

TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TP. HCM

VIỆN CÔNG NGHỆ THÔNG TIN VÀ ĐIỆN, ĐIỆN TỬ



**BÁO CÁO MÔN HỌC**  
**NGÔN NGỮ LẬP TRÌNH PYTHON**

**TÊN ĐỀ TÀI**  
**DỰ ĐOÁN VỐ NỢ ĐỐI VỚI KHOẢN VAY TIÊU DÙNG**

*Thành phố Hồ Chí Minh, ngày tháng năm 2025*

# Mục Lục

<b>LỜI MỞ ĐẦU.....</b>	<b>i</b>
<b>LỜI CẢM ƠN.....</b>	<b>ii</b>
<b>CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI.....</b>	<b>1</b>
1.1. Lý do chọn đề tài.....	1
1.2. Mục tiêu .....	1
1.3. Tổng quan về dữ liệu .....	1
1.4 . Kỹ thuật sử dụng.....	3
<b>CHƯƠNG 2: XỬ LÝ VÀ TRỰC QUAN HÓA DỮ LIỆU.....</b>	<b>4</b>
2.1. Khám phá dữ liệu ban đầu .....	4
2.2. Tiền xử lý dữ liệu .....	6
2.2.1. Thêm tiêu đề cho cột bị thiếu.....	6
2.2.2. Xử lý giá trị thiếu trong mỗi cột .....	6
2.2.3. Tách cột có nhiều giá trị trong 1 ô .....	8
2.2.4. Xử lý dữ liệu trùng nhau.....	8
2.2.5. Xử lý giá trị không hợp lệ.....	9
2.2.6. Xử lý các cột không đồng nhất dữ liệu .....	9
2.2.7. Xử lý các cột không đồng nhất kiểu dữ liệu .....	11
2.2.8. Xóa khoảng trắng đầu, cuối cho các cột dữ liệu object .....	12
2.2.9. Xử lý các ký tự không hợp lệ.....	12
2.2.10. Loại bỏ các cột không cần thiết .....	13
2.2.11. Xem lại thống kê mô tả .....	13
2.2.12. Xử lý mối quan hệ logic giữa các cột dữ liệu .....	14
2.2.13. Phân loại dữ liệu .....	17
2.2.14. Kiểm tra dữ liệu ngoại lai .....	17
2.3. Lưu dữ liệu vào CSDL.....	19
2.4. Trực quan hóa dữ liệu .....	21
2.4.1. Biểu đồ phân phối tỷ lệ vỡ nợ.....	21
2.4.2. Biểu đồ phân phối thu nhập theo nhóm rủi ro .....	22
2.4.3. Biểu đồ tình trạng sở hữu nhà theo tỷ lệ rủi ro .....	23
2.4.4. Biểu đồ tình trạng sở hữu xe theo tỷ lệ rủi ro .....	24

2.4.5. Biểu đồ phân bố tuổi theo tỷ lệ rủi ro .....	26
2.4.6. Phân bố kinh nghiệm theo nhóm rủi ro.....	28
2.4.7. Biểu đồ thể hiện nghề nghiệp có tỷ lệ vỡ nợ cao nhất .....	30
2.4.8. Biểu đồ thể hiện thành phố có tỷ lệ vỡ nợ cao nhất.....	31
2.4.9. Đề xuất giải pháp .....	32
<b>CHƯƠNG 3: PHÂN TÍCH DỮ LIỆU ĐƯA VÀO MÔ HÌNH MÁY HỌC .....</b>	<b>33</b>
3.1. Chuẩn hóa dữ liệu và mã hóa đặc trưng.....	33
3.2. Sử dụng các phương pháp kiểm định để chọn thuộc tính cần thiết cho mô hình .....	35
3.2.1. Kiểm định Chi Square.....	35
3.2.2. Kiểm định ANOVA F-Test.....	36
3.3. Các đặc trưng quan trọng được chọn từ 2 phương pháp trên.....	37
3.4. Đánh labeling .....	38
3.5 Cân bằng dữ liệu .....	39
3.6. Chia tập train và test .....	40
3.7. Xây dựng mô hình để đưa vào máy học .....	40
3.7.1. Lý do chọn mô hình .....	40
3.7.2. Mô hình Decision Tree.....	40
3.7.3. Mô hình RandomForest .....	41
<b>CHƯƠNG 4: ĐÁNH GIÁ CÁC MÔ HÌNH HỌC MÁY .....</b>	<b>43</b>
4.1. Đánh giá mô hình Decision tree.....	43
4.2. Đánh giá mô hình Random Forests.....	44
4.3. So sánh 2 mô hình.....	45
4.4. Tổng kết .....	46
<b>CHƯƠNG 5: KẾT LUẬN .....</b>	<b>47</b>
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>48</b>

## LỜI MỞ ĐẦU

Với sự phát triển nhanh chóng của nền kinh tế, các sản phẩm tài chính tiêu dùng, đặc biệt là các khoản vay tiêu dùng, ngày càng trở nên phổ biến. Tuy nhiên, trong bối cảnh này, một vấn đề nghiêm trọng mà các tổ chức tín dụng và các cá nhân phải đối mặt là khả năng vỡ nợ của người vay. Dự đoán vỡ nợ đối với khoản vay tiêu dùng không chỉ giúp giảm thiểu rủi ro tài chính mà còn đóng vai trò quan trọng trong việc bảo vệ quyền lợi của các bên liên quan.

Báo cáo này sẽ đi sâu vào việc phân tích và dự đoán khả năng vỡ nợ của các khoản vay tiêu dùng thông qua các phương pháp phân tích dữ liệu và mô hình dự báo. Mục tiêu là cung cấp cái nhìn tổng quan về các yếu tố ảnh hưởng đến khả năng vỡ nợ, từ đó đưa ra các chiến lược giúp các tổ chức tín dụng quản lý rủi ro hiệu quả hơn. Đồng thời, báo cáo cũng sẽ đề cập đến tầm quan trọng của việc áp dụng các công nghệ tiên tiến trong việc phân tích và dự báo các tình huống vỡ nợ trong lĩnh vực tín dụng tiêu dùng.

## LỜI CẢM ƠN

## CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI

### 1.1. Lý do chọn đề tài

Bài toán dự đoán khoản vay dựa trên hành vi khách hàng (Loan Prediction Based on Customer Behavior) là một bài toán được quan tâm rất nhiều bởi các ngân hàng và mọi doanh nghiệp liên quan đến hoạt động giao dịch và cho vay. Hoạt động vay của khách hàng là một trong những nguồn thu lớn, ảnh hưởng trực tiếp đến doanh thu và lợi nhuận của các ngân hàng và doanh nghiệp tài chính.

Thông qua việc phát triển bài toán về dự đoán khả năng vỡ nợ của khoản vay bởi khách hàng, doanh nghiệp có thể có các chiến lược cụ thể để từ đó chuẩn bị, tiếp cận, giám sát, đề phòng các trường hợp không mong muốn có thể xảy ra, từ đó giảm thiểu chi phí cho các chiến dịch dư thừa và không hiệu quả.

Nhận thấy sự cần thiết và tầm quan trọng của vấn đề, nhóm quyết định phát triển và tìm hiểu bài toán nêu trên.

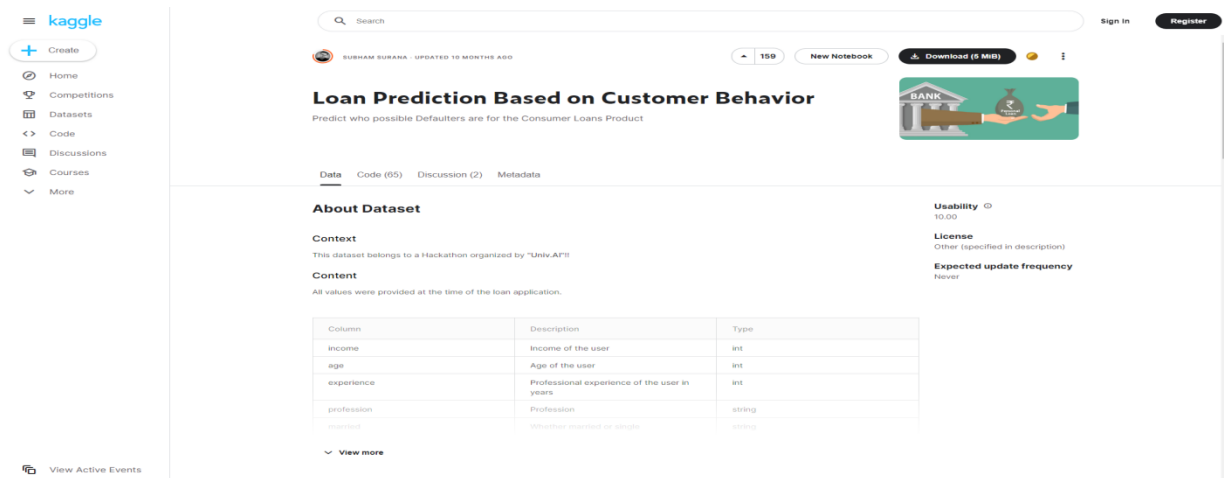
### 1.2. Mục tiêu

- Làm sạch và xử lý dữ liệu bằng ngôn ngữ python
- Trực quan hóa dữ liệu để biểu diễn thông tin dưới dạng hình ảnh, biểu đồ giúp hiểu rõ xu hướng, mẫu hình và mối quan hệ trong dữ liệu, từ đó hỗ trợ đưa ra những chiến lược linh hoạt và hiệu quả.
- Chuẩn hóa các thuộc tính số và mã hóa các thuộc tính phân loại nhằm chuyển đổi dữ liệu về dạng phù hợp với các thuật toán học máy.
- Xây dựng mô hình học máy nhằm dự đoán vỡ nợ đối với khoản vay tiêu dùng.
- Đánh giá hiệu suất mô hình và đề xuất các giải pháp tối ưu hóa

### 1.3. Tổng quan về dữ liệu

Nguồn dữ liệu: [Loan Prediction Based on Customer Behavior | Kaggle](#)

## Ngôn Ngữ Lập Trình Python - Dự Đoán Vỡ Nợ Đối Với Khoản Vay Tiêu Dùng



**Loan Prediction Based on Customer Behavior**  
Predict who possible Defaulters are for the Consumer Loans Product

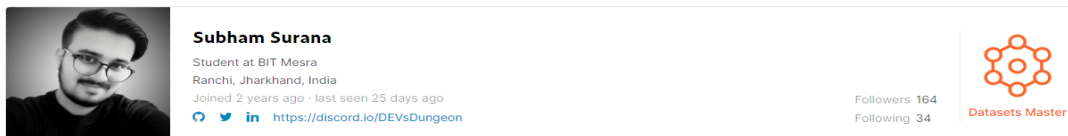
**About Dataset**

**Context**  
This dataset belongs to a Hackathon organized by "Univ.APT"

**Content**  
All values were provided at the time of the loan application.

Column	Description	Type
income	Income of the user	int
age	Age of the user	int
experience	Professional experience of the user in years	int
profession	Profession	string
married	Whether married or single	string

### Thông tin về tác giả



**Subham Surana**  
Student at BIT Mesra  
Ranchi, Jharkhand, India  
Joined 2 years ago · last seen 25 days ago  
<https://discord.io/DEVsDungeon>

Followers 164  
Following 34

**Datasets Master**

- Cập nhật lần cuối vào tháng 8 năm 2021
- Hơn 56 150 lượt xem, 6789 lượt tải về, 159 đánh giá tốt

Mô tả dữ liệu : Dữ liệu có 13 cột và 252 128 dòng

Thuộc tính	Mô tả	Kiểu dữ liệu
Id	Id	int
Income	Thu nhập	int
Age	Tuổi	int
Experience	Kinh nghiệm chuyên môn	int
Married/Single	Tình trạng hôn nhân	string
House_Ownership	Tình trạng sở hữu nhà	string

Car_Ownership	Tình trạng sở hữu xe hơi	string
Profession	Nghề nghiệp	string
CURRENT_JOB_YRS	Kinh nghiệm trong công việc hiện tại	int
CURRENT_HOUSE_YRS	Thời gian cư trú ở nơi hiện tại	int
Risk_Flag	Tình trạng vỡ nợ	String
Extra_Notes	Ghi chú	string
CITY	Thành phố cư trú	string

#### 1.4 . Kỹ thuật sử dụng

Công cụ khai thác dữ liệu: Google Colab.

Phiên bản Python: 3.10.5.

