read model results with Random Forest

Code box 2.21. Run and read forecasts with Random Forest

Table 42 Code box 2.21. Run and read forecasts with Random Forest

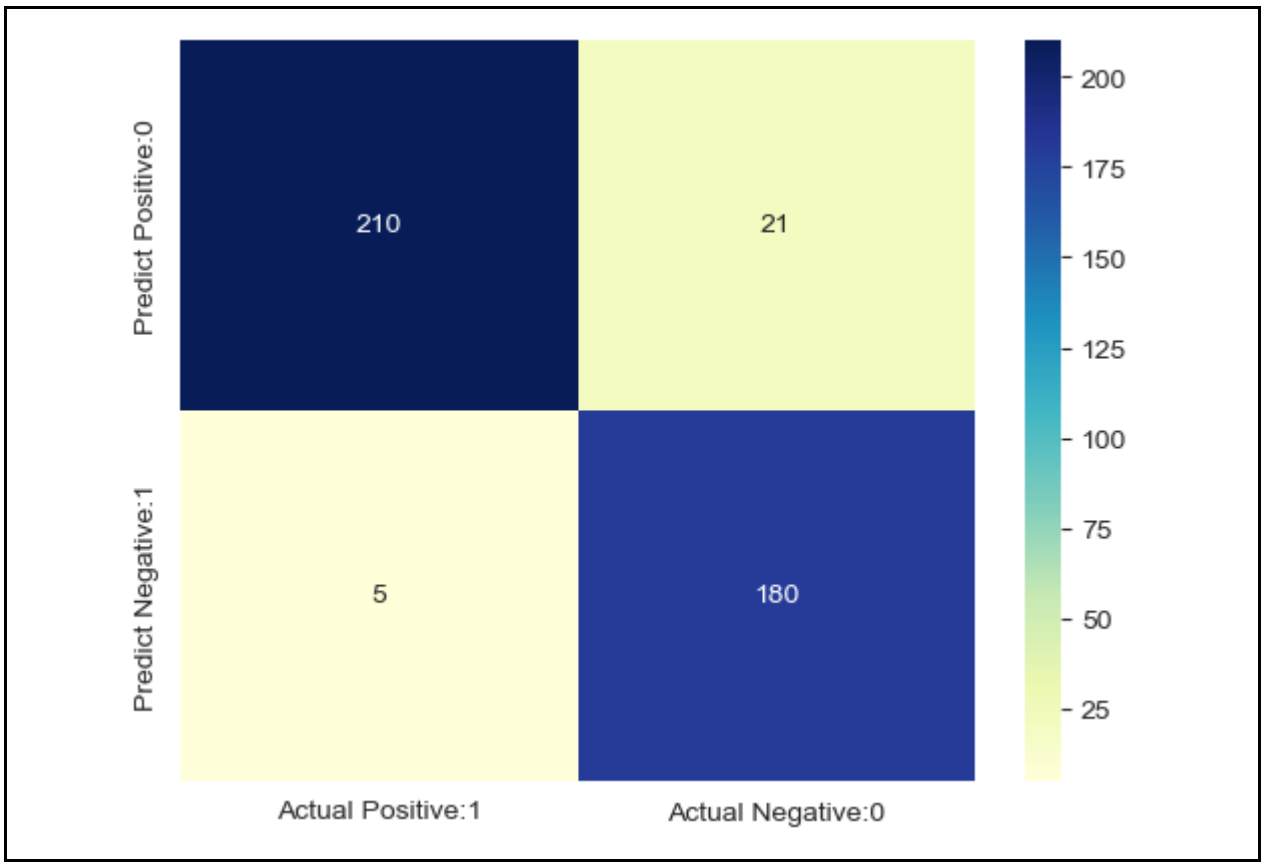
```
RF_classifier = RandomForestClassifier()
RF_classifier.fit(X_train, y_train.ravel())

y_pred = RF_classifier.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print('Random Forest accuracy: ', accuracy_score(y_test, y_pred))

cm = confusion_matrix(y_test, predictions)
……
print("Kết quả dự báo đúng {0:.2f}%".format(100*accuracy_score(predictions, y_test)))

print('\nTrue Positives(TP) = ', cm[0,0])
……
```

```
print("Kết quả dự báo đúng {0:.2f}%".format(100*accuracy_score(predictions, y_test)))
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
……
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual
Negative:0'],index=['Predict Positive:0', 'Predict Negative:1'])
sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Code box result table 2.21: Run and read forecast results with Random Forest

Table 43 Code box result table 2.21: Run and read forecast results with Random Forest

