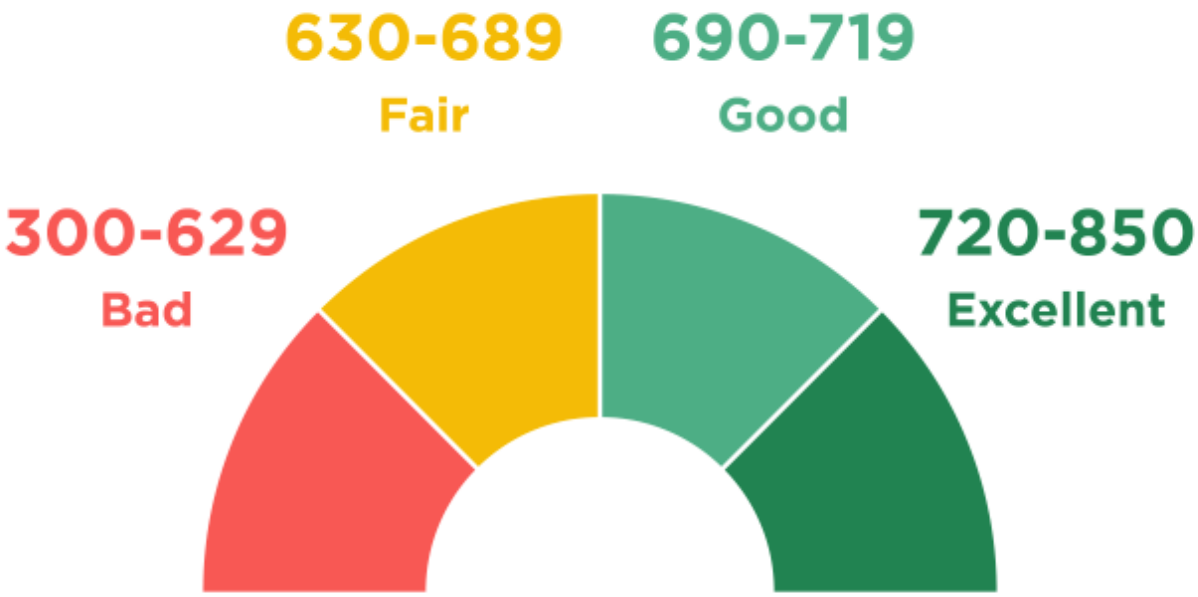
ĐÁNH GIÁ RỦI RO TÍN DỤNG BẰNG MÔ HÌNH MACHINE LEARNING



CREDIT SCORING PROJECT

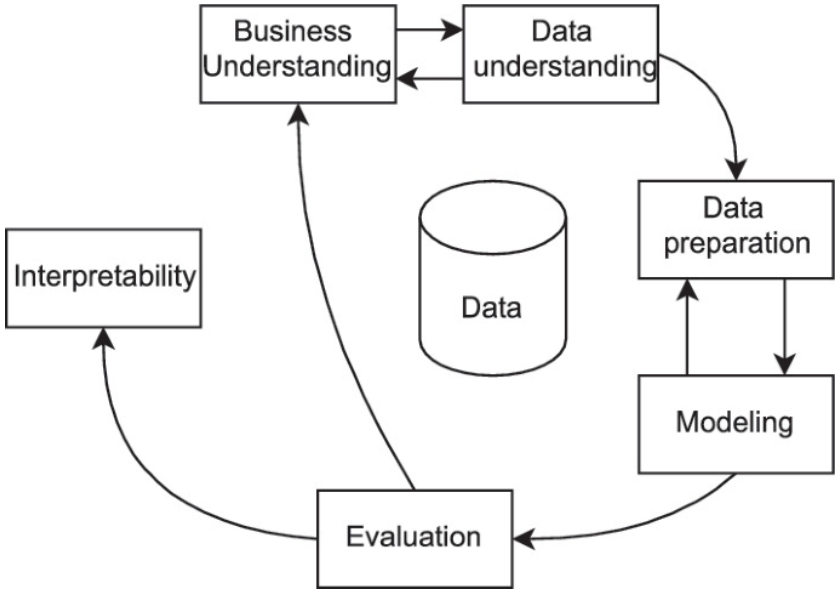
Người thực hiện : Đỗ Hoàng Nam

MỤC LỤC

LỜI MỞ ĐẦU

Đối với các tổ chức tài chính, việc hiểu rõ rủi ro tín dụng và quyết định lựa chọn khách hàng nào sẽ được cho vay hay không là yêu cầu tối quan trọng. Để làm được điều đó, họ sử dụng mô hình chấm điểm tín dụng, một trong những ứng dụng thành công nhất của mô hình nghiên cứu hoạt động và thống kê trong tài chính.

Mục đích của dự án này là kết hợp các mô hình học máy có giám sát (supervised learning) để dự đoán xác suất vỡ nợ của một nhóm cá nhân đã vay ngân hàng và phân loại chính xác họ theo xu hướng vỡ nợ của từng cá nhân.



1. DATA UNDERSTANDING

Một vấn đề lớn khi triển khai các mô hình chấm điểm tín dụng là không có dữ liệu tín dụng thực tế. Nguyên nhân là do dữ liệu tín dụng của khách hàng được bảo mật ở hầu hết các tổ chức tài chính. Vì những lý do này chúng tôi sẽ sử dụng tập dữ liệu công khai từ Kaggle.

Kaggle là một cộng đồng trực tuyến gồm các nhà khoa học dữ liệu và ngư ời thực hành học máy cung cấp các cuộc thi học máy và nền tảng dữ liệu công cộng.

Tập dữ liệu chứa thông tin về những khách hàng đã vay tiền với một tổ chức tài chính không xác định. Mục đích của việc phân tích là tìm kiếm các mối quan hệ thống kê có thể cung cấp cho chúng ta một số hiểu biết sâu sắc về rủi ro tín dụng và phát triển một số thuật toán để dự đoán rủi ro vỡ nợ tín dụng.

Do bối cảnh của tập dữ liệu không được giải thích đầy đủ nên chúng tôi sẽ không tính đến các sự kiện kinh tế vĩ mô bên ngoài, điều này có thể thay đổi hoàn toàn kết quả phân tích. Ví dụ: các quốc gia khác nhau có thể có yêu cầu về khoản vay khác nhau hoặc các giai đoạn khác nhau của chu kỳ kinh tế có thể kết thúc theo các kịch bản rất khác nhau.

Một thực tế quan trọng khác phải chú ý là chúng ta sẽ giả định tập dữ liệu không bị hệ thống điểm tín dụng làm sai lệch. Điều này có nghĩa là không có sự sàng lọc có hệ thống nào về tình trạng tín dụng của khách hàng được thực hiện cho đến ngày truy xuất dữ liệu.

Tập dữ liệu chứa 12 thuộc tính và 32581 quan sát và cấu trúc của nó được diễn giải trong Bảng bên dưới. Các biến tiềm năng mô tả cả đặc điểm của khách hàng và khoản vay. Như chúng ta có thể thấy có cả biến số và biến phân loại.