Thư viện: Numpy, Pandas, , Seaborn, sklearn, imblearn, catboost, matplotlib

Mô hình sử dụng: RandomForest , CatBoost



## CHƯƠNG 2: XỬ LÝ VÀ TRỰC QUAN HÓA DỮ LIỆU

### 2.1. Khám phá dữ liệu ban đầu

```
import pandas as pd

# Tải dữ liệu từ tệp CSV với error_bad_lines=False để bỏ qua các dòng lỗi
df = pd.read_csv('Training_Data_5.csv', encoding='ISO-8859-1')

# Hiện thị 5 hàng đầu tiên của dữ liệu
print(df.head())
# Xem thông tin tổng quan
print(df.info())
# KT phân phối chuẩn
print(df.describe())
```

<ipython-input-5-412d6f35ae5a>:4: DtypeWarning: Columns (1) have mixed types. Specify dtype option on import or set low\_memory=False.

```
df = pd.read_csv('Training_Data_5.csv', encoding='ISO-8859-1')
  Id      Income  Age  Experience Unnamed: 4 House_Ownership Car_Ownership \
0  1    1303834.0  23.0         30      single      rented          no
1  2     7574516.0  40.0         10      single      rented          no
2  3     3991815.0  66.0         42    married      rented          no
3  4      6256451.0  41.0          2      single      rented          yes
4  5    576887154.0  47.0         11      single      rented          no
```

```
      Profession      CITY  CURRENT_JOB_YRS \
0  Mechanical_engineer  Rewa,Madhya_Pradesh      3
1  Software_Developer  Parbhani, Maharashtra      9
2  Technical_writer    Alappuzha, Kerala      4
3  Software_Developer  Bhubaneswar, Odisha      2
4  Civil_servant      Tiruchirappalli[10], Tamil_Nadu      3
```

```
      CURRENT_HOUSE_YRS  Risk_Flag  Extra_Notes
0              13      safe      none
1              13      safe      none
2              10      safe      none
3              12    Risky      none
4              14    Risky      none
```

## Ngôn Ngữ Lập Trình Python - Dự Đoán Vỡ Nợ Đối Với Khoản Vay Tiêu Dùng

```

RangeIndex: 252128 entries, 0 to 252127
Data columns (total 13 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Id                     252128 non-null  int64
1   Income                 252128 non-null  object
2   Age                    251584 non-null  float64
3   Experience              252128 non-null  int64
4   Unnamed: 4             252108 non-null  object
5   House_Ownership        246672 non-null  object
6   Car_Ownership          246380 non-null  object
7   Profession              252128 non-null  object
8   CITY                   252128 non-null  object
9   CURRENT_JOB_YRS        252128 non-null  int64
10  CURRENT_HOUSE_YRS      252128 non-null  int64
11  Risk_Flag              252128 non-null  object
12  Extra_Notes            252128 non-null  object
dtypes: float64(1), int64(4), object(8)
memory usage: 25.0+ MB
None

```

	Id	Age	Experience	CURRENT_JOB_YRS	\
count	252128.000000	251584.000000	252128.000000	252128.000000	
mean	125978.656464	49.959087	10.066534	6.334144	
std	72765.984555	17.063144	6.034960	3.647098	
min	1.000000	21.000000	-20.000000	0.000000	
25%	62949.750000	35.000000	5.000000	3.000000	
50%	125996.500000	50.000000	10.000000	6.000000	
75%	188972.250000	65.000000	15.000000	9.000000	
max	252000.000000	79.000000	49.000000	14.000000	

	CURRENT_HOUSE_YRS
count	252128.000000
mean	11.997878
std	1.399049
min	10.000000
25%	11.000000
50%	12.000000
75%	13.000000
max	14.000000

## 2.2. Tiền xử lý dữ liệu

### 2.2.1. Thêm tiêu đề cho cột bị thiếu

```
# Thêm tiêu đề cho cột bị thiếu
df.rename(columns={'Unnamed: 4': 'Married/Single'}, inplace=True)

# Kiểm tra lại tên cột
print(df.columns)

Index(['Id', 'Income', 'Age', 'Experience', 'Married/Single',
       'House_Ownership', 'Car_Ownership', 'Profession', 'CITY',
       'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS', 'Risk_Flag', 'Extra_Notes'],
      dtype='object')
```

### 2.2.2. Xử lý giá trị thiếu trong mỗi cột

- Kiểm tra các giá trị thiếu:

```
#Kiểm tra các giá trị thiếu
print("\nSố lượng giá trị thiếu trong mỗi cột:")
print(df.isna().sum())
```

```
Số lượng giá trị thiếu trong mỗi cột:
Id                0
Income            0
Age              544
Experience        0
Married/Single    20
House_Ownership   5456
Car_Ownership     5748
Profession        0
CITY              0
CURRENT_JOB_YRS   0
CURRENT_HOUSE_YRS 0
Risk_Flag         0
Extra_Notes       0
dtype: int64
```

## Ngôn Ngữ Lập Trình Python - Dự Đoán Vỡ Nợ Đối Với Khoản Vay Tiêu Dùng

- Phương pháp xử lý:

+ Cột Age: Có 544 giá trị thiếu. Vì đây là dữ liệu dạng số liên tục, nên ta thay thế giá trị thiếu bằng trung vị (median) để tránh bị ảnh hưởng bởi các giá trị ngoại lệ (vì trung vị không bị lệch nhiều như giá trị trung bình).

+ Cột Married/Single : Có 20 giá trị thiếu. Cột này mang tính chất phân loại (categorical) (chỉ có 2 giá trị Married hoặc Single). Tuy nhiên, vì số lượng thiếu ít (20 dòng, chiếm 0.00793%) vì thế ta tiến hành xóa những dòng bị thiếu để đảm bảo dữ liệu sau này chính xác hơn.

+ Cột House\_Ownership: Có 5.456 giá trị thiếu: Đây cũng là dữ liệu phân loại (Owned, Rented, hoặc Other). Vì số lượng thiếu nhiều, nếu xóa hết sẽ làm mất rất nhiều dữ liệu.

→ Do đó, thay thế giá trị thiếu bằng mode (giá trị xuất hiện nhiều nhất) để giữ lại càng nhiều dữ liệu gốc càng tốt.

+ Cột Car\_Ownership: Có 5.748 giá trị thiếu: Tương tự như House\_Ownership, vì số lượng thiếu rất lớn nên cũng thay bằng mode để bảo toàn kích thước dữ liệu.

```
# Xử lý các giá trị thiếu trong các cột Age, Married/Single, House_Ownership
df['Age'] = df['Age'].fillna(df['Age'].median())
df.dropna(subset=['Married/Single'], inplace=True)
if df['House_Ownership'].notna().any():
    df['House_Ownership'] = df['House_Ownership'].fillna(df['House_Ownership'].mode()[0])
if df['Car_Ownership'].notna().any(): # Removed the extra space in 'Car_Ownership '
    df['Car_Ownership'] = df['Car_Ownership'].fillna(df['Car_Ownership'].mode()[0]) # Removed the extra space in 'Car_Ownership '

# Kiểm tra lại xem còn thiếu không
print(df.isnull().sum())
```

```
· Id                0
  Income            0
  Age               0
  Experience         0
  Married/Single     0
  House_Ownership    0
  Car_Ownership      0
  Profession         0
  CITY              0
  CURRENT_JOB_YRS    0
  CURRENT_HOUSE_YRS  0
  Risk_Flag         0
  Extra_Notes        0
  dtype: int64
```

### 2.2.3. Tách cột có nhiều giá trị trong 1 ô

- Phương pháp xử lý: Tiến hành tách cột 'CITY' ban đầu thành hai cột mới là 'City' và 'STATE' bằng cách sử dụng dấu chấm (,) làm ký tự phân tách. Sau khi tạo ra hai cột mới và gán giá trị tương ứng cho chúng, cột 'CITY' gốc sẽ được xóa khỏi bảng dữ liệu để tránh trùng lặp và đảm bảo dữ liệu được chuẩn hóa.

```
# Tách cột CITY thành City và State
df[['City', 'STATE']] = df['CITY'].str.split(',', n=1, expand=True)

if 'STATE' not in df.columns:
    df['STATE'] = None

# Bỏ cột 'CITY' ban đầu
df.drop('CITY', axis=1, inplace=True) # Thêm dòng này để bỏ cột 'CITY'

# Kiểm tra lại
print(df.head())
```

	Id	Income	Age	Experience	Married/Single	House_Ownership	\
0	1	1303834.0	23.0	30	single	rented	
1	2	7574516.0	40.0	10	single	rented	
2	3	3991815.0	66.0	42	married	rented	
3	4	6256451.0	41.0	2	single	rented	
4	5	576887154.0	47.0	11	single	rented	

	Car_Ownership	Profession	CURRENT_JOB_YRS	CURRENT_HOUSE_YRS	\
0	no	Mechanical_engineer	3	13	
1	no	Software_Developer	9	13	
2	no	Technical_writer	4	10	
3	yes	Software_Developer	2	12	
4	no	Civil_servant	3	14	

	Risk_Flag	Extra_Notes	City	STATE
0	safe	none	Rewa	Madhya_Pradesh
1	safe	none	Parbhani	Maharashtra
2	safe	none	Alappuzha	Kerala
3	Risky	none	Bhubaneswar	Odisha
4	Risky	none	Tiruchirappalli[10]	Tamil_Nadu

### 2.2.4. Xử lý dữ liệu trùng nhau

- Kiểm tra các dòng trùng nhau

```
# Kiểm tra có bao nhiêu dòng trùng nhau
print("Số dòng trùng nhau:", df.duplicated().sum())
```

Số dòng trùng nhau: 158

- Phương pháp xử lý: Xóa các dòng trùng nhau, giữ lại dòng đầu tiên

```
# Xóa dòng trùng nhau, giữ lại dòng đầu tiên
df.drop_duplicates(inplace=True)
```

### 2.2.5. Xử lý giá trị không hợp lệ

- In ra các dòng trong cột Experience < 0:

```
# In ra chỉ số dòng experiece < 0
num_invalid_exp = (df['Experience'] < 0).sum()
print(f"Các dòng có Experience < 0: {num_invalid_exp}")
```

Các dòng có Experience < 0: 236

- Phương pháp xử lý: Tính trung vị để thay thế cho các giá trị không hợp lệ

```
# Tính trung vị các giá trị Experience hợp lệ (>= 0)
median_exp = df.loc[df['Experience'] >= 0, 'Experience'].median()

# Thay các giá trị âm bằng trung vị
df.loc[df['Experience'] < 0, 'Experience'] = median_exp

# Kiểm tra lại cột experience
count = (df['Experience'] < 0).sum()
print(f"Số dòng có Experience < 0 là: {count}")
```

Số dòng có Experience < 0 là: 0

### 2.2.6. Xử lý các cột không đồng nhất dữ liệu

- Kiểm tra các cột có kiểu dữ liệu object không đồng nhất dữ liệu:

```
#Kiểm tra các cột không đồng nhất dữ liệu
data_cols = df.columns
for col in data_cols:
    # Check if the column is of object type before applying lower()
    if df[col].dtype == 'object':
        unique_vals = df[col].dropna().unique()
        lower_vals = [val.lower() if isinstance(val, str) else val for val in unique_vals]
        if len(set(lower_vals)) < len(set(unique_vals)):
            print(f"Cột '{col}' Dữ liệu không đồng nhất")
            print(unique_vals)
            print("-" * 40)
```

- Kết quả: Các cột Married/Single, House\_Ownerhsip, Risk\_Flag không đồng nhất dữ liệu, có nhiều giá trị lẫn lộn

## Ngôn Ngữ Lập Trình Python - Dự Đoán Vỡ Nợ Đối Với Khoản Vay Tiêu Dùng

```
Cột 'Married/Single' Dữ liệu không đồng nhất
['single' 'married' ' ' single' 'Single' 'Married'
 ' ' single' ' ' single' 'singl']
-----
Cột 'House_Ownership' Dữ liệu không đồng nhất
['rented' 'norent_noown' ' ' rented' 'owned' 'rEntED' 'renTEd'
 'renTEd' 'reNtEd' 'REnted' 'Rented' 'rEnted' 'renteD' 'rentED']
-----
Cột 'Risk_Flag' Dữ liệu không đồng nhất
['safe' 'Risky' ' ' safe' ' ' safe' ' Risky'
 ' Risky' ' ' safe' 'riskY' ' ' safe' ' safe'
 'ris ky' ' ' safe' ' ris ky' 'ri sky' 'r isky'
 'risky' 's Afe' 'SAfe' 'SAF e' 'saf e']
```

- Phương pháp xử lí: Chuẩn hóa chuỗi về chữ thường, bỏ khoảng trắng đầu cuối, xóa khoảng trắng giữa các chữ, sau đó áp dụng chuẩn hóa cho các cột Married/Single, House\_Ownerhsip, Risk\_Flag

```
import re

# Hàm chuẩn hóa chuỗi: về chữ thường, bỏ khoảng trắng đầu cuối, xóa khoảng trắng giữa các chữ
def normalize_string(s):
    return re.sub(r'\s+', '', str(s).strip().lower())

# Áp dụng chuẩn hóa
df['Married/Single'] = df['Married/Single'].apply(normalize_string)
df['House_Ownership'] = df['House_Ownership'].apply(normalize_string)
df['Risk_Flag'] = df['Risk_Flag'].apply(normalize_string)

# Map lại về giá trị đúng (nếu có các sai lệch nhẹ như "singl" hoặc "noownnorent")
df['Married/Single'] = df['Married/Single'].replace({
    'single': 'single',
    'singl': 'single',
    'married': 'married'
})
```

```
df['House_Ownership'] = df['House_Ownership'].replace({
    'rented': 'rented',
    'norent_noown': 'norent_noown',
    'owned': 'owned',
})

df['Risk_Flag'] = df['Risk_Flag'].replace({
    'risky': 'risky',
    'safe': 'safe'
})

# Giữ lại các hàng hợp lệ
df = df[
    df['Married/Single'].isin(['single', 'married']) &
    df['House_Ownership'].isin(['rented', 'norent_noown', 'owned']) &
    df['Risk_Flag'].isin(['risky', 'safe'])
]

# Kiểm tra kết quả
print("Giá trị cột 'Married/Single' sau chuẩn hóa:", df['Married/Single'].unique())
print("Giá trị cột 'House_Ownership' sau chuẩn hóa:", df['House_Ownership'].unique())
print("Giá trị cột 'Risk_Flag' sau chuẩn hóa:", df['Risk_Flag'].unique())
```

```
Giá trị cột 'Married/Single' sau chuẩn hóa: ['single' 'married']
Giá trị cột 'House_Ownership' sau chuẩn hóa: ['rented' 'norent_noown' 'owned']
Giá trị cột 'Risk_Flag' sau chuẩn hóa: ['safe' 'risky']
```

### 2.2.7. Xử lý các cột không đồng nhất kiểu dữ liệu

- Phương pháp xử lý:

+ Cột Income:

- Do trong cột Income có chứa nhiều kiểu dữ liệu khác nhau (số và chuỗi), nên trước tiên tiến hành chuyển các giá trị không hợp lệ (dạng chuỗi) thành NaN bằng cách dùng `pd.to_numeric(errors='coerce')`.
- Sau đó, thay thế các giá trị NaN bằng trung vị (median) của cột Income để đảm bảo tính ổn định dữ liệu.
- Cuối cùng, chuyển toàn bộ cột Income về kiểu dữ liệu int để đồng nhất và dễ xử lý.