```
Kết quả dự báo đúng 93.27%

True Positives(TP) =  208

True Negatives(TN) =  180

False Positives(FP) =  23

False Negatives(FN) =  5
Kết quả dự báo đúng 93.27%
[[208  23]
 [  5 180]]
              precision    recall  f1-score   support

           0       0.98      0.90      0.94       231
           1       0.89      0.97      0.93       185

    accuracy                           0.93       416
   macro avg       0.93      0.94      0.93       416
weighted avg       0.94      0.93      0.93       416
```



Comment: Compared with the results of the Random Forest model, the results of the decision tree model with balanced data have lower accuracy. The decision tree model with balanced data has an accuracy of 0.913, while your Random Forest model has a higher accuracy of 0.9375. In addition, your Random Forest model has a higher precision and recall of class 1 than the decision tree model with balanced data. In general, the Random Forest model has better results than the decision tree model with balanced data.

2.8.2. Important variables with Random Forest

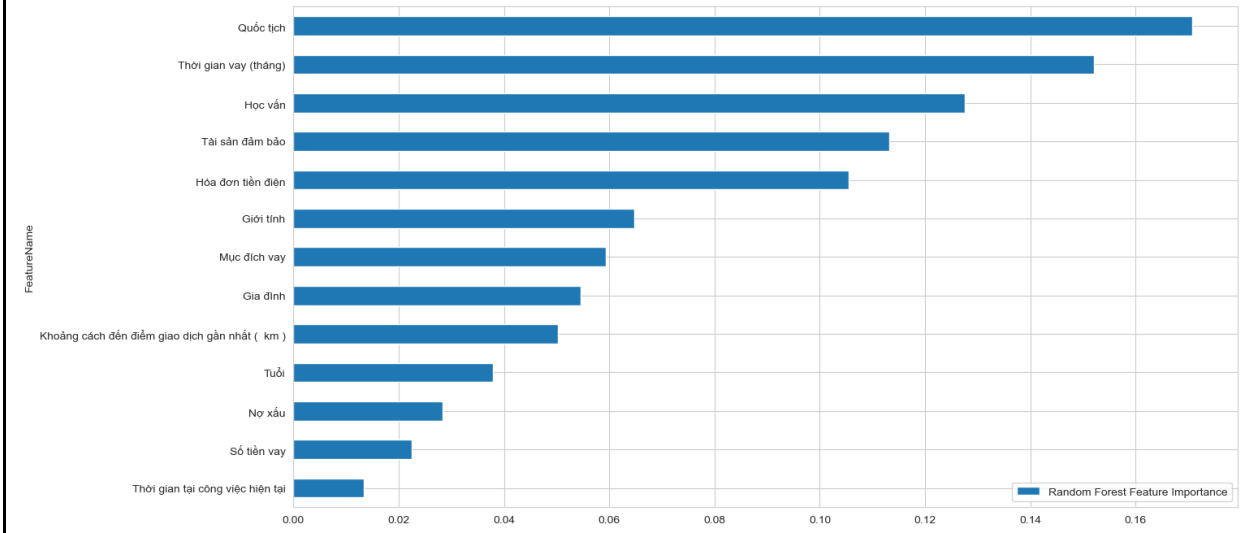
Code box 2.22. Important Variables Random Forest

Table 44 Code box 2.22. Important Variables Random Forest