VARIABLE	DESCRIPTION	TYPE
person_age	Customers age (in years)	numerical
person_income	Annual income (in dollars)	numerical
person_home_ownership	Home ownership	categorical
person_emp_length	Employment length (in years)	numerical
loan_intent	Loan intent	categorical
loan_grade	Loan grade	categorical
loan_amnt	Loan amount (in dollars)	numerical
loan_int_rate	Loan interest rate	numerical
loan_status	Loan status	categorical
loan_percent_income	Loan percentage income	numerical
cb_person_default_on_file	Historical default (Y or N)	categorical
cb_person_cred_hist_length	Credit history length (in years)	numerical

Person_age: tuổi của khách hàng

person_income: thu nhập hàng năm (USD)

person_home_ownership: tình trạng sở hữu nhà

person_emp_length: số năm làm việc

loan_intent: mục đích khoản vay

loan_grade: xếp hạng khoản vay

loan_amnt: giá trị khoản vay (USD)

loan_int_rate: lãi suất khoản vay

loan_status: kết quả sau quá trình cho vay

loan_percent_income: tỷ lệ khoản vay trên thu nhập hàng năm

cb_person_default_on_file: lịch sử nợ xấu

cb_person_cred_hist_length: số năm đã vay nợ

1.1. Dataset information

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status
0	22	59000	RENT	123.0	PERSONAL	D	35000	16.02	1
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	0
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	1
3	23	65500	RENT	4.0	MEDICAL	C	35000	15.23	1
4	24	54400	RENT	8.0	MEDICAL	C	35000	14.27	1

```
for col in df.columns:
    print(col, ':',len(df[col].value_counts()))
```

```
person_age : 58
person_income : 4295
person_home_ownership : 4
person_emp_length : 36
loan_intent : 6
loan_grade : 7
loan_amnt : 753
loan_int_rate : 348
loan_status : 2
loan_percent_income : 77
cb_person_default_on_file : 2
cb_person_cred_hist_length : 29
```

Chú thích:

person_home_hownership có bốn cấp độ: Mortage, Own, Rent and Others

loan_intent có sáu cấp độ khác nhau: Debt consolidation, Education, Home improvement, Medical, Personal, Venture

loan_grade có bảy cấp độ: A, B, C, D, E, F, G. Cấp độ này tính đến sự kết hợp của một số chỉ số rủi ro tín dụng từ báo cáo tín dụng và đơn xin vay. Những yếu tố này có thể bao gồm mức độ hỗ trợ của người bảo lãnh, lịch sử trả nợ, dòng tiền, chi phí dự kiến hàng năm, v.v.

cb_person_default_on_file chỉ có hai cấp độ: Y nếu khách hàng đã từng có nợ xấu, N nếu không

loan_status có hai mức: 0 thể hiện không vỡ nợ, 1 thể hiện vỡ nợ.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   person_age                  32581 non-null  int64  
 1   person_income               32581 non-null  int64  
 2   person_home_ownership       32581 non-null  object 
 3   person_emp_length           31686 non-null  float64
 4   loan_intent                 32581 non-null  object 
 5   loan_grade                  32581 non-null  object 
 6   loan_amnt                   32581 non-null  int64  
 7   loan_int_rate               29465 non-null  float64
 8   loan_status                 32581 non-null  int64  
 9   loan_percent_income         32581 non-null  float64
 10  cb_person_default_on_file   32581 non-null  object 
 11  cb_person_cred_hist_length  32581 non-null  int64  
dtypes: float64(3), int64(5), object(4)
memory usage: 3.0+ MB
```

Về tổng quan có 2 cột chứa giá trị null là: personal_emp_length và loan_int_rate.

```
df.describe()
```

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_cred_hist_length
count	32581.000000	3.258100e+04	31686.000000	32581.000000	29465.000000	32581.000000	32581.000000	32581.000000
mean	27.734600	6.607485e+04	4.789686	9589.371106	11.011695	0.218164	0.170203	5.804211
std	6.348078	6.198312e+04	4.142630	6322.086646	3.240459	0.413006	0.106782	4.055001
min	20.000000	4.000000e+03	0.000000	500.000000	5.420000	0.000000	0.000000	2.000000
25%	23.000000	3.850000e+04	2.000000	5000.000000	7.900000	0.000000	0.090000	3.000000
50%	26.000000	5.500000e+04	4.000000	8000.000000	10.990000	0.000000	0.150000	4.000000
75%	30.000000	7.920000e+04	7.000000	12200.000000	13.470000	0.000000	0.230000	8.000000
max	144.000000	6.000000e+06	123.000000	35000.000000	23.220000	1.000000	0.830000	30.000000

Mô tả những dữ liệu numeric, ta thấy giá trị max của độ tuổi là 144, số năm đi làm là 123, đều không thực tế, ta sẽ khám phá sâu hơn tập dữ liệu trong phần sau.

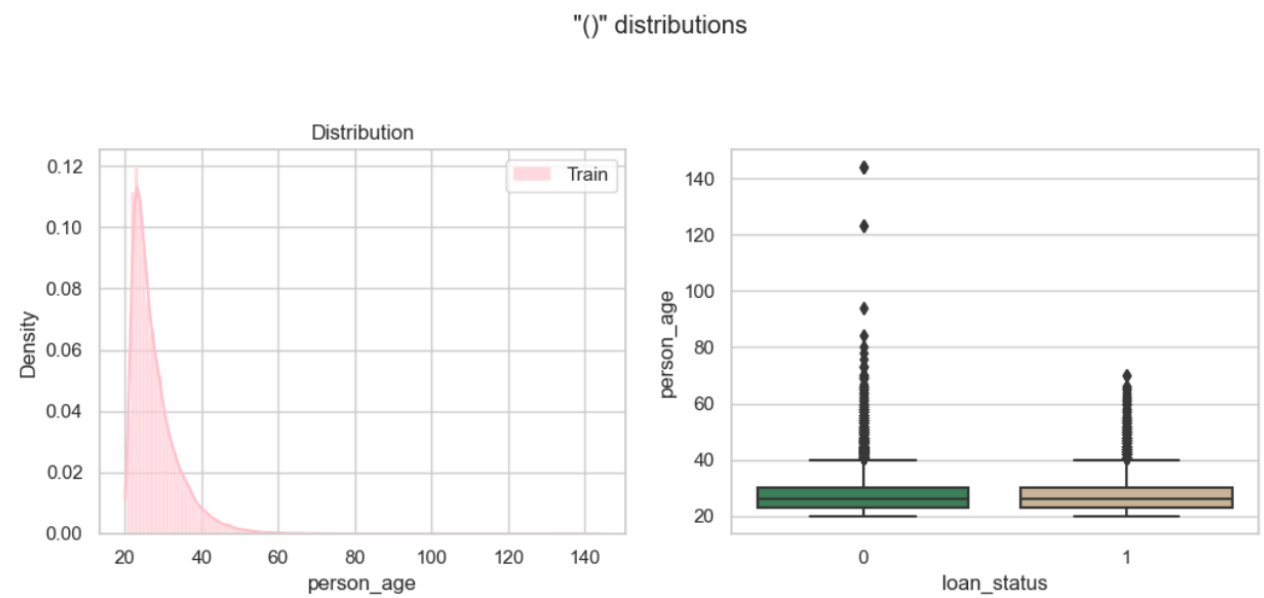
2. EXPLORATORY DATA ANALYSIS

2.1. Numeric variables

Sau khi viết hàm plot_distribution_int24 để mô tả phân phối của các biến integer phân loại dựa trên biến label, ta áp dụng với biến person_age.

Hầu hết khách hàng nằm trong khoảng từ 20-40 tuổi, không có sự khác biệt lớn về độ tuổi giữa hai biến label.