+ Chuyển cả hai cột Age và Experience sang kiểu dữ liệu int nhằm đảm bảo dữ liệu đúng dạng số nguyên, phù hợp với bản chất thực tế của các thông tin này.

```
# Chuyển các giá trị ko hợp lệ trong Income thành nan
df['Income'] = pd.to_numeric(df['Income'], errors='coerce')
# Thay các giá trị(NaN)thành giá trị median, sau đó chuyển sang kiểu int
df['Income'] = df['Income'].fillna(df['Income'].median()).astype(int)
df['Age'] = df['Age'].astype(int)
df['Experience'] = df['Experience'].astype(int)
# Kiểm tra lại kiểu dữ liệu
print(df.dtypes)
```



Id	int64
Income	int64
Age	int64
Experience	int64
Married/Single	object
House_Ownership	object
Car_Ownership	object
Profession	object
CURRENT_JOB_YRS	int64
CURRENT_HOUSE_YRS	int64
Risk_Flag	object
Extra_Notes	object
City	object
STATE	object
dtype:	object

---

## 2.2.8. Xóa khoảng trắng đầu, cuối cho các cột dữ liệu object

```
# Xóa khoảng trắng ở đầu và cuối cho tất cả các cột object
for col in df.select_dtypes(include='object').columns:
    df[col] = df[col].str.strip()
```

## 2.2.9. Xử lý các ký tự không hợp lệ

- Phương pháp xử lý: Trong cột City xuất hiện các ký tự không hợp lệ, ví dụ như "Rewa[10]", "Delhi[5]", v.v.. Tiến hành làm sạch dữ liệu bằng cách xóa toàn bộ các ký tự không hợp lệ bằng phương thức str.replace() với biểu thức chính quy (regex)

```
# Xử lý các ký tự không hợp lệ trong cột City
df['City'] = df['City'].str.replace(r'\[.*?\]', '', regex=True)

# In ra 10 dòng đầu tiên của cột City sau khi xử lý
print("10 dòng đầu tiên của cột City sau khi làm sạch:")
print(df['City'].head(10))
```

10 dòng đầu tiên của cột City sau khi làm sạch:

```
0          Rewa
1      Parbhani
2      Alappuzha
3      Bhubaneswar
4  Tiruchirappalli
5          Jalgaon
6      Tiruppur
7      Jamnagar
8          Kota
9      Karimnagar
Name: City, dtype: object
```

### 2.2.10. Loại bỏ các cột không cần thiết

Các cột như 'Extra\_Notes' và 'Id' không mang giá trị phân tích:

- 'Id' chỉ là mã định danh duy nhất cho từng dòng dữ liệu, không có ý nghĩa thống kê hay ảnh hưởng đến mô hình phân tích.
- 'Extra\_Notes' chứa thông tin không liên quan hoặc không có giá trị sử dụng trong phân tích dữ liệu hiện tại.

```
# Xóa cột 'Extra_Notes' và 'Id'
cols_to_drop = ['Extra_Notes', 'Id']
df = df.drop(columns=[col for col in cols_to_drop if col in df.columns])

# Kiểm tra lại danh sách các cột
print("Danh sách các cột còn lại:", df.columns.tolist())
```

Danh sách các cột còn lại: ['Income', 'Age', 'Experience', 'Married/Single', 'House\_Ownership', 'Car\_Ownership', 'Profession', 'CURRENT\_JOB\_YRS', 'CURRENT\_HOUSE\_YRS', 'Risk\_Flag', 'City', 'STATE']

### 2.2.11. Xem lại thống kê mô tả

```
# Xem mô tả thống kê
print(df.describe())
```

	Income	Age	Experience	CURRENT_JOB_YRS	\
count	2.519500e+05	251950.000000	251950.000000	251950.000000	
mean	6.586715e+08	49.960564	10.085350	6.333963	
std	5.217090e+10	17.045290	6.002009	3.647106	
min	1.031000e+04	21.000000	0.000000	0.000000	
25%	2.570783e+06	35.000000	5.000000	3.000000	
50%	5.166669e+06	50.000000	10.000000	6.000000	
75%	7.715562e+06	65.000000	15.000000	9.000000	
max	5.566620e+12	79.000000	49.000000	14.000000	

	CURRENT_HOUSE_YRS
count	251950.000000
mean	11.997837
std	1.399041
min	10.000000
25%	11.000000
50%	12.000000
75%	13.000000
max	14.000000

## 2.2.12. Xử lý mối quan hệ logic giữa các cột dữ liệu

### - Kiểm tra và xử lý lỗi giữa cột Experience và Age:

+ Lọc ra các dòng Experience >= Age.

+ Xóa toàn bộ các dòng dữ liệu có Experience >= Age ra khỏi bộ dữ liệu. Vì lỗi chỉ có 9 dòng, nó chỉ chiếm khoảng 0.0035% trên tổng số 252,128 dòng, nên việc loại bỏ những dòng này không gây ảnh hưởng đáng kể đến chất lượng và độ đại diện của tập dữ liệu.

+ Sau khi xóa, kiểm tra lại số dòng còn lại để đảm bảo rằng không còn dòng lỗi nào.

```
# Kiểm tra logic giữa các cột
# 1. Experience >= Age
invalid_experience_age = df[df['Experience'] >= df['Age']]
print("\nSố dòng có lỗi Experience >= Age:", len(invalid_experience_age))
print(invalid_experience_age[['Age', 'Experience']].head())

# Xóa các dòng có Experience >= Age
df = df[df['Experience'] < df['Age']]
print("Số dòng còn lỗi Experience >= Age:", len(df[df['Experience'] >= df['Age']]))
```

```
Số dòng có lỗi Experience >= Age: 9
   Age  Experience
0    23         30
234   22         25
373   46         49
422   25         30
460   28         30
Số dòng còn lỗi Experience >= Age: 0
```

#### - Kiểm tra điều kiện Age - Experience < 18 và xử lý:

+ Lọc ra các dòng có điều kiện Age - Experience < 18

+ Tiến hành điều chỉnh giá trị Experience nhằm đảm bảo logic hợp lý. Cụ thể, cập nhật lại Experience = Age - 18 cho những dòng vi phạm. Vì

- Số dòng lỗi nhiều (32,800 dòng), ảnh hưởng độ đại diện và giảm độ tin cậy khi phân tích hoặc huấn luyện mô hình.
- Không thể gán bằng 0 hoặc null vì cột CURRENT\_JOB\_YRS đang > 0, nên nếu làm vậy sẽ phá vỡ tính logic giữa số năm làm việc hiện tại và tổng kinh nghiệm.
- Không thể thay thế bằng trung vị của cột Experience vì trung vị cột (=10) nên khi thay thế thì vẫn sai logic giữa cột Age và Experience ( Age-Experience <18)
- Việc điều chỉnh giúp giữ lại dữ liệu và đảm bảo tính hợp lý, vì người đi làm thường bắt đầu từ 18 tuổi.
- Khó xây dựng mô hình dự đoán Experience (hồi quy dựa trên Age, Education, Job Role...)=> Phức tạp, tốn công xây dựng mô hình riêng và kiểm định độ chính xác

+ Sau khi điều chỉnh, kiểm tra lại số dòng còn lại để đảm bảo rằng không còn dòng lỗi nào.

```
# 2. Age - Experience < 18
invalid_age_experience = df[df['Age'] - df['Experience'] < 18]
print("\nSố dòng có lỗi Age - Experience < 18:", len(invalid_age_experience))
print(invalid_age_experience[['Age', 'Experience']].head())

# Điều chỉnh Experience sao cho Age - Experience >= 18
df.loc[df["Age"] - df["Experience"] < 18, "Experience"] = df["Age"] - 18
print("Số dòng còn lỗi Age - Experience < 18 sau điều chỉnh:", len(df[df["Age"] - df["Experience"] < 18]))
```

```
Số dòng có lỗi Age - Experience < 18: 32080
Age  Experience
8    24         17
9    23         12
24   23         15
45   25         16
65   27         12
Số dòng còn lỗi Age - Experience < 18 sau điều chỉnh: 0
```

### - Kiểm tra và xử lý lỗi giữa CURRENT\_JOB\_YRS và Experience

+ Lọc ra các dòng CURRENT\_JOB\_YRS > Experience

+ Tiến hành xử lý gán CURRENT\_JOB\_YRS = Experience. Vì

- Không thể xóa các dòng lỗi vì số lượng lên tới 15 965 dòng (~6.33% tổng dữ liệu).  
→ Nếu xóa, sẽ làm mất dữ liệu nhiều, ảnh hưởng độ đại diện và độ tin cậy của tập dữ liệu.
- Không thể gán bằng 0 hoặc Null cho Experience, vì:

+ Những dòng đó có CURRENT\_JOB\_YRS > 0, tức là họ đang có kinh nghiệm làm việc thực tế.

+ Nếu gán Experience = 0 hoặc null, thì sẽ mâu thuẫn: làm việc nhiều năm nhưng tổng kinh nghiệm lại bằng 0 → dữ liệu bị phi lý.

- Không thể thay thế bằng trung vị của cột Experience vì trung vị cột (=10) nên khi thay thế thì vẫn sai logic giữa cột Experience và CURRENT\_JOB\_YRS (CURRENT\_JOB\_YRS > Experience)
- .....

+ Sau khi điều chỉnh, kiểm tra lại số dòng còn lại để đảm bảo rằng không còn dòng lỗi nào.

## Ngôn Ngữ Lập Trình Python - Dự Đoán Vỡ Nợ Đối Với Khoản Vay Tiêu Dùng

```
# 3. # CURRENT_JOB_YRS > Experience
invalid_job_yrs = df[df['CURRENT_JOB_YRS'] > df['Experience']]
print("\nSố dòng có CURRENT_JOB_YRS > Experience:", len(invalid_job_yrs))
print(invalid_job_yrs[['CURRENT_JOB_YRS', 'Experience']].head())

# Xử lý
df.loc[df[df['CURRENT_JOB_YRS'] > df['Experience'], 'CURRENT_JOB_YRS'] = df['Experience']
print("Số dòng còn lỗi CURRENT_JOB_YRS > Experience sau điều chỉnh:", len(df[df['CURRENT_JOB_YRS'] > df['Experience']]))
```

```
Số dòng có CURRENT_JOB_YRS > Experience: 15965
  CURRENT_JOB_YRS  Experience
8              11           6
45             13           7
65             12           9
88             11           6
90             10           9
Số dòng còn lỗi CURRENT_JOB_YRS > Experience sau điều chỉnh: 0
```

### 2.2.13. Phân loại dữ liệu

```
# Phân loại dữ liệu
numeric_cols = ['Income', 'Age', 'Experience', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS']
categorical_features = ['Profession', 'City', 'STATE', 'Married/Single', 'Car_Ownership', 'House_Ownership']
```

### 2.2.14. Kiểm tra dữ liệu ngoại lai

#### Lý do chọn phương pháp IQR:

Phân phối dữ liệu:

-Income: Mean ( $6.59e8$ ) > Median ( $5.17e6$ ), khoảng từ median đến max ( $5.57e12 - 5.17e6 \approx 5.57e12$ ) cực lớn so với từ min đến median ( $5.17e6 - 1.03e4 \approx 5.16e6$ ) => Dữ liệu lệch phải

- Age: Mean (49.96)  $\approx$  Median (50): Gần đối xứng

- Experience: Mean (10,08)  $\approx$  Median (10), khoảng từ median đến max ( $49 - 10 = 39$ ) lớn hơn từ min đến median ( $10 - 0 = 0$ ). => Lệch phải....