```
# Tìm kiếm các biến quan trọng
importance_rf = RF_classifier.feature_importances_
features_importances_rf = pd.DataFrame({'FeatureName':
balanced_data.columns[0:len(balanced_data.columns)-1], 'Random Forest Feature Importance':
importance_rf})
features_importances_rf.sort_values(by=['Random Forest Feature Importance'],
ascending=False)
# Vẽ biểu đồ
features_importances_rf.sort_values("Random Forest Feature Importance").plot(figsize=(15,8),
x="FeatureName", y=["Random Forest Feature Importance"], kind="barh")
```

Code box results table 2.22: Important Variables Random Forest

Table 45 Code box results table 2.22: Important Variables Random Forest



Comment: In Decision Tree, the variables Bad debt, education, distance to the nearest transaction point, gender, collateral and time at current job were determined to have the highest influence on forecast results. Meanwhile, in Random Forest, the variables Nationality, Borrowing period, Education, Collateral, Electricity bill and Gender are identified as having the greatest influence on the forecast results. These variables can be used to improve the prediction results of the Random Forest model.

2.8.3. Random Forest's ROC_Curve

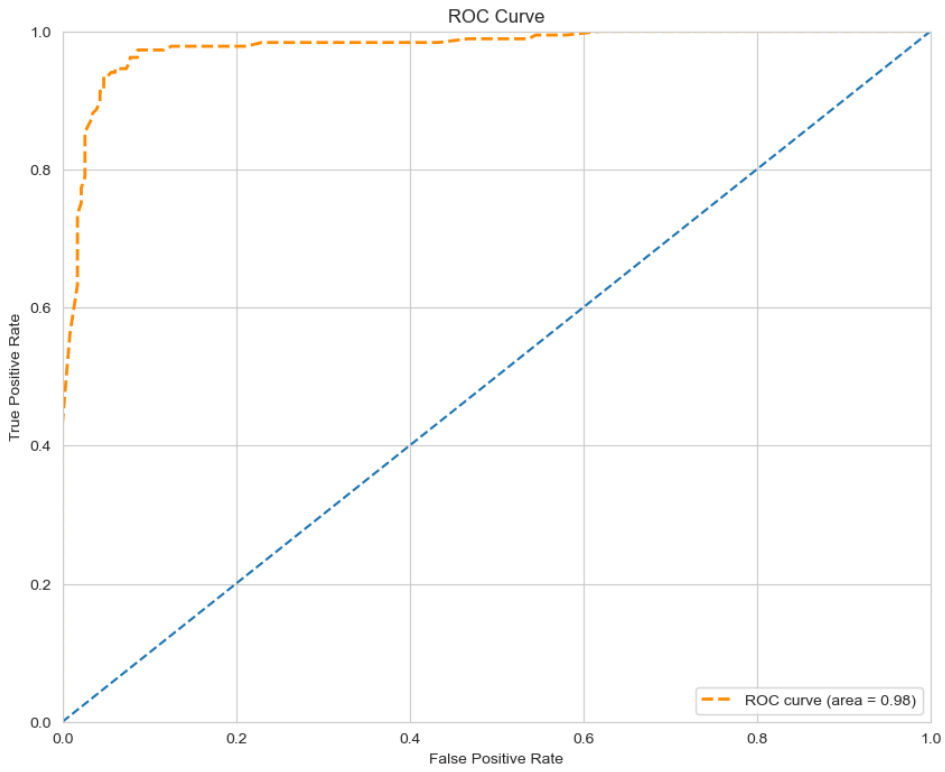
Code box 2.23. Random Forest's ROC_Curve

Table 46 Code box 2.23. Random Forest's ROC_Curve

```
y_pred_prob_test = RF_classifier.predict_proba(X_test)[:, 1]
fpr, tpr, thres = roc_curve(y_test, y_pred_prob_test)
roc_auc = auc(fpr, tpr)

_plot_roc_curve(fpr, tpr, thres, roc_auc)
```

Code box results table 2.23: Code box 2.23. Random Forest's ROC_Curve

Table 47 Code box results table 2.23: Code box 2.23. Random Forest's ROC_Curve



Comment: Comparing the roc_curve results of the Random Forest model and the Decision Tree model, we see that the Random Forest model has a higher area value of 0.98 than the Decision Tree model's value of 0.92. This shows that the Random Forest model has a good ability to classify True Positive and True Negative cases and has higher accuracy than Decision Tree.

2.9. Step 9: New customer forecast with used data set

Code box 2.24. Generate new customer data and forecasts

Table 48 Code box 2.24. Generate new customer data and forecasts