```
plot_distributions_int64('person_age')
```



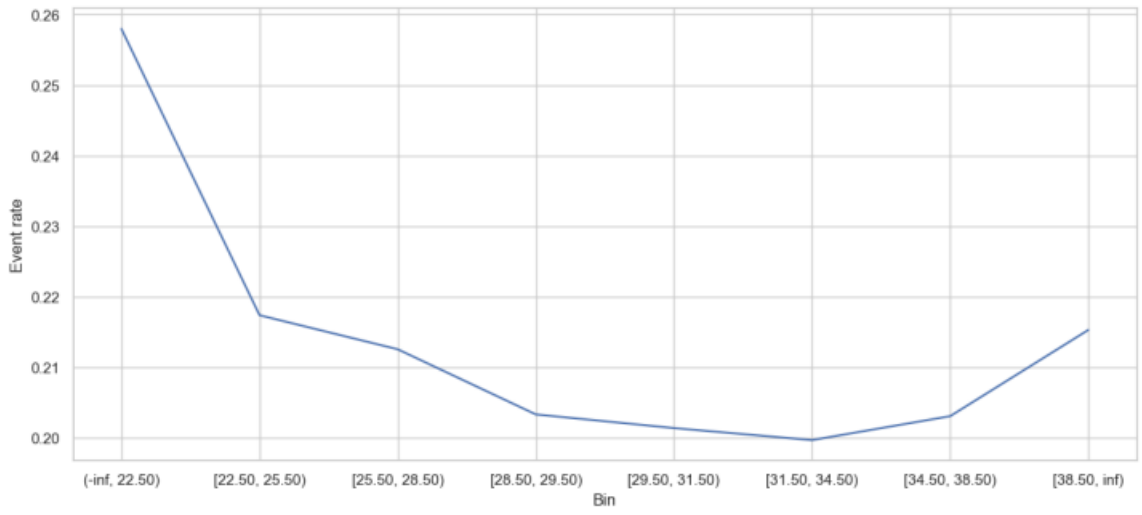
Để quan sát rõ hơn phân phối của person_age, ta chia biến này thành 8 nhóm (bin) và tính event rate của từng nhóm, sau đó biểu diễn bằng đường line.

```
x = df['person_age'].values
y = df['loan_status']
#look model for bin in next variable loan_int_rate
optb3= OptimalBinning(name='Age', dtype="numerical", solver ='cp')
optb3.fit(x, y)
age = optb3.binning_table.build()
age_bin = age[['Bin', 'Count', "Non-event", 'Event' , 'Event rate']] .loc[0:8]
age_bin.loc[0:7]  # don't show Special and Missing
```

Out[215]:

	Bin	Count	Non-event	Event	Event rate
0	(-inf, 22.50)	4877	3619	1258	0.257945
1	[22.50, 25.50)	10475	8198	2277	0.217375
2	[25.50, 28.50)	6469	5094	1375	0.212552
3	[28.50, 29.50)	1687	1344	343	0.203320
4	[29.50, 31.50)	2458	1963	495	0.201383
5	[31.50, 34.50)	2529	2024	505	0.199684
6	[34.50, 38.50)	2019	1609	410	0.203071
7	[38.50, inf)	2067	1622	445	0.215288

```
rcParams['figure.figsize'] = 14,6
sns.lineplot(data=age_bin.loc[0:7], x='Bin', y='Event rate');
```



Đối với Person_age, loan_status có mối tương quan nghịch với chỉ số này: nó giảm theo độ tăng của tuổi. Tuy nhiên, sau 34,5 tuổi xu hướng lại ngược lại, chỉ số age này không thực sự quan trọng.

Ta làm tương tự với các biến int24 khác, sau đây ta sẽ khám phá các biến Float:

Sau khi viết hàm plot_distribution_float để mô tả phân phối của các biến integer phân loại dựa trên biến label, ta áp dụng với biến loan_int_rate:

có sự khác biệt đáng kể giữa phân phối label trong khi loan_status = 0, lãi suất nằm trong khoảng 7.5%-12.5% còn đối với loan_status = 1, lãi suất nằm trong khoảng 11%-15.5%, có thể suy ra khách hàng chịu khoản vay với lãi suất cao hơn có xu hướng vỡ nợ cao hơn.

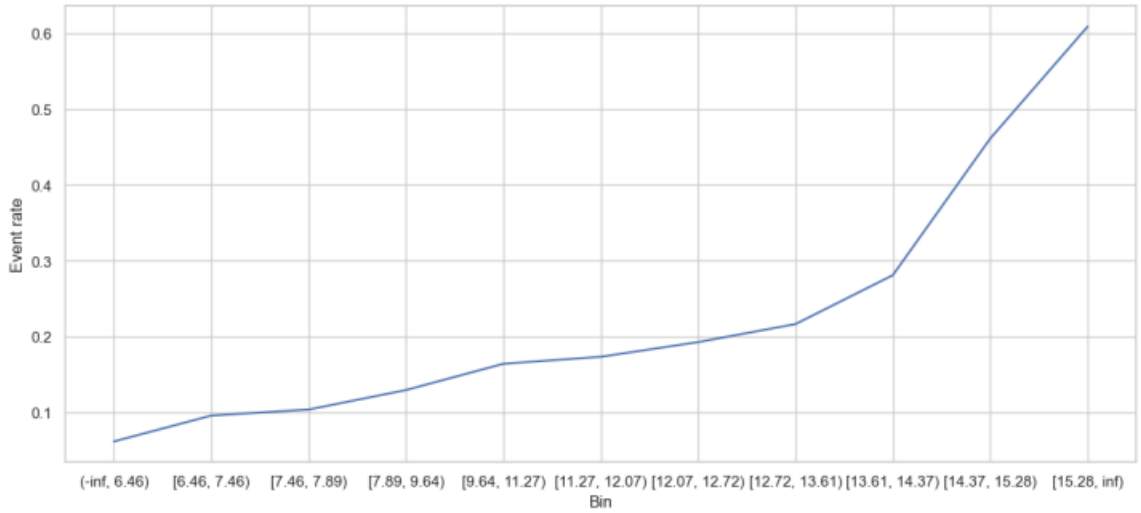
Ta chia thành các nhóm và tính event rate, thật vậy lãi suất càng cao, tỷ lệ vỡ nợ càng lớn.