=> Vì những thuộc tính có phân phối hỗn hợp nên dùng phương pháp IQR để kiểm tra dữ liệu ngoại lai thay vì Z-score

#### Cơ sở lý thuyết

- IQR (Interquartile Range) là khoảng tứ phân vị, được tính bằng hiệu số giữa phân vị 75% (Q3) và phân vị 25% (Q1) của tập dữ liệu:

+ Phù hợp với dữ liệu có phân phối hỗn hợp

- + Tập trung vào vùng dữ liệu trung tâm, không bị ảnh hưởng bởi giá trị cực đại
- + Phù hợp với tập dữ liệu lớn
- + Tính toán nhanh mà không cần mô hình phức tạp
- + Xác định các giá trị bất thường trong một cột mà không cần xem xét mối quan hệ giữa các cột

```
# Hàm kiểm tra ngoại lai bằng IQR
def detect_outliers_iqr(df, numeric_cols):
    Q1 = df[numeric_cols].quantile(0.25)
    Q3 = df[numeric_cols].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    outliers = df[(df[numeric_cols] < lower_bound) | (df[numeric_cols] > upper_bound)]
    outlier_ratio = len(outliers)

    return outlier_ratio
# Kiểm tra tỷ lệ ngoại lai
print("Tỷ lệ ngoại lai:")
for col in numeric_cols:
    num_outliers = detect_outliers_iqr(df, col)
    print(f"- {col}: {num_outliers} ngoại lai")
```

Sau khi áp dụng phương pháp IQR (Interquartile Range), kết quả phát hiện dữ liệu ngoại lai như sau:

- Cột Income: Có 7.790 giá trị ngoại lai, chiếm khoảng 3,09% tổng dữ liệu. Dữ liệu này có phân phối lệch phải rõ rệt, do đó việc loại bỏ các giá trị cực lớn giúp giảm ảnh hưởng đến mô hình học máy.
- Cột Experience: Có 3 giá trị ngoại lai, không đáng kể so với tổng thể. Phân phối cũng có xu hướng lệch phải nhẹ.

```
· Tỷ lệ ngoại lai:
  - Income: 7790 ngoại lai
  - Age: 0 ngoại lai
  - Experience: 3 ngoại lai
  - CURRENT_JOB_YRS: 0 ngoại lai
  - CURRENT_HOUSE_YRS: 0 ngoại lai
```

## Xử lý ngoại lai

Phương pháp xử lý: Các ngoại lệ được thay thế bằng trung vị. Vì :

- Trung vị (median) là một giá trị mạnh mẽ (robust) đối với phân phối lệch, không bị ảnh hưởng bởi các giá trị cực đại như trung bình (mean).
- Cột Income có phân phối lệch phải mạnh (mean = 6.59e8, median = 5.17e6), nên trung vị đại diện tốt hơn cho giá trị trung tâm của dữ liệu.
- Thay thế ngoại lai bằng trung vị giúp giữ dữ liệu trong phạm vi hợp lý mà không làm thay đổi đáng kể phân phối tổng thể.
- Phương pháp này đơn giản, dễ triển khai, và phù hợp khi số lượng ngoại lai không quá lớn (3.09%).

```
# Xử lý ngoại lai
def replace_outliers_iqr(df, numeric_cols, method='median'):
    Q1 = df[numeric_cols].quantile(0.25) # Calculate Q1 for the column
    Q3 = df[numeric_cols].quantile(0.75) # Calculate Q3 for the column
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    if method == 'median':
        median_value = df[numeric_cols].median()
        df.loc[(df[numeric_cols] < lower_bound) | (df[numeric_cols] > upper_bound), numeric_cols] = median_value
        print(f"Cột {numeric_cols}: Các ngoại lai đã được thay thế bằng trung vị ({median_value}).")
    return df

# Thay thế ngoại lai
for col in ['Income', 'Experience']:
    df = replace_outliers_iqr(df, col, method='median')
    num_outliers = detect_outliers_iqr(df, col)
    print(f"Số ngoại lai còn lại ở cột {col}: {num_outliers}")
```

Cột Income: Các ngoại lai đã được thay thế bằng trung vị (5166669.0).  
Số ngoại lai còn lại ở cột Income: 0  
Cột Experience: Các ngoại lai đã được thay thế bằng trung vị (9.0).  
Số ngoại lai còn lại ở cột Experience: 0

## 2.3. Lưu dữ liệu vào CSDL

### - Xuất file đã làm sạch

```
# Xuất file csv
print ("Xuất file csv")
df.to_csv('Cleaned_Data.csv', index=False)
```



## Ngôn Ngữ Lập Trình Python - Dự Đoán Vỡ Nợ Đối Với Khoản Vay Tiêu Dùng

### - Tiến hành đưa file lên python để lưu vào SQL

```
# Cài đặt thư viện
import pandas as pd
from sqlalchemy import create_engine
# Đọc file CSV
df = pd.read_csv('Cleaned_Data.csv', encoding='utf-8')

# Kết nối MySQL
username = 'root'
password = '123456'
host = 'localhost'
database = 'data_cleaned'

engine = create_engine(f"mysql+mysqlconnector://{username}:{password}@{host}/{database}")

# Ghi dữ liệu vào bảng
df.to_sql(name='ecommerce_data_cleaned', con=engine, if_exists='replace', index=False)
```

MySQL Workbench interface showing the execution of a SQL query. The query is: `SELECT * FROM data_cleaned.ecommerce_data_cleaned;`

The result grid displays the following columns: Income, Age, Experience, Married/Single, House\_Ownership, Car\_Ownership, Profession, CURRENT\_JOB\_YRS, CURRENT\_HOUSE\_YRS, Rsk\_Flag, City, and STATE. The data is sorted by Income in descending order.

The output pane shows the execution details: 1 13:42:31 SELECT \* FROM data\_cleaned.ecommerce\_data\_cleaned. Message: 251887 row(s) returned.

## 2.4. Trực quan hóa dữ liệu

### 2.4.1. Biểu đồ phân phối tỷ lệ vỡ nợ

```
import matplotlib.pyplot as plt

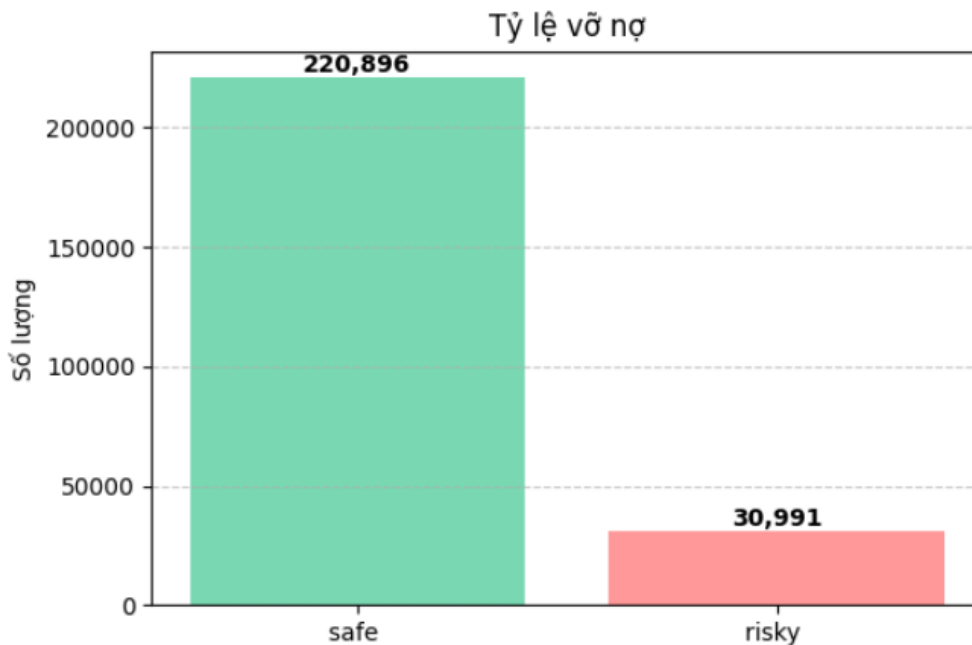
# Đếm số lượng từng lớp
value_counts = df['Risk_Flag'].value_counts()
safe_count = value_counts[0]
risky_count = value_counts[1]

safe_color = '#79D8B2' # xanh lá pastel đậm hơn
risky_color = '#FF9999' # đỏ nhạt

# Vẽ biểu đồ
plt.figure(figsize=(6, 4))
bars = plt.bar(['safe ', 'risky '], [safe_count, risky_count], color=[safe_color, risky_color])

# Hiển thị số lượng trên mỗi cột
for bar in bars:
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, height + 100, f'{height:}',
             ha='center', va='bottom', fontsize=10, fontweight='bold')

plt.title('Tỷ lệ vỡ nợ')
plt.ylabel('Số lượng')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```



Hình 1: Biểu đồ phân phối tỷ lệ vỡ nợ

### Kết luận:

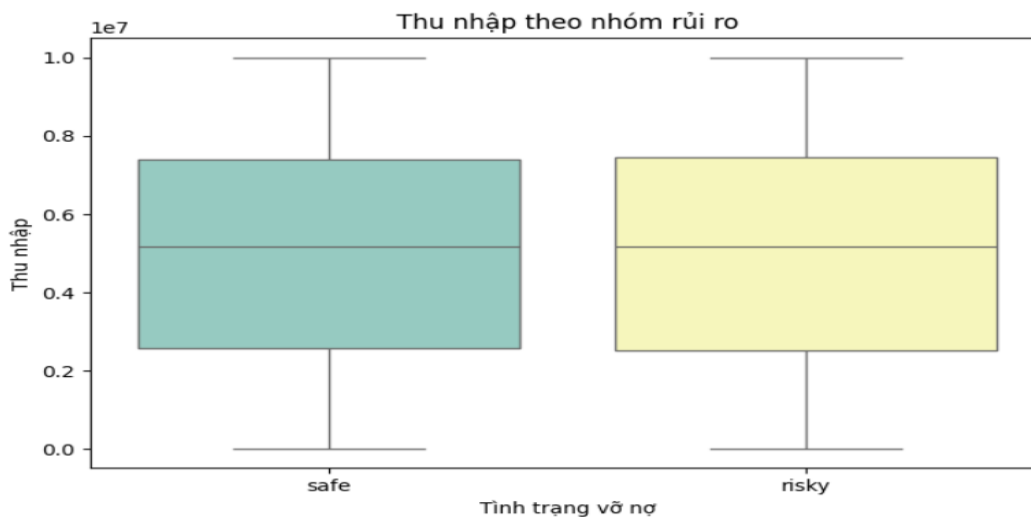
Có 30,991 người đang trong tình trạng có rủi ro vỡ nợ (chiếm ~12.3%).

Không những thế 220,896 người đang ở trong ngưỡng an toàn (chiếm ~ 87,7%).

=> Những người đang ở trong ngưỡng an toàn gấp gần 7 lần so với những người có tình trạng vỡ nợ.

### 2.4.2. Biểu đồ phân phối thu nhập theo nhóm rủi ro

```
import seaborn as sns
plt.figure(figsize=(8, 5))
sns.boxplot(x="Risk_Flag", y="Income", data=df, palette="Set3")
plt.title("Thu nhập theo nhóm rủi ro")
plt.xlabel("Tình trạng vỡ nợ")
plt.ylabel("Thu nhập")
plt.show()
```



### Kết luận:

Cả 2 nhóm đều có tỉ lệ an toàn và rủi ro trong mức từ  $0.25 \times 10^7$  đến  $0.75 \times 10^7$  ( $10^7 = 1 \times 10^7$ )

Từ đó có thể kết luận thì tỉ lệ an toàn và rủi ro không được quyết định rõ ràng thông qua thu nhập.

### 2.4.3. Biểu đồ tình trạng sở hữu nhà theo tỷ lệ rủi ro

```
import matplotlib.pyplot as plt
|
# Tính số lượng cho House_Ownership
house_counts = df.groupby(['House_Ownership', 'Risk_Flag']).size().unstack(fill_value=0)
house_categories = ['owned', 'rented', 'norent_noown'] # Đảm bảo thứ tự cố định
house_counts = house_counts.reindex(house_categories, fill_value=0) # Sắp xếp theo thứ tự

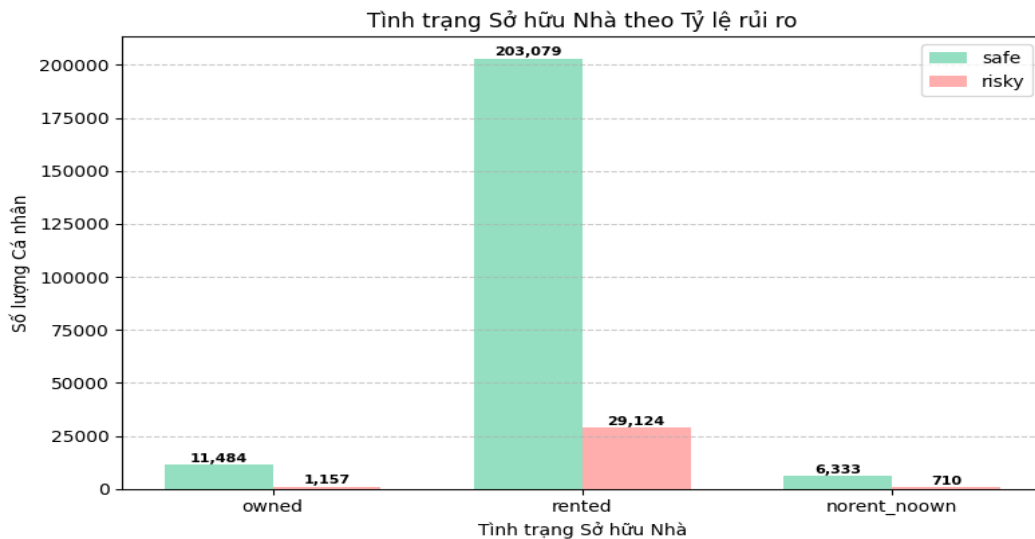
# Thiết lập biểu đồ
plt.figure(figsize=(8, 5))
bar_width = 0.35
indices = np.arange(len(house_categories))
safe_color = '#79D8B2' # Xanh lá pastel
risky_color = '#FF9999' # Đỏ nhạt

# Vẽ cột cho nhóm safe và risky
safe_bars = plt.bar(indices, house_counts['safe'], bar_width, label='safe', color=safe_color, alpha=0.8)
risky_bars = plt.bar(indices + bar_width, house_counts['risky'], bar_width, label='risky', color=risky_color, alpha=0.8)

# Hiển thị số liệu trên cột
for bar in safe_bars + risky_bars:
    height = bar.get_height()
    if height > 0: # Chỉ hiển thị nếu có giá trị
        plt.text(bar.get_x() + bar.get_width()/2, height + 50, f'{int(height):,}',
                 ha='center', va='bottom', fontsize=8, fontweight='bold')