```
# Tạo dữ liệu khách hàng mới
sample_test = [1, 1, 900000, 450000000, 4, 1, 2, 30, 1, 0, 2, 2, 6]
# Thực hiện dự báo
prediction = RF_classifier.predict([sample_test])[0]
# Câu lệnh điều kiện
if prediction == 0:
    print("Khách hàng trả trợ đúng hạn.")
else:
    print("Khách hàng trả nợ không đúng hạn.")
```

Code box results table 2.24: Run and read forecast results with Random Forest

Table 49 Code box results table 2.24: Run and read forecast results with Random Forest

```
Khách hàng trả trợ đúng hạn.
```

Comments: New customer data can be introduced to enable forecasting.

2.10. Step 10: Compare results with different models.

2.10.1. Build models for comparison

Code box 2.25. Build models for comparison

Table 50 Code box 2.25. Build models for comparison

```
# define the classification algorithms
models = [
    ('Logistic Regression', LogisticRegression()),
    ('KNN', KNeighborsClassifier()),
    ('Naive Bayes', GaussianNB()),
    ('SVM', SVC()),
    ('Decision Tree', DecisionTreeClassifier()),
    ('Random Forest', RandomForestClassifier()),
    ('Gradient Boosting', GradientBoostingClassifier()),
    ('XGBoost', XGBClassifier()),
    ('LightGBM', LGBMClassifier()),
    ('CatBoost', CatBoostClassifier(verbose=0)),
    ('ANN', MLPClassifier()),
    ('AdaBoost', AdaBoostClassifier()),
    ('Bagging', BaggingClassifier())
]
results = []
names = []
reports = []
times = []
for name, model in models:
    kfold = KFold(n_splits=10, shuffle=True, random_state=42)
    ……
    print(f"{name} - Mean Accuracy: {cv_results.mean():.3f}, Std Dev:
{cv_results.std():.3f}")
    print(f"{name} - Mean Time: {times[-1]:.3f} seconds")
```

Code box results table 2.25: Build models for comparison

Table 51 Code box results table 2.25: Build models for comparison