```
x = df['loan_int_rate'].values
y = df['loan_status']

optb3= OptimalBinning(name='Interest', dtype="numerical", solver ='cp')
optb3.fit(x, y)
var = optb3.binning_table.build()
var_bin = var[['Bin', 'Count', "Non-event", 'Event' , 'Event rate']] .loc[0:12]
var_bin.loc[0:12]  # don't show Special and Missing
```

	Bin	Count	Non-event	Event	Event rate
0	(-inf, 6.46)	2084	1955	129	0.061900
1	[6.46, 7.46)	2531	2288	243	0.096009
2	[7.46, 7.89)	2569	2302	267	0.103931
3	[7.89, 9.64)	3186	2773	413	0.129630
4	[9.64, 11.27)	5303	4432	871	0.164247
5	[11.27, 12.07)	2652	2192	460	0.173454
6	[12.07, 12.72)	1831	1478	353	0.192791
7	[12.72, 13.61)	2891	2265	626	0.216534
8	[13.61, 14.37)	1662	1195	467	0.280987
9	[14.37, 15.28)	1755	946	809	0.460969
10	[15.28, inf)	3001	1175	1826	0.608464
11	Special	0	0	0	0.000000
12	Missing	3116	2472	644	0.206675

```
#note loc[0, var_bin.loc[0:12] -2]
rcParams['figure.figsize'] = 14,6
sns.lineplot(data=var_bin.loc[0:10], x='Bin', y='Event rate');
```



Ta chia thành các nhóm và tính event rate, thật vậy lãi suất càng cao, tỷ lệ vỡ nợ càng lớn. Ta làm tương tự với các biến float khác, sau đây ta sẽ khám phá các biến Categorical

2.2. Categorical variables

Sau khi viết hàm plot_distribution_ categorical để mô tả phân phối của các biến categorical phân loại dựa trên biến label, ta áp dụng với biến person_home_ownership:

```
df_distribution_categorical('person_home_ownership')
```

	person_home_ownership	count	count_default	default_ratio
0	MORTGAGE	13444	1690	0.125707
1	OTHER	107	33	0.308411
2	OWN	2584	193	0.074690
3	RENT	16446	5192	0.315700

Ta thấy có sự đáng biệt lớn về default_ratio giữa các category của person_home_ownership, cao nhất là Rent: với khách hàng đi thuê nhà, tỷ lệ vỡ nợ lên đến 35%, thấp nhất là 7% đối với khách hàng có sở hữu nhà. Ta sẽ xem mô tả với 2 phân loại này

```
df_r = df[df['person_home_ownership']=='RENT']
df_r.describe()
```

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_cred_hist_length
count	16446.000000	1.644600e+04	16076.000000	16446.000000	14893.000000	16446.000000	16446.000000	16446.000000
mean	27.545117	5.499775e+04	3.849216	8862.331266	11.455334	0.315700	0.182573	5.700535
std	6.418637	4.185308e+04	3.645185	5817.105516	3.124370	0.464808	0.110340	4.089008
min	20.000000	4.000000e+03	0.000000	500.000000	5.420000	0.000000	0.000000	2.000000
25%	23.000000	3.399600e+04	1.000000	4800.000000	8.940000	0.000000	0.100000	3.000000
50%	26.000000	4.800000e+04	3.000000	7500.000000	11.490000	0.000000	0.160000	4.000000
75%	30.000000	6.600000e+04	6.000000	12000.000000	13.610000	1.000000	0.250000	8.000000
max	144.000000	2.039784e+06	123.000000	35000.000000	23.220000	1.000000	0.780000	30.000000

```
df_own = df[df['person_home_ownership']=='OWN']
df_own.describe()
```

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_cred_hist_length
count	2584.000000	2.584000e+03	2410.000000	2584.000000	2356.000000	2584.000000	2584.000000	2584.000000
mean	27.698529	5.783481e+04	5.167635	9029.943885	10.861150	0.074690	0.188777	5.868421
std	6.119729	5.204285e+04	4.397831	6203.047718	3.262993	0.262942	0.116613	4.021009
min	21.000000	4.888000e+03	0.000000	900.000000	5.420000	0.000000	0.010000	2.000000
25%	23.000000	3.046550e+04	2.000000	4800.000000	7.880000	0.000000	0.100000	3.000000
50%	26.000000	4.700000e+04	4.000000	7500.000000	10.990000	0.000000	0.170000	4.000000
75%	30.000000	6.957750e+04	8.000000	12000.000000	13.240000	0.000000	0.250000	8.000000
max	60.000000	1.200000e+06	31.000000	35000.000000	21.270000	1.000000	0.720000	30.000000

Về độ tuổi và thu nhập thì 2 nhóm này không quá khác biệt. Ta khám phá tiếp biến cb_person_default_on_file: lịch sử nợ xấu

```
df_distribution_categorical('cb_person_default_on_file')
```

	cb_person_default_on_file	count	count_default	default_ratio
0	N	26836	4936	0.183932
1	Y	5745	2172	0.378068

Khách hàng từng có lịch sử nợ xấu có tỷ lệ vỡ nợ cao hơn gấp đôi so với khách hàng chưa từng có nợ xấu.

3. DATA PREPARATION

3.1. Missing value

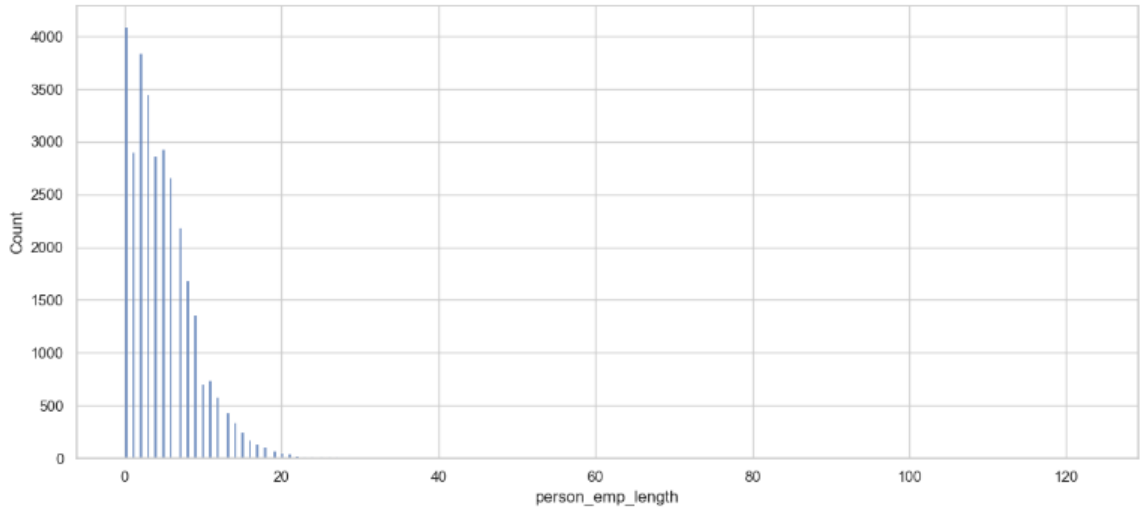
Trong phần thông tin dữ liệu ta đã biết 2 trường chứa giá trị null là: personal_emp_length và loan_int_rate.