# Tùy chỉnh biểu đồ
plt.xlabel('Tình trạng Sở hữu Nhà')
plt.ylabel('Số lượng Cá nhân')
plt.title('Tình trạng Sở hữu Nhà theo Tỷ lệ rủi ro')
plt.xticks(indices + bar_width / 2, ['owned', 'rented', 'norent_noown'], rotation=0)
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Hiển thị biểu đồ
plt.show()
```



### Kết luận:

Đối với những người đang thuê nhà thì tỉ lệ rủi ro có tới 29,124 người (chiếm đến ~94% so với tổng 30,991 người đang trong tình trạng vỡ nợ).

Do đó có thể kết luận những người có tỉ lệ vỡ nợ cao nhất là những người đang thuê nhà.

#### 2.4.4. Biểu đồ tình trạng sở hữu xe theo tỷ lệ rủi ro

```
import matplotlib.pyplot as plt
import numpy as np

# Tính số lượng cho Car_Ownership (chỉ có 'yes' và 'no')
car_counts = df.groupby(['Car_Ownership', 'Risk_Flag']).size().unstack(fill_value=0)
car_categories = ['no', 'yes'] # Đảm bảo thứ tự cố định (hoặc ['yes', 'no'] tùy theo ý muốn)
car_counts = car_counts.reindex(car_categories, fill_value=0) # Sắp xếp theo thứ tự

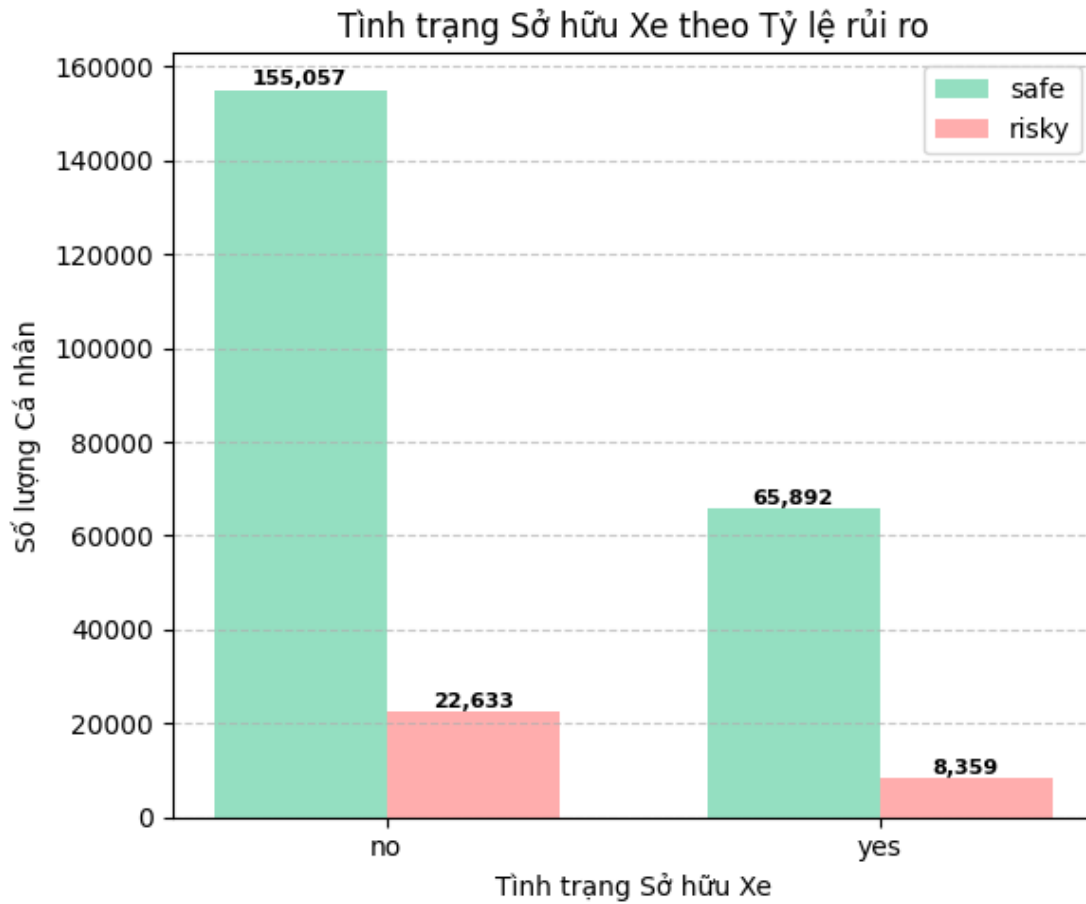
# Thiết lập biểu đồ
plt.figure(figsize=(6, 5))
bar_width = 0.35
indices = np.arange(len(car_categories))
safe_color = '#79D8B2' # Xanh lá pastel
risky_color = '#FF9999' # Đỏ nhạt

# Vẽ cột cho nhóm safe và risky
safeBars = plt.bar(indices, car_counts['safe'], bar_width, label='safe', color=safe_color, alpha=0.8)
riskyBars = plt.bar(indices + bar_width, car_counts['risky'], bar_width, label='risky', color=risky_color, alpha=0.8)

# Hiển thị số liệu trên cột
for bar in safeBars + riskyBars:
    height = bar.get_height()
    if height > 0: # Chỉ hiển thị nếu có giá trị
        plt.text(bar.get_x() + bar.get_width()/2, height + 50, f'{int(height):,}',
                 ha='center', va='bottom', fontsize=8, fontweight='bold')

# Tùy chỉnh biểu đồ
plt.xlabel('Tình trạng Sở hữu Xe')
plt.ylabel('Số lượng Cá nhân')
plt.title('Tình trạng Sở hữu Xe theo Tỷ lệ rủi ro')
plt.xticks(indices + bar_width / 2, ['no', 'yes']) # Hoặc ['yes', 'no']
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()

# Hiển thị biểu đồ
plt.show()
```



### Kết luận:

Đối với những người đang không có xe thì tỉ lệ rủi ro có tới 22,633 người (chiếm đến ~73% so với tổng 30,991 người đang trong tình trạng vỡ nợ).

Do đó có thể kết luận những người có tỉ lệ vỡ nợ cao nhất là những người đang không xe.

### 2.4.5. Biểu đồ phân bố tuổi theo tỷ lệ rủi ro

```
import matplotlib.pyplot as plt

# Tạo các khoảng tuổi
age_min, age_max = df['Age'].min(), df['Age'].max()
bins = np.linspace(age_min, age_max, 7) # Tạo 6 khoảng tuổi
bin_labels = [f'{int(bins[i])}-{int(bins[i+1])}' for i in range(len(bins)-1)]

# Tính số lượng cho mỗi khoảng tuổi
total_counts, _ = np.histogram(df['Age'], bins=bins)
safe = df[df['Risk_Flag'] == 'safe']['Age']
risky = df[df['Risk_Flag'] == 'risky']['Age']
safe_counts, _ = np.histogram(safe, bins=bins)
risky_counts, _ = np.histogram(risky, bins=bins)

# Thiết lập biểu đồ
fig, ax1 = plt.subplots(figsize=(10, 5)) # Mở rộng kích thước biểu đồ
bar_width = 0.5
indices = np.arange(len(bin_labels))
bar_color = '#4682B4' # Xanh dương đậm
safe_color = '#79D8B2' # Xanh pastel cho đường safe
line_color = '#FF9999' # Đỏ nhạt cho đường risky
```

```
# Vẽ cột cho tổng số người
bars = ax1.bar(indices, total_counts, bar_width, label='Tổng số người', color='#ADD8E6', alpha=0.8)

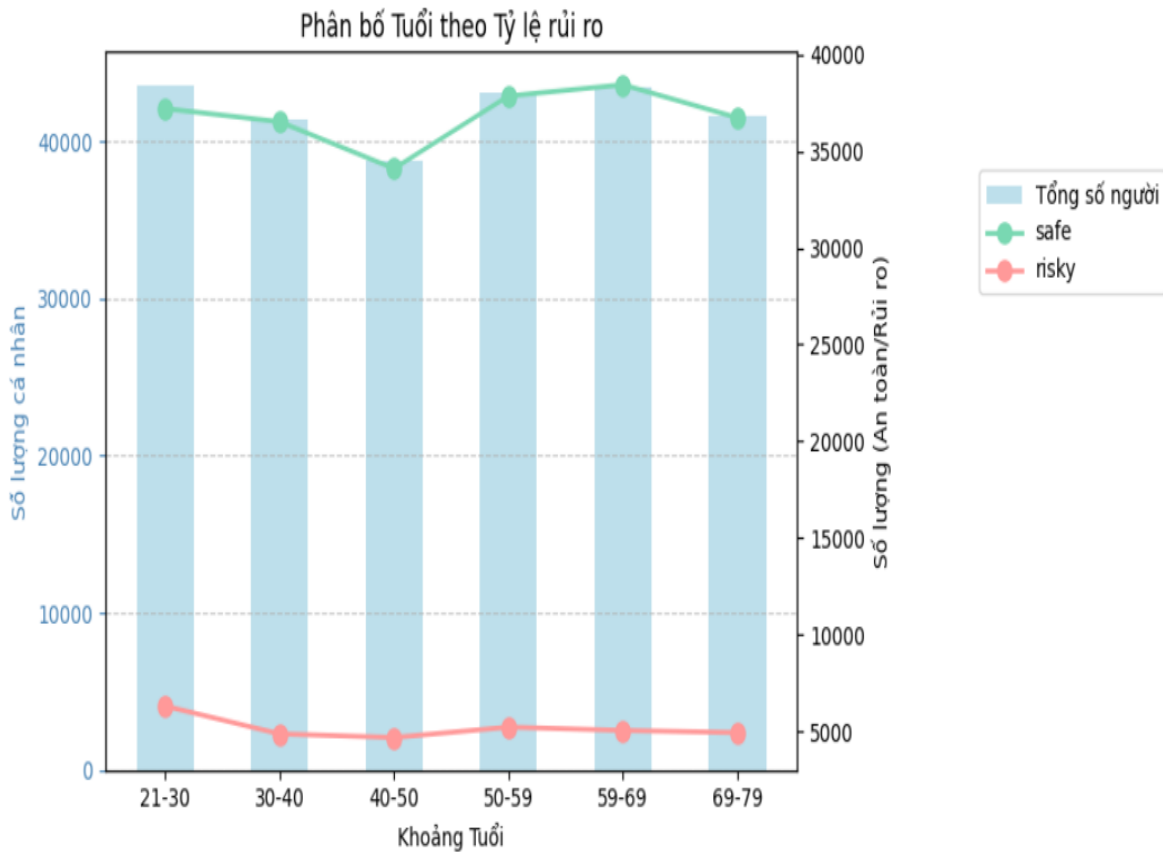
# Tạo trục y thứ hai cho đường
ax2 = ax1.twinx()
safe_line = ax2.plot(indices, safe_counts, marker='o', label='safe', color=safe_color, linewidth=2, markersize=8)[0]
risky_line = ax2.plot(indices, risky_counts, marker='o', label='risky', color=line_color, linewidth=2, markersize=8)[0]

# Tùy chỉnh biểu đồ
ax1.set_xlabel('Khoảng Tuổi')
ax1.set_ylabel('Số lượng cá nhân', color=bar_color)
ax2.set_ylabel('Số lượng (An toàn/Rủi ro)', color='black') # Màu đen để trung tính vì có 2 đường
ax1.set_title('Phân bố Tuổi theo Tỷ lệ rủi ro')
ax1.set_xticks(indices)
ax1.set_xticklabels(bin_labels)
ax1.grid(axis='y', linestyle='--', alpha=0.7)

# Tùy chỉnh màu trục y
ax1.tick_params(axis='y', colors=bar_color)
ax2.tick_params(axis='y', colors='black')

# Kết hợp chú thích từ cả hai trục và đặt bên ngoài
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines + lines2, labels + labels2, loc='center left', bbox_to_anchor=(1.25, 0.75))

# Hiển thị biểu đồ
plt.tight_layout()
plt.show()
```



### Kết luận:

Theo biểu đồ ta có thể thấy độ tuổi có tỉ lệ rủi ro cao nhất là những người nằm trong độ tuổi 21-30 tuổi.

Có thể nhận xét rằng những người trong độ tuổi 21-30 tuổi là những người có khả năng vỡ nợ cao nhất



## 2.4.6. Phân bố kinh nghiệm theo nhóm rủi ro

```
import matplotlib.pyplot as plt
# Tạo các khoảng kinh nghiệm
exp_min, exp_max = df['Experience'].min(), df['Experience'].max()
bins = np.linspace(exp_min, exp_max, 7) # Tạo 6 khoảng kinh nghiệm
bin_labels = [f'{int(bins[i])}-{int(bins[i+1])}' for i in range(len(bins)-1)]

# Tính số lượng cho mỗi khoảng kinh nghiệm
total_counts, _ = np.histogram(df['Experience'], bins=bins)
safe = df[df['Risk_Flag'] == 'safe']['Experience']
risky = df[df['Risk_Flag'] == 'risky']['Experience']
safe_counts, _ = np.histogram(safe, bins=bins)
risky_counts, _ = np.histogram(risky, bins=bins)

# Thiết lập biểu đồ
fig, ax1 = plt.subplots(figsize=(10, 5)) # Kích thước biểu đồ đủ rộng cho chú thích
bar_width = 0.5
indices = np.arange(len(bin_labels))
bar_color = '#4682B4' # Xanh dương đậm
safe_color = '#79D8B2' # Xanh pastel cho đường safe
line_color = '#FF9999' # Đỏ nhạt cho đường risky

# Vẽ cột cho tổng số người
bars = ax1.bar(indices, total_counts, bar_width, label='Tổng số người', color='#ADD8E6', alpha=0.8)