```
Logistic Regression - Mean Accuracy: 0.508, Std Dev: 0.013
```

```
Logistic Regression - Mean Time: 0.120 seconds
KNN - Mean Accuracy: 0.645, Std Dev: 0.010
KNN - Mean Time: 0.186 seconds
Naive Bayes - Mean Accuracy: 0.512, Std Dev: 0.019
Naive Bayes - Mean Time: 0.048 seconds
SVM - Mean Accuracy: 0.506, Std Dev: 0.017
SVM - Mean Time: 6.026 seconds
Decision Tree - Mean Accuracy: 0.912, Std Dev: 0.009
Decision Tree - Mean Time: 0.114 seconds
Random Forest - Mean Accuracy: 0.940, Std Dev: 0.008
Random Forest - Mean Time: 3.426 seconds
Gradient Boosting - Mean Accuracy: 0.864, Std Dev: 0.017
Gradient Boosting - Mean Time: 3.450 seconds
XGBoost - Mean Accuracy: 0.935, Std Dev: 0.006
XGBoost - Mean Time: 3.057 seconds
LightGBM - Mean Accuracy: 0.916, Std Dev: 0.012
LightGBM - Mean Time: 1.107 seconds
CatBoost - Mean Accuracy: 0.907, Std Dev: 0.014
CatBoost - Mean Time: 20.863 seconds
ANN - Mean Accuracy: 0.506, Std Dev: 0.020
ANN - Mean Time: 1.762 seconds
AdaBoost - Mean Accuracy: 0.858, Std Dev: 0.016
AdaBoost - Mean Time: 1.463 seconds
Bagging - Mean Accuracy: 0.936, Std Dev: 0.007
Bagging - Mean Time: 1.095 seconds
```

2.10.2. Compare accuracy

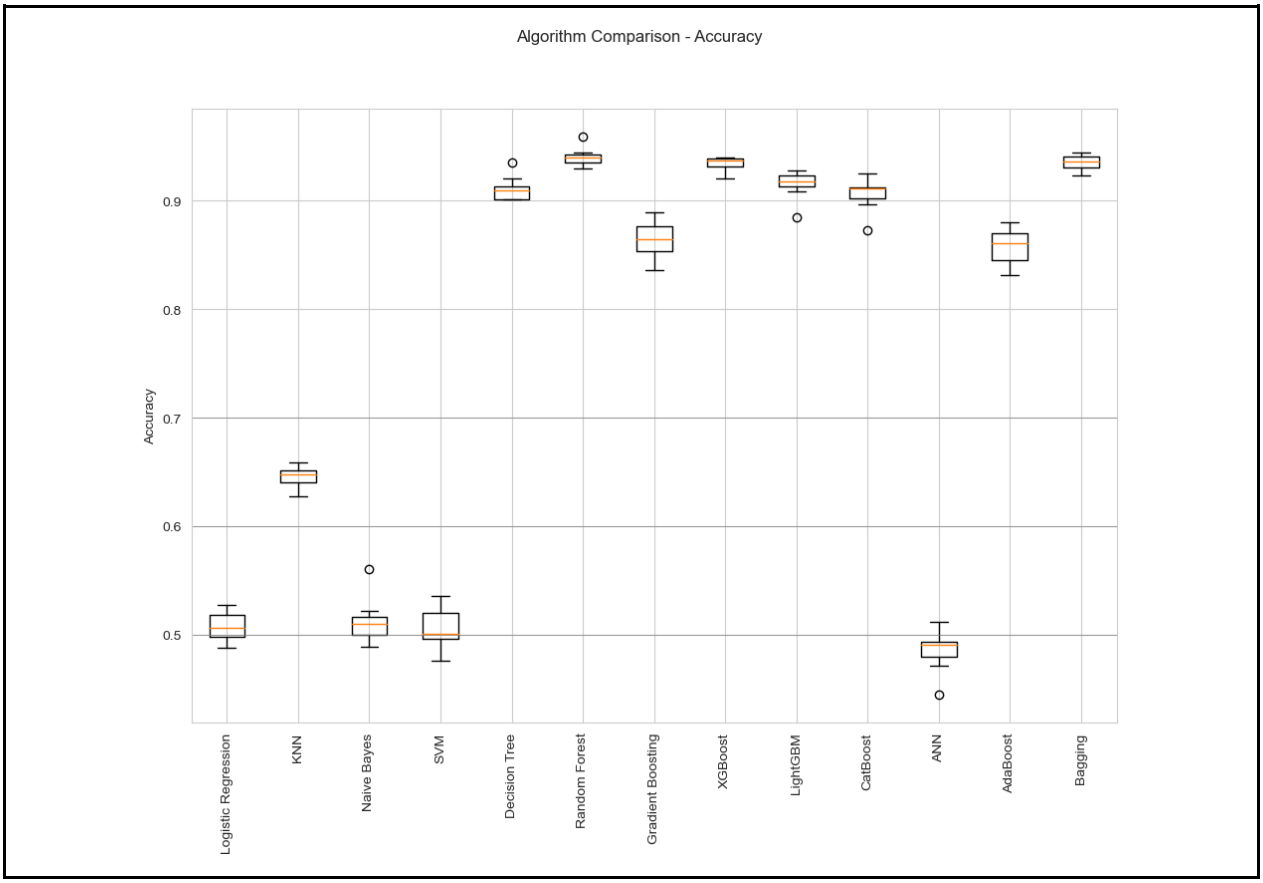
Code box 2.26. Compare accuracy

Table 52 Code box 2.26. Compare accuracy

```python
# plot the performance of each algorithm using a box plot
fig1 = plt.figure(figsize=(12, 8))
fig1.suptitle('Algorithm Comparison - Accuracy')
ax1 = fig1.add_subplot(111)
plt.boxplot(results)
ax1.set_xticklabels(names, rotation=90)
ax1.set_ylabel('Accuracy')

# Add horizontal lines at y-ticks
y_ticks = [0.5, 0.6, 0.7]
for y_tick in y_ticks:
    ax1.axhline(y=y_tick, color='grey', linewidth=0.5)