```
np.round(df.isna().sum()* 100 / df.shape[0], 3)
```

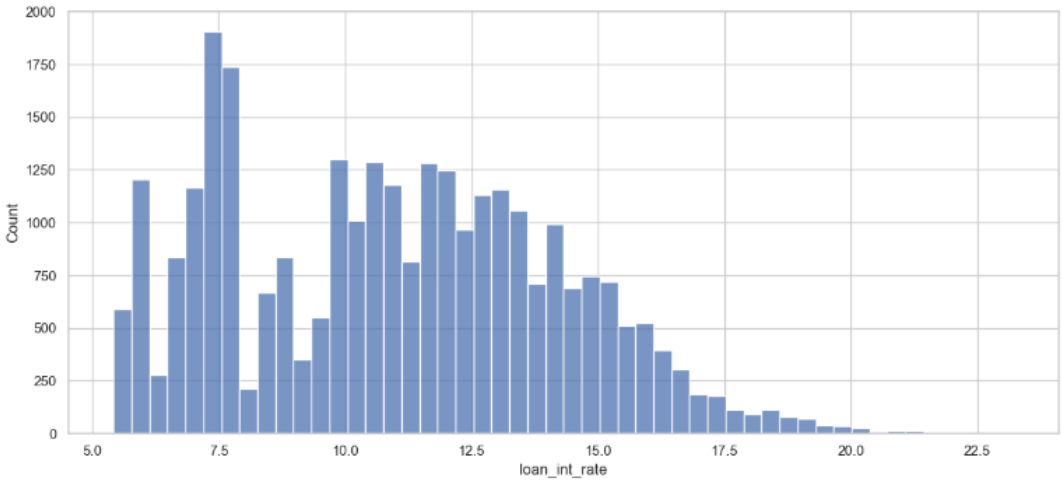
```
person_age                  0.000
person_income               0.000
person_home_ownership       0.000
person_emp_length           2.736
loan_intent                 0.000
loan_grade                  0.000
loan_amnt                   0.000
loan_int_rate               9.548
loan_status                 0.000
loan_percent_income         0.000
cb_person_default_on_file   0.000
cb_person_cred_hist_length  0.000
dtype: float64
```

Ta biểu diễn phân phối của 2 biến này

```
sns.histplot(df['person_emp_length']);
#for col in df.columns:
```




```
sns.histplot(df['loan_int_rate']);
```



Phân phối của 2 biến people_emp_length và loan_int_rate bị lệch, có các ngoại lệ. Vì vậy, việc impute bằng trung vị (median) được ưu tiên hơn.

```
df.person_emp_length.fillna(df.person_emp_length.median(), inplace=True)
df.loan_int_rate.fillna(df.loan_int_rate.median(), inplace=True)
df.isnull().sum()
```

```
person_age                    0
person_income                 0
person_home_ownership         0
person_emp_length             0
loan_intent                   0
loan_grade                    0
loan_amnt                     0
loan_int_rate                 0
loan_status                   0
loan_percent_income           0
cb_person_default_on_file     0
cb_person_cred_hist_length    0
dtype: int64
```

3.2. Drop duplicates

```
dups = df.duplicated()
```

```
df[dups]
```

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status
15975	23	42000	RENT	5.0	VENTURE	B	6000	9.99	0
15989	23	90000	MORTGAGE	7.0	EDUCATION	B	8000	10.36	0
15995	24	48000	MORTGAGE	4.0	MEDICAL	A	4000	5.42	0
16025	24	10000	RENT	8.0	PERSONAL	A	3000	7.90	1
16028	23	100000	MORTGAGE	7.0	EDUCATION	A	15000	7.88	0
...	...	...	...	...	...	...	...	...	...
32010	42	39996	MORTGAGE	2.0	HOMEIMPROVEMENT	A	2500	5.42	0
32047	36	250000	RENT	2.0	DEBTCONSOLIDATION	A	20000	7.88	0
32172	49	120000	MORTGAGE	12.0	MEDICAL	B	12000	10.99	0
32259	39	40000	OWN	4.0	VENTURE	B	1000	10.37	0
32279	43	11340	RENT	4.0	EDUCATION	C	1950	NaN	1

165 rows × 12 columns

Có 165 dòng bị lặp lại trong tập dữ liệu. ta thực hiện bỏ chúng

```
df.drop_duplicates(inplace=True)
```

```
df.shape
```

```
(32416, 12)
```

Bởi vì loan_percent_income = loan_amnt/person_income, nên chúng ta bỏ cột loan_percent_ income.

```
df[['person_income', 'loan_amnt', 'loan_percent_income']].head()
```

	person_income	loan_amnt	loan_percent_income
0	59000	35000	0.59
1	9600	1000	0.10
2	9600	5500	0.57
3	65500	35000	0.53
4	54400	35000	0.55

```
df.drop('loan_percent_income', axis=1, inplace=True)
```

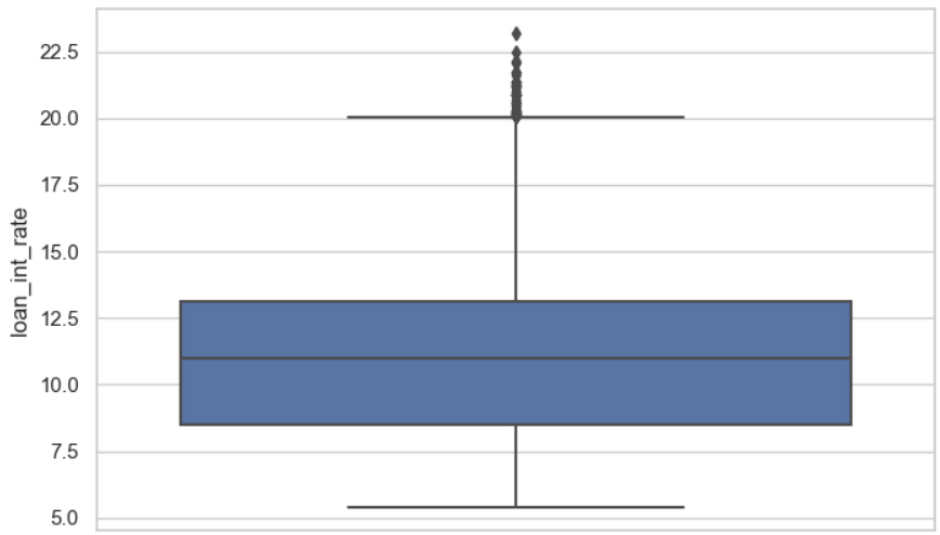
3.3. Handling outliers

Vì chúng tôi đã vẽ box_plot chứa các giá trị ngoại lệ - outliers của các biến ở trên nên chúng ta sẽ define hàm detect_outliers để xác định outliers dựa quantile và xử lý các giá trị ngoại lệ trong bước này bằng cách thay thế outlier bằng lower_whisker hoặc upper_whisker.