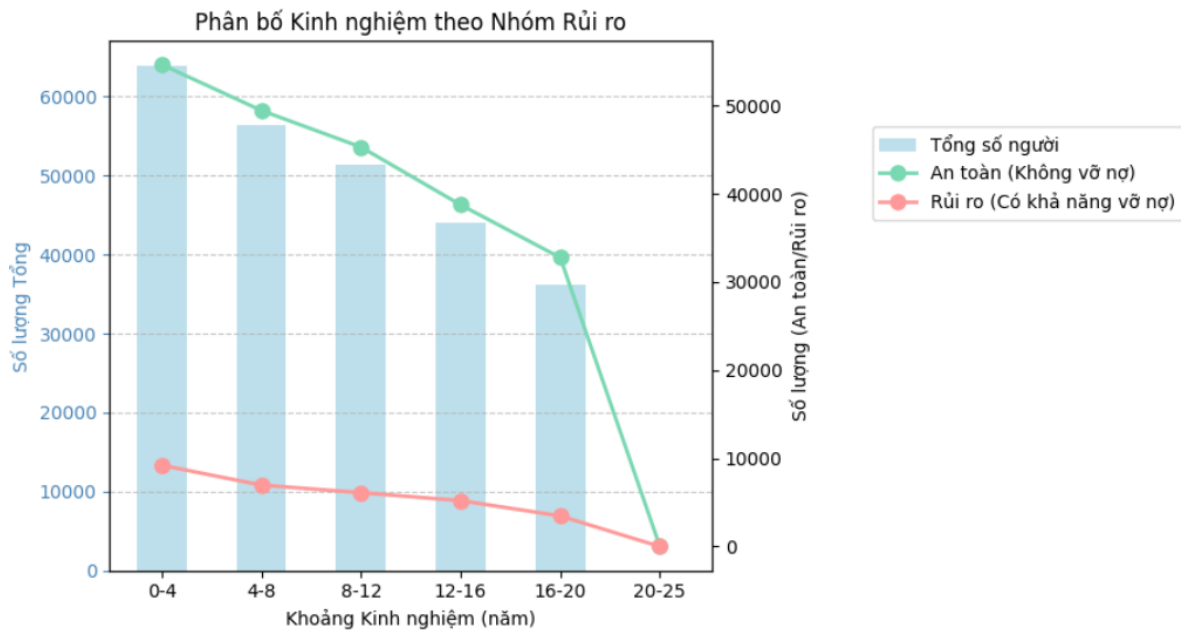
# Tạo trục y thứ hai cho đường
ax2 = ax1.twinx()
safe_line = ax2.plot(indices, safe_counts, marker='o', label='An toàn (Không vỡ nợ)', color=safe_color, linewidth=2, markersize=8)[0]
risky_line = ax2.plot(indices, risky_counts, marker='o', label='Rủi ro (Có khả năng vỡ nợ)', color=line_color, linewidth=2, markersize=8)[0]

# Tùy chỉnh biểu đồ
ax1.set_xlabel('Khoảng Kinh nghiệm (năm)')
ax1.set_ylabel('Số lượng Tổng', color=bar_color)
ax2.set_ylabel('Số lượng (An toàn/Rủi ro)', color='black') # Màu đen để trung tính vì có 2 đường
ax1.set_title('Phân bố Kinh nghiệm theo Nhóm Rủi ro')
ax1.set_xticks(indices)
ax1.set_xticklabels(bin_labels)
ax1.grid(axis='y', linestyle='--', alpha=0.7)

# Tùy chỉnh màu trục y
ax1.tick_params(axis='y', colors=bar_color)
ax2.tick_params(axis='y', colors='black')

# Kết hợp chú thích từ cả hai trục và đặt bên ngoài
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax1.legend(lines + lines2, labels + labels2, loc='center left', bbox_to_anchor=(1.25, 0.75))

# Hiển thị biểu đồ
plt.tight_layout()
plt.show()
```



### Kết luận:

Theo biểu đồ ta có thể thấy độ tuổi kinh nghiệm có tỉ lệ rủi ro cao nhất là những người có kinh nghiệm từ 0-4 năm.

Có thể nhận xét rằng những người có kinh nghiệm từ 0-4 năm là những người có khả năng vỡ nợ cao nhất.

## 2.4.7. Biểu đồ thể hiện nghề nghiệp có tỷ lệ vỡ nợ cao nhất

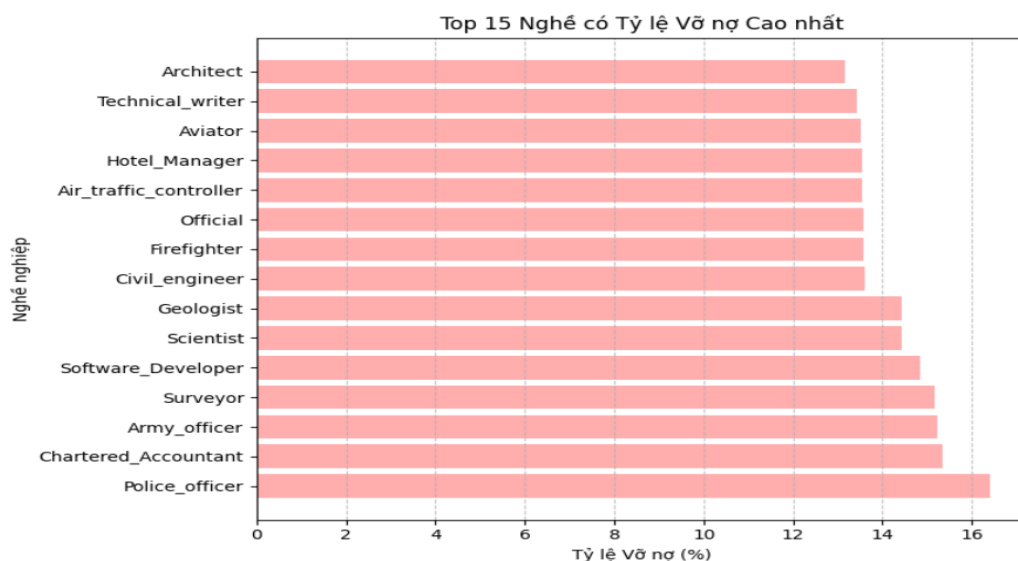
```
import matplotlib.pyplot as plt
# Tính tỷ lệ vỡ nợ theo nghề nghiệp
# Tổng số người trong mỗi nghề
total_by_profession = df.groupby('Profession').size()
# Số lượng "risky" trong mỗi nghề
risky_by_profession = df[df['Risk_Flag'] == 'risky'].groupby('Profession').size()
# Tỷ lệ vỡ nợ (%)
risk_ratio = (risky_by_profession / total_by_profession * 100).fillna(0)
# Sắp xếp theo tỷ lệ giảm dần và lấy top 15
risk_ratio = risk_ratio.sort_values(ascending=False).head(15)

# Thiết lập biểu đồ
plt.figure(figsize=(8, 6)) # Chiều cao giảm vì chỉ có 15 nghề
bar_color = '#FF9999' # Đỏ nhạt cho tỷ lệ vỡ nợ

# Vẽ biểu đồ cột ngang
bars = plt.barh(risk_ratio.index, risk_ratio, color=bar_color, alpha=0.8)

# Tùy chỉnh biểu đồ
plt.xlabel('Tỷ lệ Vỡ nợ (%)')
plt.ylabel('Nghề nghiệp')
plt.title('Top 15 Nghề có Tỷ lệ Vỡ nợ Cao nhất')
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()

# Hiển thị biểu đồ
plt.show()
```



### Kết luận:

Theo biểu đồ thì ta thấy ngành nghề có tỉ lệ vỡ nợ cao nhất là Police-officer với tỉ lệ cao nhất ngưỡng lên tới trên 16%. Các ngành đứng thứ 2,3 cũng chỉ có tỉ lệ vỡ nợ từ 15% đổ xuống.

### 2.4.8. Biểu đồ thể hiện thành phố có tỷ lệ vỡ nợ cao nhất

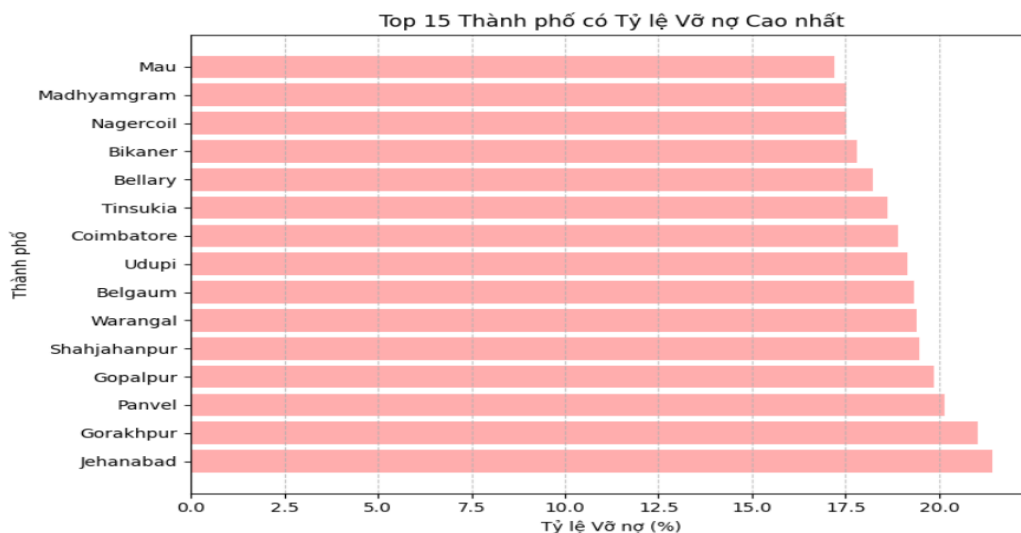
```
import matplotlib.pyplot as plt
# Tính tỷ lệ vỡ nợ theo thành phố
# Tổng số người trong mỗi thành phố
total_by_city = df.groupby('City').size()
# Số lượng "risky" trong mỗi thành phố
risky_by_city = df[df['Risk_Flag'] == 'risky'].groupby('City').size()
# Tỷ lệ vỡ nợ (%)
risk_ratio = (risky_by_city / total_by_city * 100).fillna(0)
# Sắp xếp theo tỷ lệ giảm dần và lấy top 15
risk_ratio = risk_ratio.sort_values(ascending=False).head(15)

# Thiết lập biểu đồ
plt.figure(figsize=(8, 6)) # Chiều cao phù hợp cho 15 thành phố
bar_color = '#FF9999' # Đỏ nhạt cho tỷ lệ vỡ nợ

# Vẽ biểu đồ cột ngang
bars = plt.barh(risk_ratio.index, risk_ratio, color=bar_color, alpha=0.8)

# Tùy chỉnh biểu đồ
plt.xlabel('Tỷ lệ Vỡ nợ (%)')
plt.ylabel('Thành phố')
plt.title('Top 15 Thành phố có Tỷ lệ Vỡ nợ Cao nhất')
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()

# Hiển thị biểu đồ
plt.show()
```



### Kết luận:

Theo biểu đồ thì ta thấy ngành nghề có tỉ lệ vỡ nợ cao nhất là Jehanabad với tỉ lệ khá là cao lên tới trên 22%

#### 2.4.9. Đề xuất giải pháp

Tăng cường kiểm soát rủi ro với nhóm thuê nhà và thuê xe:

- Áp dụng tiêu chí tín dụng nghiêm ngặt hơn, yêu cầu tài sản đảm bảo hoặc lịch sử tín dụng tốt.
- Cung cấp chương trình giáo dục tài chính để quản lý nợ hiệu quả.

Tập trung vào nhóm tuổi 21-30 và nghề nghiệp rủi ro cao:

- Cung cấp chính sách tín dụng linh hoạt (giảm lãi suất, tái cơ cấu nợ) cho nhóm tuổi 40-50 và nghề như kiến trúc sư, phi công.
- Tư vấn tài chính cá nhân cho các nhóm có thu nhập không ổn định.

Giảm thiểu rủi ro ở thành phố có tỷ lệ vốn nợ rủi ro cao:

- Tại Mau, Madhyamgram, Nagercoil, tăng giám sát tín dụng và nâng cao nhận thức tài chính.
- Hợp tác với tổ chức địa phương cung cấp khoản vay lãi suất ưu đãi.

Khuyến khích quản lý vốn nợ an toàn:

- Thưởng cho khách hàng duy trì vốn nợ an toàn (giảm lãi suất, tăng hạn mức).
- Xây dựng công cụ trực tuyến để theo dõi và quản lý vốn nợ.

Nghiên cứu nhóm có tỷ lệ rủi ro thấp:

- Tìm hiểu lý do nhóm sở hữu nhà/xe và nghề cảnh sát có rủi ro thấp, áp dụng kinh nghiệm cho nhóm khác.
- Khuyến khích nhóm rủi ro cao học thói quen tài chính từ nhóm rủi ro thấp qua đào tạo, hội thảo.

## CHƯƠNG 3: PHÂN TÍCH DỮ LIỆU ĐƯA VÀO MÔ HÌNH MÁY HỌC

### 3.1. Chuẩn hóa dữ liệu và mã hóa đặc trưng

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest, chi2, f_classif

# Đọc file đã làm sạch
df = pd.read_csv('Cleaned_Data.csv')

# Định nghĩa đặc trưng số và phân loại
numeric_cols = ['Income', 'Age', 'Experience', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS']
categorical_features = ['Profession', 'City', 'STATE', 'Married/Single', 'Car_Ownership', 'House_Ownership']

# 1. Đặc trưng số (numeric): chuẩn hóa
scaler = StandardScaler()
df[numeric_cols] = scaler.fit_transform(df[numeric_cols])

# 2. Đặc trưng nhị phân (2 giá trị): Label Encoding cho Married/Single và Car_Ownership
df['Married/Single'] = df['Married/Single'].str.lower().map({'single': 0, 'married': 1})
df['Car_Ownership'] = df['Car_Ownership'].map({'no': 0, 'yes': 1})

# 3. One-Hot Encoding cho House_Ownership
house_dummies = pd.get_dummies(df['House_Ownership'], prefix='House_Ownership').astype(int)
df = pd.concat([df, house_dummies], axis=1)
df.drop('House_Ownership', axis=1, inplace=True)

# 4. Chuẩn hóa Profession, City, STATE
def frequency_encoding(df, column):
    freq = df[column].value_counts(normalize=True)
    return df[column].map(freq)

for col in ['Profession', 'City', 'STATE']:
    df[col] = frequency_encoding(df, col)

# 5. Label Encoding cho cột Risk_Flag
le_risk_flag = LabelEncoder()
df['Risk_Flag'] = df['Risk_Flag'].str.lower().map({'safe': 0, 'risky': 1})

# In ra kết quả sau khi xử lý đặc trưng
print("Dữ liệu sau khi xử lý đặc trưng:")
print(df.head())
```

- numeric\_cols=[ Income, Age, Experience, CURRENT\_JOB\_YRS, CURRENT\_HOUSE\_YRS]

- Đây là các cột chứa giá trị số, và các mô hình học máy cần chuẩn hóa chúng để tất cả có phạm vi tương đương. Nếu không, các đặc trưng có phạm vi rộng hơn (ví dụ: Income) sẽ chi phối quá trình học.
- Xử lý: Dùng StandardScaler để chuẩn hóa các giá trị này, chuyển chúng về phân phối chuẩn với trung bình = 0 và độ lệch chuẩn = 1.