plt.show()
```

Code box results table 2.26: Compare accuracy

Table 53 Code box results table 2.26: Compare accuracy



Comment: The models used to forecast "Khả năng trả nợ" gave Logictic Regression, KNN, Naive Bayes, SVM and ANN the lowest results. The highest and most stable are the three models Random Forest, XGBoost and Bagging.

2.10.3. Compare F1 - Score

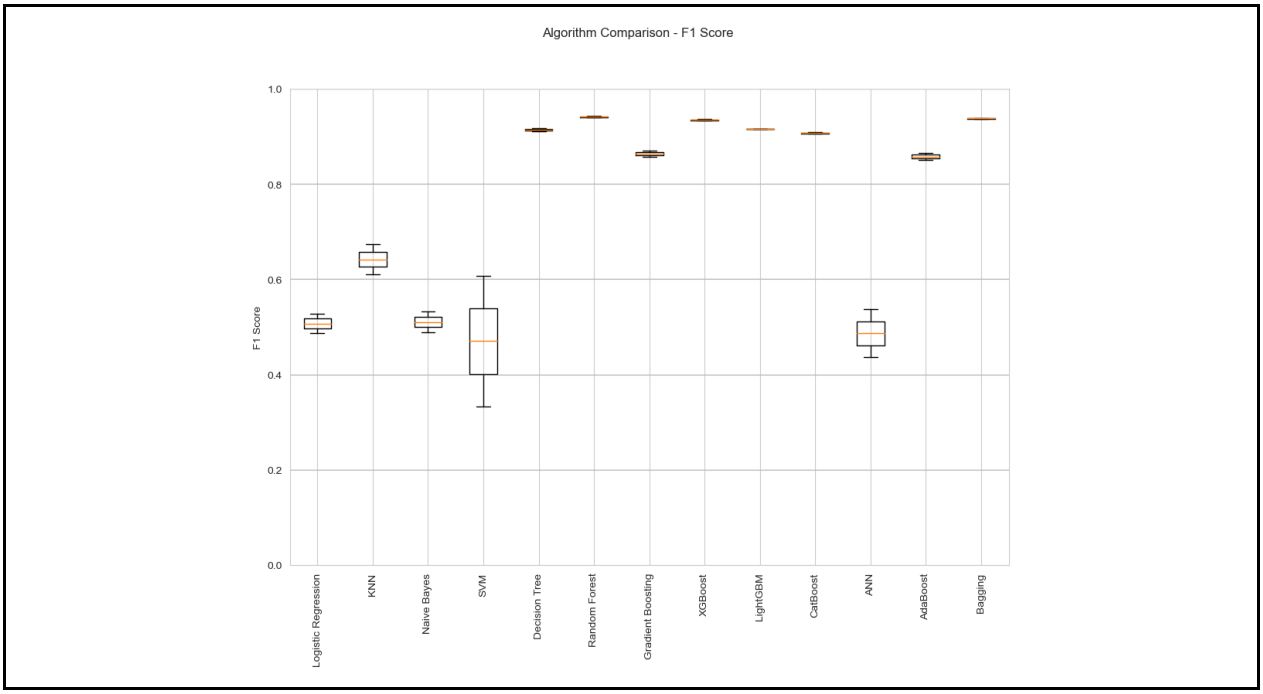
Code box 2.27. Compare F1 - Score

Table 54 Code box 2.27. Compare F1 - Score

```
# compute and plot the f1 score for each algorithm using a box plot
f1_scores = []
for report in reports:
    f1_scores.append([report['0']['f1-score'], report['1']['f1-score']])
.....
# Add horizontal lines at y-ticks 0.2, 0.4, 0.6, 0.8
y_ticks = [0.2, 0.4, 0.6, 0.8]
for y_tick in y_ticks:
    ax3.axhline(y=y_tick, color='grey', linewidth=0.25)