```
#Find outliers:
detect_outlier('loan_int_rate')
```

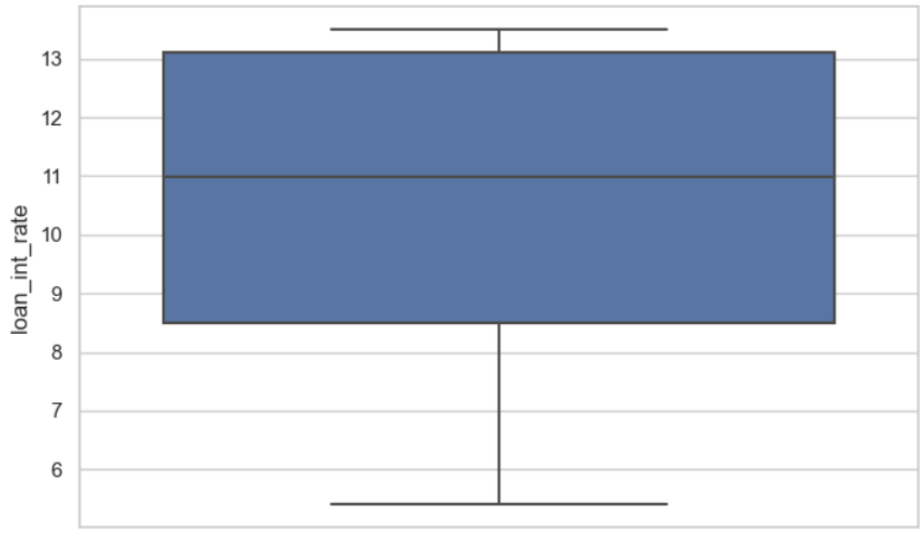
```
lower_whisker: 1.5600000000000014
upper whisker: 20.04
Number of outliers: 70
% outliers:  0.2159427443237907 %
```

```
boxplot('loan_int_rate')
```



```
new_df('loan_int_rate',0, 0.22)
```

```
boxplot(df_cleaned['loan_int_rate'])  #df_cleaned not df
```



Làm tương tự với các biến Numeric còn lại. Với loan_int_rate, mức 20% là thực tế, có ý nghĩa trong nên chúng ta không xóa những giá trị ngoại lệ này.

```
new_df('person_age',0, 0.046)
new_df('person_income',0, 0.046)
new_df('person_emp_length',0, 0.0263)
new_df('loan_amnt',0, 0.0518)
#new_df('loan_int_rate',0, 0.00216)
new_df('cb_person_cred_hist_length',0, 0.036)
```

```
df_cleaned
```

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_grade	loan_amnt	loan_int_rate	loan_status
0	22	59000	RENT	14.0	PERSONAL	D	23100	13.49	1
1	21	9600	OWN	5.0	EDUCATION	B	1000	11.14	0
2	25	9600	MORTGAGE	1.0	MEDICAL	C	5500	12.87	1
3	23	65500	RENT	4.0	MEDICAL	C	23100	13.49	1
4	24	54400	RENT	8.0	MEDICAL	C	23100	13.49	1
...	...	...	...	...	...	...	...	...	...
32576	40	53000	MORTGAGE	1.0	PERSONAL	C	5800	13.16	0
32577	40	120000	MORTGAGE	4.0	PERSONAL	A	17625	7.49	0
32578	40	76000	RENT	3.0	HOMEIMPROVEMENT	B	23100	10.99	1
32579	40	140000	MORTGAGE	5.0	PERSONAL	B	15000	11.48	0
32580	40	42000	RENT	2.0	MEDICAL	B	6475	9.99	0

32416 rows × 11 columns

Số lượng dòng, cột của khung dữ liệu và giá trị duy nhất của các biến không thay đổi.

4. MODELLING

4.1. Feature Engineering

Machine learning model cũng cần có Input và Output. Và không phải Input nào cũng phù hợp

Input

Ví dụ dạng data là "chữ" hay "categorical", đều cần phải tiền xử lý trước (convert nó sang number) để Machine Learning model mới hiểu được, ta có thể thực hiện bằng cách sử dụng biến giả hoặc one-hot-vector.

Ta sử dụng biến giả để chuyển đổi biến phân loại thành kiểu số để chạy mô hình.

```
df_cleaned_dum = pd.get_dummies(df_cleaned)
```

```
df_cleaned_dum
```

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	cb_person_cred_hist_length	person_home_ownership_MORTGAGE
0	22	59000	14.0	23100	13.49	1	3	False
1	21	9600	5.0	1000	11.14	0	2	False
2	25	9600	1.0	5500	12.87	1	3	True
3	23	65500	4.0	23100	13.49	1	2	False
4	24	54400	8.0	23100	13.49	1	4	False
...	...	...	...	...	...	...	...	..
32576	40	53000	1.0	5800	13.16	0	15	True
32577	40	120000	4.0	17625	7.49	0	15	True
32578	40	76000	3.0	23100	10.99	1	15	False
32579	40	140000	5.0	15000	11.48	0	15	True
32580	40	42000	2.0	6475	9.99	0	15	False

32416 rows × 26 columns

Output

chúng ta đã analyze và biết được output cuối cùng cho bài toán này là "1":default hoặc "0":not default (cột loan_status)

Chúng ta sẽ theo convention của Data Science, đặt X là input data, y là output (hay target)

```
X = df_cleaned_dum.drop('loan_status', axis=1).copy()
y = df_cleaned_dum['loan_status']
#y = y[X.index]
X.shape, y.shape
```

```
((32416, 25), (32416,))
```

Train/Test selection

Train dataset để model sẽ học được tính chất của phân bố trong data. Test dataset là để chúng ta kiểm thử lại độ hiệu quả của model bao gồm có "học quá sâu không? hay là học có đủ tốt chưa?"

Cách làm rất đơn giản, train và test sẽ được sample ngẫu nhiên trong feature data. Test size nên chọn khoảng 20% - 30%. Chúng ta sẽ dùng thư viện sklearn, 1 thư viện chuyên giải quyết các bài toán Machine Learning

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)
```

Theo lý thuyết thống kê, nếu quá trình sample đủ tốt thì cả 2 dataset (X_train và X_test) đều mô phỏng giống "tính chất" của nhau.

Model sẽ có thể suy đoán tốt các tính chất của X_test kể cả model không được biết trước nó như thế nào.

Data Scientist gọi nó là predictive model (mô hình dự đoán)

Ta sử dụng thuật toán StandardScaler để để quy đổi tỷ lệ của các giá trị khác nhau để so sánh.

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Kiểm tra lại label ratio ở tập test, ta thấy dữ liệu khá mất cân bằng.

```
y_test.value_counts(normalize=True)
```

```
loan_status
0    0.781285
1    0.218715
Name: proportion, dtype: float64
```

4.2. Model Selection

Với output chỉ có 2 kết quả là “1” & “0” hay "Good" & "Bad" thì đây người ta gọi là bài toán binary-classification

Bên cạnh bài toán classification thì còn có những bài toán regression (output là 1 khoảng điểm - continuous).