- categorical\_features = ['Profession', 'City', 'STATE', 'Married/Single', 'Car\_Ownership', 'House\_Ownership']

- Đây là các cột chứa giá trị phân loại, các mô hình học máy yêu cầu dữ liệu dạng số. Nếu không xử lý, dữ liệu sẽ không thể đưa vào mô hình.

+ Label Encoding: Cột: Married/Single, Car\_Ownership

- Lý do xử lý: Đây là các cột phân loại với chỉ 2 giá trị, ví dụ: "single" hoặc "married" trong Married/Single, và "yes" hoặc "no" trong Car\_Ownership.
- Xử lý: Dùng Label Encoding để chuyển đổi các giá trị chuỗi thành số. Ví dụ: "single" -> 0, "married" -> 1; "no" -> 0, "yes" -> 1.

Cột: Risk\_Flag

- Lý do xử lý: Cột này có 2 giá trị phân loại, ví dụ: "safe" và "risky".
- Xử lý: Dùng Label Encoding để mã hóa "safe" thành 0 và "risky" thành 1.

+ One-Hot Encoding: Cột: House\_Ownership

- Lý do xử lý: Cột này chứa nhiều giá trị khác nhau, ví dụ: "Owned", "Rented", "Norent\_noown". Mô hình học máy không thể xử lý trực tiếp các giá trị chuỗi.
- Xử lý: Dùng One-Hot Encoding để chuyển các giá trị này thành các cột nhị phân. Nếu có 3 giá trị như "Owned", "Rented", và "Norent\_noown", ta sẽ tạo ra 3 cột:  
House\_Ownership\_norent\_noown House\_Ownership\_owned  
House\_Ownership\_rented, mỗi cột sẽ có giá trị 1 hoặc 0.

+ Frequency Encoding: Cột: Profession, City, STATE

- Lý do xử lý: Đây là các cột chứa giá trị chuỗi, như "Engineer", "Teacher", "New York", "California", v.v. Mô hình học máy không thể làm việc trực tiếp với các giá trị chuỗi.
- Xử lý: Sử dụng Frequency Encoding để thay thế mỗi giá trị bằng tỷ lệ xuất hiện của nó trong bộ dữ liệu. Ví dụ, nếu "Engineer" xuất hiện 30%, ta sẽ thay thế "Engineer" bằng 0.3.

- Kết quả:

```
Dữ liệu sau khi xử lý đặc trưng:
Income      Age      Experience  Married/Single  Car_Ownership  Profession \
0  0.907864 -0.584421    0.123178         0             0      0.020052
1 -0.356528  0.940960   -0.050777         1             0      0.020616
2  0.442698 -0.525753   -1.268468         0             1      0.020052
3  0.058096 -0.173742    0.297134         0             0      0.017520
4  0.675441  0.823623   -0.050777         0             0      0.017520

CURRENT_JOB_YRS  CURRENT_HOUSE_YRS  Risk_Flag      City      STATE \
0      0.819348      0.716325         0  0.003370  0.101456
1     -0.593559     -1.428003         0  0.002731  0.023041
2     -1.158722      0.001549         1  0.002409  0.018477
3     -0.876140      1.431102         1  0.003211  0.065630
4     -1.723885      0.001549         0  0.003402  0.101456

House_Ownership_norent_noown  House_Ownership_owned  House_Ownership_rented
0                             0                             0                             1
1                             0                             0                             1
2                             0                             0                             1
3                             0                             0                             1
4                             0                             0                             1
```

## 3.2. Sử dụng các phương pháp kiểm định để chọn thuộc tính cần thiết cho mô hình

### 3.2.1. Kiểm định Chi Square

#### - Mục đích phương pháp

Chi-Square Test (Kiểm định Chi Bình Phương) là một phương pháp thống kê được sử dụng để kiểm tra sự độc lập giữa các biến phân loại. Mục đích của phương pháp này trong bài toán của bạn là xác định xem các đặc trưng phân loại có liên quan đến biến mục tiêu Risk\_Flag (biến phân loại với các giá trị như "safe" và "risky") hay không.

#### - Cách thức hoạt động:

- + Đặc trưng phân loại và biến mục tiêu
- + Tính toán Chi-Square
- + Xử lý dữ liệu trước khi kiểm định
- + Áp dụng Chi-Square Test trong thực tế



### - Kết quả đầu ra:

```
# 6. Phương pháp 1: Chi-Square Test cho các đặc trưng phân loại
categorical_features = ['Married/Single', 'Car_Ownership', 'House_Ownership_norent_noown',
                        'House_Ownership_owned', 'House_Ownership_rented']
X_chi2 = df[categorical_features].copy()
X_chi2 = X_chi2.apply(lambda x: x + abs(x.min()) if x.min() < 0 else x)

chi2_selector = SelectKBest(score_func=chi2, k='all')
chi2_selector.fit(X_chi2, df['Risk_Flag'])
chi2_scores = pd.DataFrame({
    'Feature': categorical_features,
    'Chi2 Score': chi2_selector.scores_,
    'P-value': chi2_selector.pvalues_
})
significant_chi2_features = chi2_scores[chi2_scores['P-value'] <= 0.05]['Feature'].tolist()

# In đặc trưng được chọn từ Chi-Square Test
print("\nĐặc trưng được chọn từ Chi-Square Test (p-value <= 0.05):")
print(significant_chi2_features)
print("\nChi tiết điểm số Chi-Square Test:")
print(chi2_scores)
```

Đặc trưng được chọn từ Chi-Square Test (p-value <= 0.05):  
['Married/Single', 'Car\_Ownership', 'House\_Ownership\_norent\_noown', 'House\_Ownership\_owned', 'House\_Ownership\_rented']

Chi tiết điểm số Chi-Square Test:

	Feature	Chi2 Score	P-value
0	Married/Single	101.410712	7.475843e-24
1	Car_Ownership	74.949690	4.828630e-18
2	House_Ownership_norent_noown	32.323878	1.304998e-08
3	House_Ownership_owned	116.222131	4.249335e-27
4	House_Ownership_rented	12.288264	4.558154e-04

### 3.2.2. Kiểm định ANOVA F-Test

#### - Mục đích phương pháp

Phương pháp ANOVA (Analysis of Variance) hoặc F-test là một bài kiểm tra thống kê dùng để xác định sự khác biệt có ý nghĩa giữa các nhóm dữ liệu. Trong bài toán của bạn, ANOVA sẽ được sử dụng để kiểm tra mối quan hệ giữa các đặc trưng số và biến mục tiêu Risk\_Flag, nhằm xác định xem các đặc trưng số có ảnh hưởng đáng kể đến khả năng phân loại "safe" hay "risky".

#### - Cách thức hoạt động:

- + Đặc trưng số và biến mục tiêu
- + Tính toán F-Score
- + P-value
- + Xử lý dữ liệu

+ Áp dụng ANOVA trong thực tế

- Kết quả đầu ra

```
# 7. Phương pháp 2: ANOVA (F-test) cho đặc trưng số
numeric_cols = ['Income', 'Age', 'Experience', 'CURRENT_JOB_YRS', 'CURRENT_HOUSE_YRS',
                'Profession', 'City', 'STATE']
anova_selector = SelectKBest(score_func=f_classif, k='all')
anova_selector.fit(df[numeric_cols], df['Risk_Flag'])
anova_scores = pd.DataFrame({
    'Feature': numeric_cols,
    'F-Score': anova_selector.scores_,
    'P-value': anova_selector.pvalues_
})
significant_anova_features = anova_scores[anova_scores['P-value'] <= 0.05]['Feature'].tolist()

# In đặc trưng được chọn từ ANOVA
print("\nĐặc trưng được chọn từ ANOVA (F-test) (p-value <= 0.05):")
print(significant_anova_features)
print("\nChi tiết điểm số ANOVA (F-test):")
print(anova_scores)
```

Đặc trưng được chọn từ ANOVA (F-test) (p-value <= 0.05):  
['Age', 'Experience', 'CURRENT\_JOB\_YRS', 'CURRENT\_HOUSE\_YRS', 'Profession', 'City']

Chi tiết điểm số ANOVA (F-test):

	Feature	F-Score	P-value
0	Income	2.529796	1.117159e-01
1	Age	118.689450	1.242319e-27
2	Experience	458.254284	1.420934e-101
3	CURRENT_JOB_YRS	98.234505	3.753050e-23
4	CURRENT_HOUSE_YRS	4.924502	2.647932e-02
5	Profession	18.778443	1.468740e-05
6	City	11.962709	5.428486e-04
7	STATE	0.012485	9.110320e-01

### 3.3. Các đặc trưng quan trọng được chọn từ 2 phương pháp trên

```
[ ] # 8. Tổng hợp các đặc trưng quan trọng từ cả hai phương pháp
selected_features = list(set(significant_chi2_features + significant_anova_features))
print("\nCác đặc trưng cuối cùng được chọn:")
print(selected_features)
```



Các đặc trưng cuối cùng được chọn:

['CURRENT\_HOUSE\_YRS', 'House\_Ownership\_owed', 'Age', 'CURRENT\_JOB\_YRS', 'Married/Single', 'Profession', 'Experience', 'House\_Ownership\_rented', 'Car\_Ownership', 'House\_Ownership\_norent\_noown', 'City']

### 3.4. Đánh labeling

```
# 9. Gán X và y
X = df[selected_features]
y = df['Risk_Flag']

# In kết quả tổng hợp
print("\nĐặc trưng (X) sau khi chọn lọc:")
print(X.head())
print("\nNhãn (y):")
print(y.head())
```

Đặc trưng (X) sau khi chọn lọc:

	CURRENT_HOUSE_YRS	House_Ownership_owned	Age	CURRENT_JOB_YRS	\
0	0.716325	0	-0.584421	0.819348	
1	-1.428003	0	0.940960	-0.593559	
2	0.001549	0	-0.525753	-1.158722	
3	1.431102	0	-0.173742	-0.876140	
4	0.001549	0	0.823623	-1.723885	

	Married/Single	Profession	Experience	House_Ownership_rented	\
0	0	0.020052	0.123178	1	
1	1	0.020616	-0.050777	1	
2	0	0.020052	-1.268468	1	
3	0	0.017520	0.297134	1	
4	0	0.017520	-0.050777	1	

	Car_Ownership	House_Ownership_norent_noown	City
0	0	0	0.003370
1	0	0	0.002731
2	1	0	0.002409
3	0	0	0.003211
4	0	0	0.003402

Nhãn (y):

0	0
1	0
2	1
3	1
4	0

Name: Risk\_Flag, dtype: int64

### 3.5 Cân bằng dữ liệu

```
from imblearn.combine import SMOTETomek

# Cân bằng dữ liệu toàn bộ trước khi chia
smote_tomek = SMOTETomek(random_state=42)
X_resampled, y_resampled = smote_tomek.fit_resample(X, y)

# Kiểm tra phân phối lớp
print("Phân phối lớp sau SMOTE + Tomek Links:")
print(pd.Series(y_resampled).value_counts(normalize=True))
```

SMOTETomek là phương pháp kết hợp giữa:

- SMOTE (Synthetic Minority Oversampling Technique): Tạo mẫu giả lập cho lớp rủi ro (1) bằng cách nội suy giữa các điểm dữ liệu, tăng số mẫu để cân bằng với lớp an toàn (0).
- Tomek Links: Loại bỏ các cặp mẫu nhiễu ở biên giới giữa hai lớp, giúp phân tách lớp rõ ràng hơn.
- random\_state=42: Đảm bảo kết quả tái lập, tránh dao động chỉ số (F1-score, Recall, v.v.) qua mỗi lần chạy.
- fit\_resample(): Học cấu trúc dữ liệu (X, y), sau đó áp dụng SMOTETomek để cân bằng tập dữ liệu đầu ra.

- Ưu điểm:

+ Kết hợp oversampling (SMOTE) và làm sạch nhiễu (Tomek Links).

+ Giảm nguy cơ overfitting

+ Hiệu quả với dữ liệu lớn, tỷ lệ mất cân bằng cao

- Nhược điểm:

- Tốn thời gian tính toán (tìm hàng xóm, nhiễu).
- Có thể tạo/xóa nhiễu không mong muốn nếu dữ liệu chồng lấn nặng.
- Phụ thuộc vào tham số k, cần thử nghiệm.