plt.show()
```

Code box results table 2.27: Compare F1 - Score

Table 55 Code box results table 2.27: Compare F1 - Score



Comment: Random Forest, XGBost, and Bagging are still the models with the highest F1-Score in usage models. The highest of the F1-Score results is Random Forest.

2.10.4. Compare precision and recall

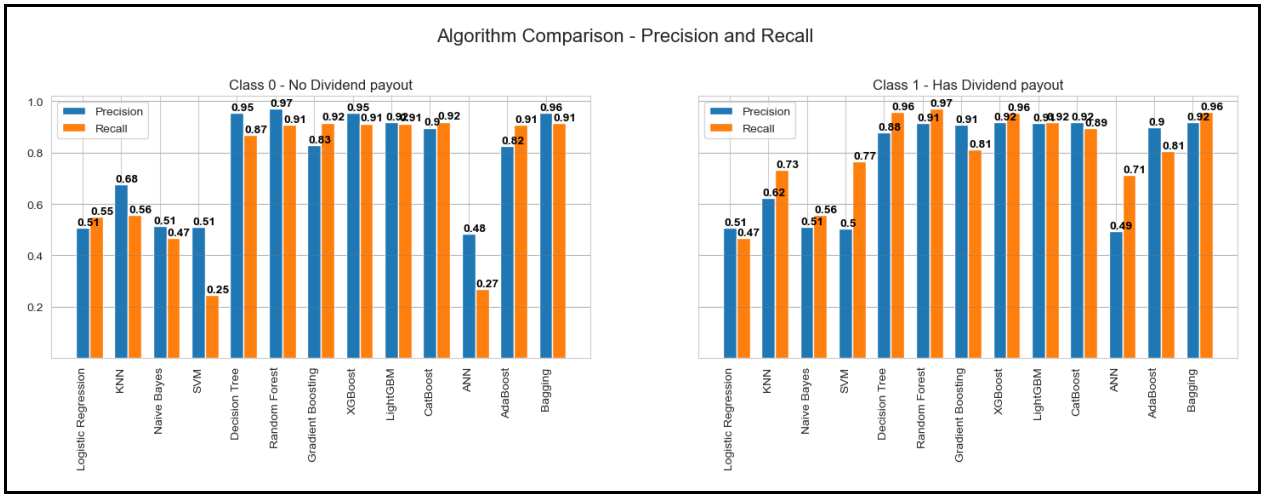
Code box 2.27. Compare F1 - Score

Table 56 Code box 2.27. Compare F1 - Score

```
# compute and plot the precision and recall for each algorithm using a bar
chart
precision_0 = []
recall_0 = []
precision_1 = []
recall_1 = []
for report in reports:
……
# add horizontal lines
for ax in (ax1, ax2):
    for y in [0.2, 0.4, 0.6, 0.8, 1]:
        ax.axhline(y=y, color='gray', alpha=0.5, linewidth=0.5)

plt.show()
```

Code box results table 2.27: Compare F1 - Score

Table 57 Code box results table 2.27: Compare F1 - Score



Comment: Based on the results chart, it can be seen that the highest Recall for class 0 is CatBoost and Gradient Boosting with 0.92, followed by Random Forest with 0.91. The highest precision for class 0 is Random Forest with 0.97 and followed by Bagging with 0.96. For class 1, the results show that Random Forest has the highest result with a Recall of 0.97. Bagging gives the highest Precision with 0.96.

2.10.5. Compare model run time

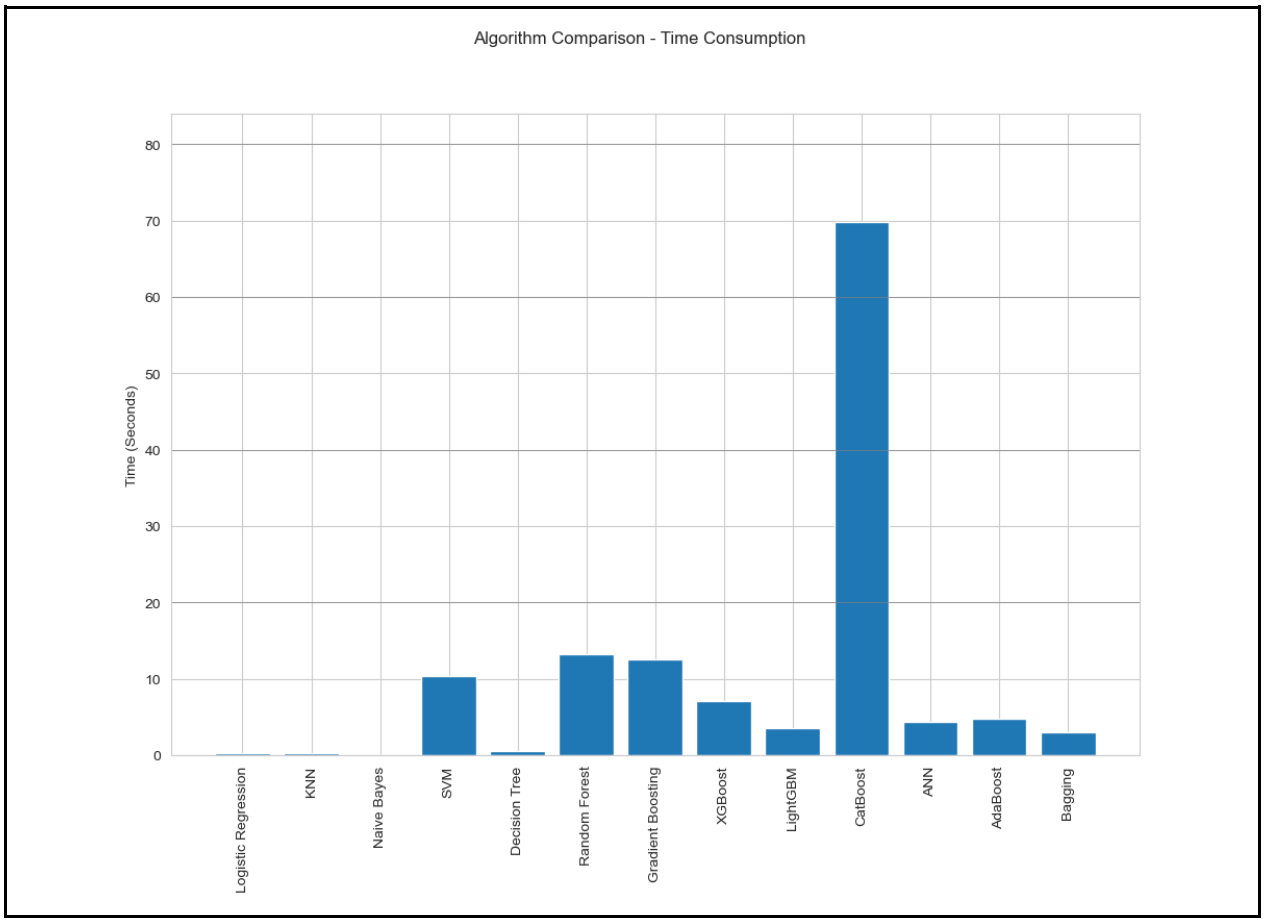
Code box 2.28. Compare model run time

Table 58 Code box 2.28. Compare model run time