Có rất nhiều thuật toán Machine Learning sẵn có hiện nay ví dụ:

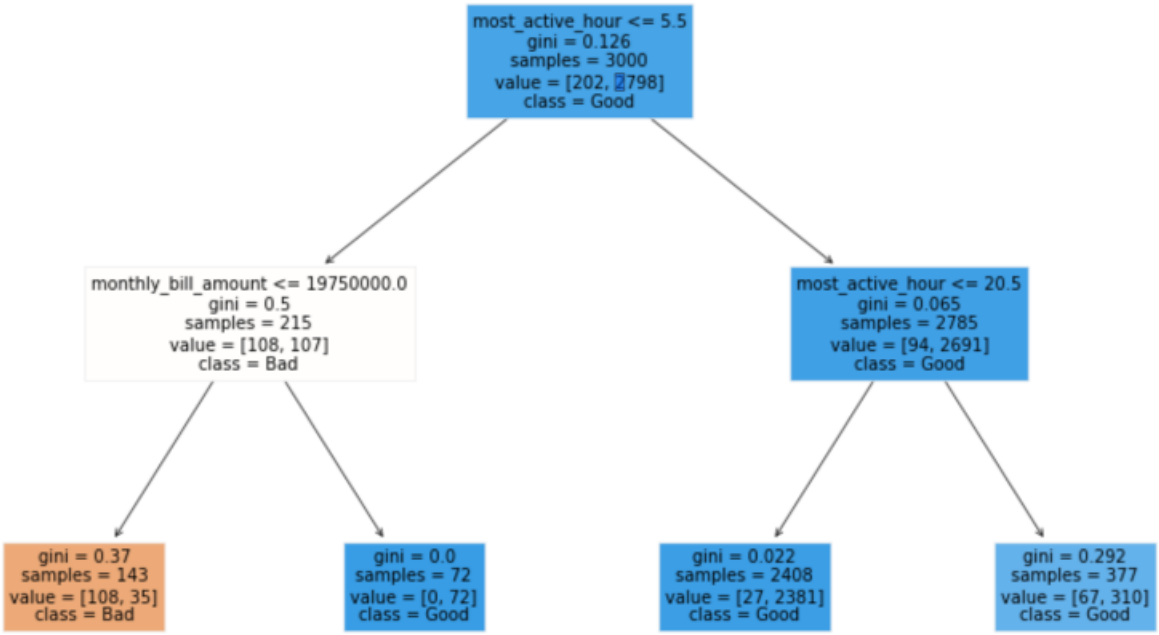
- Classifier (dành cho classification problem)
  - Nearest Neighbors
  - Linear SVM
  - RBF SVM
  - Gaussian Process
  - Decision Tree
  - Random Forest
  - Neural Net
  - AdaBoost
  - Naive Bayes
  - QDA
  - ...
- Regressor (dành cho regression problem)
  - Gradient Boosting Regression
  - Elastic Net Regression
  - Stochastic Gradient Descent Regression
  - Support Vector Machine
  - Bayesian Ridge Regression

- CatBoost Regressor
- Kernel Ridge Regression
- Linear Regression
- XGBoost Regressor
- LGBM Regressor
- ...

Mỗi mô hình có nhược và ưu riêng, bao gồm cả bộ tham số riêng. Việc hiểu và chọn model nào tốt cũng là 1 quá trình học hỏi lâu dài. Trong phạm vi dự án, ta sẽ áp dụng 3 thuật toán là LOGISTIC REGRESSION, DECISION TREE và XGBOOST.

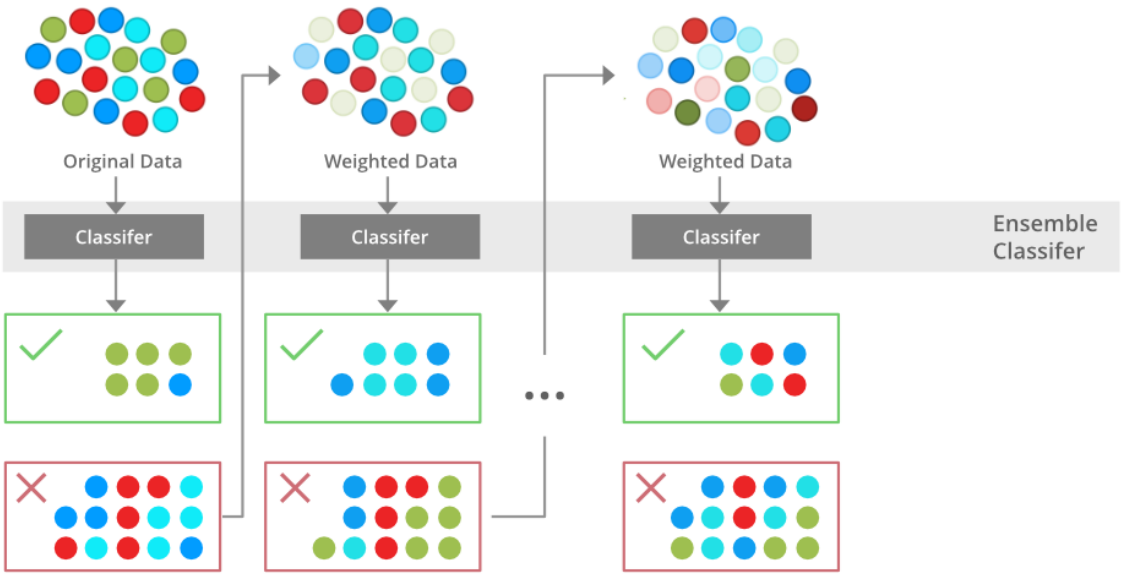
Mình sẽ làm ví dụ 2 mô hình phổ biến hiện nay là Decision Tree và XGBoost.

Decision Tree là 1 mô hình phân nhánh (giống if-else condition), ví dụ với hình minh họa bên dưới, nếu most_active_hour <= 5.5 model sẽ xét điều kiện tiếp theo monthly_bill_amount <= 19.75M, thì khách hàng được phân loại là "Bad" còn lại là "Good".



Trong khi decision tree dựa trên phương pháp bagging còn XGBOOST dựa trên kỹ thuật boosting. XGBoost là một thuật toán học máy tập trung vào cây quyết định (decision trees) được kết hợp lại thông qua phương pháp boosting. Nó có khả năng xử lý cả bài toán phân loại và hồi quy.