- Kết quả:

```
Phân phối lớp sau SMOTE + Tomek Links:
Risk_Flag
0      0.5
1      0.5
Name: proportion, dtype: float64
```

### 3.6. Chia tập train và test

Sử dụng `train_test_split` để chia dữ liệu thành tập huấn luyện (80%) và kiểm tra (20%), với `stratify=y_resampled` để giữ tỷ lệ lớp cân bằng (gần 50:50) và `random_state=42` để đảm bảo tính tái lập. Điều này giúp mô hình học máy được đánh giá chính xác, tổng quát hóa tốt, và phù hợp với mục tiêu phát hiện lớp rủi ro trong bài toán dự đoán vỡ nợ.

```
# Chia tập train và test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, stratify=y_resampled, random_state=42)
```

### 3.7. Xây dựng mô hình để đưa vào máy học

#### 3.7.1. Lý do chọn mô hình

Bài toán dự đoán vỡ nợ mang tính phân loại, có dữ liệu phức tạp, đòi hỏi độ chính xác cao nên Decision Tree và Random Forest là hai lựa chọn rất phù hợp.

#### 3.7.2. Mô hình Decision Tree

##### 3.7.2.1. Cơ sở lý thuyết

Decision tree là một mô hình Supervised learning, có thể được áp dụng vào cả hai bài toán Classification và Regression. Việc xây dựng một Decision tree trên dữ liệu huấn luyện cho trước là việc đi xác định các câu hỏi và thứ tự của chúng. Một điểm đáng lưu ý của decision tree là nó có thể làm việc với các đặc trưng (trong các tài liệu về Decision tree, các đặc trưng thường được gọi là thuộc tính – attribute) dạng categorical, thường là rời rạc và không có thứ tự. Ví dụ, mưa, nắng hay xanh, đỏ, v.v.

Decision Tree cũng làm việc với dữ liệu có vector đặc trưng bao gồm cả thuộc tính dạng categorical và liên tục (numeric).

#### Ưu điểm

- Cây quyết định dễ hiểu
- Ít yêu cầu việc chuẩn hoá dữ liệu.
- Cây quyết định có thể xử lý cả dữ liệu có giá trị bằng số và dữ liệu có giá trị là tên thể loại và kể cả dữ liệu bị thiếu
- Cây quyết định có thể xử lý tốt một lượng dữ liệu lớn trong thời gian ngắn

### Nhược điểm

- Khi được áp dụng với bộ dữ liệu phức tạp, nhiều biến và thuộc tính khác nhau có thể dẫn đến mô hình bị overfitting, quá khớp với dữ liệu training. Từ đó không đưa ra kết quả phân loại chính xác khi áp dụng cho dữ liệu test và dữ liệu mới.
- Khi có sự thay đổi nhỏ trong bộ dữ liệu có thể gây ảnh hưởng đến cấu trúc của mô hình.
- Thuật toán cây quyết định có khả năng “bias” hay thiên vị nếu bộ dữ liệu không được cân bằng.

#### 3.7.2.2. Huấn luyện mô hình *Decision tree*

```
) from sklearn.tree import DecisionTreeClassifier
   from sklearn.metrics import classification_report
   import time

   # Đo thời gian bắt đầu
   start_time = time.time()

   # Khởi tạo và huấn luyện mô hình Decision Tree
   model = DecisionTreeClassifier(
       random_state=42,          # để kết quả có thể lặp lại
       class_weight='balanced'  # (tùy chọn) nếu dữ liệu imbalanced
   )
   model.fit(X_train, y_train)
```

#### - Dự đoán trên tập TEST

```
y_pred = model.predict(X_test)
```

### 3.7.3. Mô hình RandomForest

#### 3.7.3.1. Cơ sở lý thuyết

Random Forests là thuật toán học có giám sát (supervised learning), có thể được sử dụng cho cả Classification và Regression. Nó cũng là thuật toán linh hoạt và dễ sử dụng nhất.

Random forests tạo ra cây quyết định trên các mẫu dữ liệu được chọn ngẫu nhiên, được dự đoán từ mỗi cây và chọn giải pháp tốt nhất bằng cách bỏ phiếu. Random forests có nhiều ứng dụng, chẳng hạn như công cụ đề xuất, phân loại hình ảnh và lựa chọn tính năng. Nó có thể được sử dụng để phân loại các ứng viên cho vay trung thành, xác định hoạt động gian lận và dự đoán các bệnh. Nó nằm ở cơ sở của thuật toán Boruta, chọn các tính năng quan trọng trong tập dữ liệu.

**Ưu điểm :**

- Không bị vấn đề Overfitting.
- Thích ứng nhanh với tập dữ liệu, thích hợp cho việc phân tích Dữ liệu lớn
- Có thể xử lý hàng ngàn tính năng đầu vào và các biến tại một thời điểm.
- Có thể được áp dụng cho các vấn đề học không được giám sát.

**Nhược điểm:**

Chậm tạo dự đoán bởi vì nó có nhiều cây quyết định.

Thời gian thực thi lâu vì sử dụng nhiều cây quyết định.

Mô hình khó hiểu hơn so với Cây quyết định.

**3.7.3.2. Huấn luyện mô hình Random Forests**

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
import time # nhớ import time nếu chưa import

# Đo thời gian bắt đầu
RS_stime = time.time()

# Tạo và huấn luyện mô hình
model = RandomForestClassifier(class_weight='balanced', random_state=42)
model.fit(X_train, y_train)
```

**- Dự đoán trên tập TEST**

```
y_pred_brf = model.predict(X_test)
```

## CHƯƠNG 4: ĐÁNH GIÁ CÁC MÔ HÌNH HỌC MÁY

### 4.1. Đánh giá mô hình Decision tree

```
print("\nDecision Tree Report:")
print(classification_report(y_test, y_pred))

# Đo thời gian thực thi
DT_ex_time = time.time() - start_time
print(f"Thời gian thực thi: {DT_ex_time:.2f} giây")
```

```
Decision Tree Report:
              precision    recall  f1-score   support

     0       0.94         0.90         0.92         43551
     1       0.90         0.94         0.92         43551

 accuracy          0.92         0.92         0.92         87102
 macro avg         0.92         0.92         0.92         87102
weighted avg         0.92         0.92         0.92         87102

Thời gian thực thi: 6.59 giây
```

#### - Nhận xét:

Độ chính xác tổng thể (accuracy):

- Mô hình đạt 92%, nghĩa là trong tổng số 87,102 mẫu, có 92% mẫu được dự đoán đúng. Đây là một độ chính xác cao.

Precision (độ chính xác theo lớp):

- Lớp 0: 94% — trong số các mẫu mà mô hình dự đoán là 0, có 94% là đúng.
- Lớp 1: 90% — trong số các mẫu mà mô hình dự đoán là 1, có 90% là đúng.  
→ Nhìn chung, precision của hai lớp khá đồng đều, chênh lệch không quá lớn.

Recall (độ nhạy theo lớp):

- Lớp 0: 90% — mô hình dự đoán đúng 90% các mẫu thực sự là 0.
- Lớp 1: 94% — mô hình dự đoán đúng 94% các mẫu thực sự là 1.  
→ Recall cho lớp 1 cao hơn một chút so với lớp 0.



F1-score (trung bình hài hòa giữa precision và recall):

- Cả lớp 0 và lớp 1 đều đạt 0.92 F1-score.  
→ Điều này cho thấy mô hình cân bằng tốt giữa precision và recall.

Macro avg và weighted avg đều bằng 0.92 → chứng tỏ mô hình khá cân bằng giữa hai lớp (không bị thiên lệch về một lớp nào).

Thời gian thực thi của mô hình: 6.59 giây

## 4.2. Đánh giá mô hình Random Forests

```
print("Balanced Random Forest Report:")
print(classification_report(y_test, y_pred_brf))

# Đo thời gian thực thi
RF_ex_time = time.time() - RS_stime
print(f"Thời gian thực thi: {RF_ex_time:.2f} giây")
```

```
➤ Balanced Random Forest Report:
              precision    recall  f1-score   support

     0       0.96         0.92         0.94         43551
     1       0.92         0.96         0.94         43551

 accuracy          0.94
 macro avg         0.94         0.94         0.94         87102
weighted avg         0.94         0.94         0.94         87102

Thời gian thực thi: 82.10 giây
```

### - Nhận xét:

Độ chính xác tổng thể (accuracy):

- Mô hình đạt 94%, tức là trong tổng số 87,102 mẫu, có 94% mẫu được dự đoán đúng.  
→ So với bảng trước (92%), mô hình này tốt hơn.

Precision (độ chính xác theo lớp):

- Lớp 0: 96% — trong số các mẫu dự đoán là 0, có 96% là đúng.
- Lớp 1: 92% — trong số các mẫu dự đoán là 1, có 92% là đúng.  
→ Precision lớp 0 nhỉnh hơn lớp 1.

Recall (độ nhạy theo lớp):

- Lớp 0: 92% — mô hình dự đoán đúng 92% các mẫu thực sự là 0.
- Lớp 1: 96% — mô hình dự đoán đúng 96% các mẫu thực sự là 1.  
→ Recall lớp 1 nhỉnh hơn lớp 0.

F1-score (trung bình giữa precision và recall):

- Cả hai lớp đều đạt 0.94 F1-score.  
→ Rất cân bằng và cao.

Macro avg và weighted avg đều là 0.94 → mô hình vẫn duy trì sự cân bằng tốt giữa hai lớp.

Thời gian thực thi: 82.10 giây

### 4.3. So sánh 2 mô hình

Tiêu chí	Mô hình Decision tree	Mô hình Random Forests
Accuracy	92%	94%
F1-score trung bình	0.92	0.94
Precision lớp 0	0.94	0.96
Recall lớp 0	0.9	0.92
Precision lớp 1	0.9	0.92
Recall lớp 1	0.94	0.96
Thời gian thực thi	6.59 giây	82.10 giây
Độ cân bằng Precision/Recall	Tốt	Rất tốt

#### 4.4. Tổng kết

Mô hình Random Forests cho độ chính xác (accuracy 94%) và F1-score cao hơn, đồng thời cân bằng tốt giữa precision và recall.

Mô hình Decision tree tuy độ chính xác thấp hơn (92%), nhưng thời gian thực thi nhanh hơn nhiều (chỉ 6.59 giây so với 82.10 giây).

Nhìn chung, cả Decision Tree và Random Forest đều cho kết quả dự đoán rất tốt. Tuy nhiên, do bài toán dự đoán vỡ nợ yêu cầu độ chính xác cao để hạn chế rủi ro tài chính, nên việc lựa chọn mô hình 2 sẽ là phương án hợp lý hơn.

## CHƯƠNG 5: KẾT LUẬN

Trong quá trình thực hiện đề tài "Dự đoán khoản vay dựa trên hành vi khách hàng", nhóm đã tiến hành toàn bộ các bước từ khảo sát dữ liệu, tiền xử lý, trực quan hóa, đến xây dựng và đánh giá các mô hình máy học. Đây là một đề tài thực tiễn, có ý nghĩa lớn đối với lĩnh vực tài chính - ngân hàng, giúp các doanh nghiệp chủ động hơn trong việc đánh giá rủi ro khoản vay và đề xuất các chiến lược phù hợp nhằm tối ưu hóa hoạt động kinh doanh.

Trước tiên, chúng em đã tiến hành xử lý dữ liệu thô bằng cách loại bỏ các giá trị thiếu, không hợp lệ, đồng thời chuẩn hóa và mã hóa các thuộc tính để dữ liệu sẵn sàng cho quá trình xây dựng mô hình. Các biểu đồ trực quan đã được sử dụng nhằm làm nổi bật mối quan hệ giữa các yếu tố như thu nhập, độ tuổi, tình trạng sở hữu nhà/xe và tỷ lệ vỡ nợ, từ đó đưa ra những phân tích trực quan và dễ hiểu.

Ở giai đoạn tiếp theo, nhóm áp dụng các phương pháp kiểm định như Chi-square và ANOVA F-test để chọn lọc những đặc trưng quan trọng nhất, đồng thời thực hiện cân bằng dữ liệu nhằm cải thiện độ chính xác của mô hình. Các mô hình Decision Tree và Random Forest đã được triển khai để thực hiện dự đoán, trong đó Random Forest cho thấy hiệu quả cao hơn về cả độ chính xác và khả năng tổng quát hóa dữ liệu.

Kết quả đánh giá dựa trên các tiêu chí về thời gian thực thi, độ chính xác (Accuracy) và sai số trung bình tuyệt đối (MAE) đã cho thấy các mô hình xây dựng đạt hiệu suất khá tốt, hoàn thành mục tiêu đã đề ra.

Thông qua quá trình thực hiện đề tài, nhóm không chỉ củng cố kiến thức về xử lý và phân tích dữ liệu mà còn tiếp cận và áp dụng thành công các kỹ thuật học máy vào một bài toán thực tiễn. Đây là bước nền tảng quan trọng cho việc nghiên cứu sâu hơn và ứng dụng dữ liệu trong lĩnh vực quản lý thông tin cũng như các lĩnh vực liên quan trong tương lai.

## TÀI LIỆU THAM KHẢO

- [1]. Decision Tree, [Decision Tree algorithm — Machine Learning cho dữ liệu dạng bảng \(machinelearningcoban.com\)](http://machinelearningcoban.com)
- [2]. [Decision Tree - GeeksforGeeks](http://www.geeksforgeeks.org/decision-tree/)
- [3.] Random Forest, <https://www.datacamp.com/community/tutorials/random-forests-classifier-python>