```
# plot the time consumption of each algorithm using a bar chart
fig2 = plt.figure(figsize=(12, 8))
fig2.suptitle('Algorithm Comparison - Time Consumption')
ax2 = fig2.add_subplot(111)
ax2.bar(names, times)
ax2.set_xticklabels(names, rotation=90)
ax2.set_ylabel('Time (Seconds)')

# Add horizontal lines at y-ticks
y_ticks = [20, 40, 60, 80]
for y_tick in y_ticks:
    ax2.axhline(y=y_tick, color='grey', linewidth=0.5)

plt.show()
```

Code box results table 2.28: Compare model run time

Table 59 Code box results table 2.28: Compare model run time



Comment: The fastest running times are Logic Regression, KNN and Naïve Bayes. Random Forest and Gradient Boosting are almost equal, with CatBoost having the longest run time.

III. Comments and conclusions

The above report has built a model to predict the repayment ability of customers on time or not. In which, the two main models used in the model are Decision Tree and Random Forest., in addition, many more models are used to evaluate the results in the most satisfactory way.

Based on the above results, it can be seen that Random Forest is the model that gives the best results based on the comparison of Recall, Precesion, F1-Score on prediction classes (0 and 1) . In addition, this model also shows the stability of the model when cross-validations.

Some limitations when doing this is that the data set is not close to reality, there are many variables that do not have a good influence on the model. The future development direction is that it is possible to add input variables affecting the customer's debt repayment ability, can use other forecasting models to be able to compare the efficiency and find the model that is consistent with the article. Furthermore, after developing the complete model, one can apply it in practice to evaluate the model's effectiveness and reliability in a real-world environment by creating web or applications. It can be used to make decisions about whether or not to grant a loan to a specific customer and track the results of the model over time so that improvements can be made.

Github:

https://github.com/MiCasa0403001/credit-risk-model-in-R-Python

Reference documents

Nguyễn Đức Thành, Lê Thanh Bình, Nguyễn Thị Ánh Tuyết, & Trần Đức Minh. (2018). Ứng dụng học máy trong dự báo tín dụng. NXB Giáo dục.