Đọc thêm: phân loại decision tree và XGBoost (link)



4.3. Training Model

LogisticRegression

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(random_state= 0).fit(X_train, y_train)
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
```

XGBoost

Thông số max_deep là độ sâu của cây quyết định, learning_rate: tỷ lệ học

```
import xgboost as xgb
default_xgb_model = xgb.XGBClassifier(max_depth=8, learning_rate=0.1 ,n_jobs=-1).fit(X_train, y_train)
```

4.4. Evaluate Model

Sau khi train xong, chúng ta cần kiểm tra model performance trên tập test_set để đảm bảo rằng model có khả năng generalize với unseen data.

Thông thường chúng ta sẽ hay nghe nhiều đến accuracy thể hiện sự dự đoán chính xác nhưng nó chỉ đúng với có nghĩa với balance dataset.

Đối với unbalanced dataset sẽ không có nhiều ý nghĩa.

Khi nói đến dự đoán so với thực tế, sẽ có 4 trường hợp xảy ra trong bài toán này

- Dự đoán 1 và thực tế 1 (True Positive = TP)
- Dự đoán 0 và thực tế 1 (False Negative = FN)
- Dự đoán 1 và thực tế 0 (False Positive = FP)
- Dự đoán 0 và thực tế 0 (True Negative = TN)

		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP), hit	False negative (FN), type II error, miss, underestimation
	Negative (N)	False positive (FP), type I error, false alarm, overestimation	True negative (TN), correct rejection

Trong đó accuracy được hiểu là 2 trường hợp Dự đoán 0 và thực tế 0 và Dự đoán 1 và thực tế 1 (TN + TP)

Nếu số lượng 0 chiếm nhiều hơn 1 thì accuracy sẽ chỉ miêu tả phần 0 còn phần 1 sẽ bị giảm nhẹ. Khiến độ đo accuracy không còn hữu dụng trong unbalanced dataset

Ví dụ: Số lượng thực tế 0 & 1 là 90:10. Nếu mô hình đơn giản chỉ cần dự đoán 100KH này là 0 thì accuracy là 90/100 = 90%. 90% accuracy là một số ấn tượng nhưng model không dự đoán được 1 là ko hợp lý

Để đối phó với vấn đề unbalanced dataset, người ta thường sử dụng các độ đo đánh giá khác nhau như:

- Precision
- Recall
- F1-score
- Area Under the ROC Curve (AUC-ROC)
- Area Under the Precision-Recall Curve (AUC-PR).

Các độ đo này tập trung vào việc đánh giá hiệu suất của mô hình một cách chi tiết hơn, bằng cách tính toán các giá trị dựa trên số lượng dự đoán đúng và sai của từng lớp

Precision và Recall

Là 2 độ đo phổ biến trong unbalanced dataset. Thường các bài toán sẽ cân nhắc chọn tối ưu 1 trong 2 (ít trường hợp đạt được tối ưu cả 2 trong thực tế). Và lựa chọn ưu tiên giữa Precision hay Recall là phù thuộc vào bài toán doanh nghiệp

Trong binary labeling, chúng ta thường hay đặt giá trị label là 0 và 1, trong đó 1 còn được gọi là positive label.

Bài toán Credit Scoring, thông thường chúng ta kiếm khách hàng rủi ro nên ưu tiên chọn 0 là Good và 1 là Bad

Giải thích sự ưu tiên ở 2 độ đo trong bài toán này:

- Nếu bạn chọn ưu tiên Precision cao hơn và chấp nhận Recall thấp xuống: đồng nghĩa với việc doanh nghiệp muốn đảm bảo rằng những khách hàng được dự đoán là "0" thực sự có khả năng là "0" cao. Điều này giúp doanh nghiệp đảm bảo rằng những khách hàng có khả năng trả nợ đúng hạn và đáng tin cậy được chấp nhận cho vay, giảm thiểu rủi ro tín dụng và tăng tính ổn định cho công ty. (hay nói tắt là an toàn là trên hết)

- Nếu bạn chọn ưu tiên Recall cao hơn và chấp nhận Precision thấp xuống: điều này đồng nghĩa với việc họ muốn đảm bảo rằng doanh nghiệp có thể phát hiện được càng nhiều khách hàng "0" càng tốt. Điều này có thể giúp đánh giá cao cơ hội kinh doanh với khách hàng tiềm năng và tăng tổng số lượng khách hàng chấp nhận cho vay. Tuy nhiên, việc chấp nhận nhiều khách hàng có thể tăng nguy cơ cho vấn đề nợ xấu (khách hàng thực tế không trả nợ đúng hạn). (hay nói tắt là càng quét càng nhiều càng tốt)

Đọc thêm: Precision&Recall (link)

Giờ chúng ta cũng đã hiểu trực giác 2 độ đo Precision và Recall. Bắt đầu tính thử với bài toán này

Trước hết chúng ta lấy kết quả của các model đã train ở phần trên, đầu tiên là mô hình Logistics Regression:

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

```
from sklearn import metrics
cm = metrics.confusion_matrix(y_test, y_pred)
```

Dựa vào confusion matrix, Logistics Regression đạt được kết quả như sau:

```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.87      0.96      0.91      7598
           1       0.77      0.49      0.60      2127

    accuracy                           0.86      9725
   macro avg       0.82      0.72      0.76      9725
weighted avg       0.85      0.86      0.84      9725
```

Decision Tree đạt được kết quả như sau:

```
from sklearn.metrics import confusion_matrix
y_pred = dt1.predict(X_test)
confusion_matrix(y_test, y_pred)
```

```
array([[7482,   88],
       [ 766, 1389]], dtype=int64)
```

```
cm = metrics.confusion_matrix(y_test, y_pred)
```

```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.91      0.99      0.95      7570
           1       0.94      0.64      0.76      2155

    accuracy                           0.91      9725
   macro avg       0.92      0.82      0.86      9725
weighted avg       0.91      0.91      0.91      9725
```

XGBoost đạt được kết quả như sau:

```
from sklearn import metrics
y_pred = default_xgb_model.predict(X_test)
cm = metrics.confusion_matrix(y_test, y_pred)
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.99      0.96      7598
           1       0.96      0.73      0.83      2127

    accuracy                           0.93      9725
   macro avg       0.94      0.86      0.89      9725
weighted avg       0.93      0.93      0.93      9725
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

```
0.9333676092544987
```

Trong 3 mô hình thì XGBoost có số điểm cao nhất, ta chọn thuật toán này để tiếp tục cải thiện độ chính xác của mô hình dự báo điểm tín dụng.

5. IMPROVE MODEL WITH GRIDSEARCH

Ta sẽ tiếp tục cải thiện mô hình XGBoost bằng GridSearch để tìm ra các thông số max_depth phù hợp cho mô hình.

```
from sklearn.model_selection import GridSearchCV
parameters = {'max_depth': [4, 5, 6, 7, 8, 9]}
```

```
xgb = xgb.XGBClassifier()
default_xgb_model = GridSearchCV(xgb, parameters)
```

```
default_xgb_model.fit(X_train, y_train)
```

```
print('Accuracy of XGB classifier on training set: {:.6f}'
      .format(default_xgb_model.score(X_train, y_train)))
```

```
Accuracy of XGB classifier on training set: 0.946543
```

```
default_xgb_model.best_params_
```

```
{'max_depth': 5}
```

Kết quả GridSearch đưa ra thông số tốt nhất max_depth = 5, ta kiểm tra xem GridSearch xem có tốt hơn hay không

```
xgb_model = xgb.XGBClassifier(max_depth=5).fit(X_train, y_train)

print('Accuracy of XGB classifier on training set: {:.5f}'
      .format(xgb_model.score(X_train, y_train)))
#if error, need run code import advance
```

```
Accuracy of XGB classifier on training set: 0.94654
```

```
print('Accuracy of XGB classifier on training set: {:.5f}'
      .format(xgb_model.score(X_test, y_test)))
```

```
Accuracy of XGB classifier on training set: 0.93224
```

```
xgb_model = xgb.XGBClassifier(max_depth=8).fit(X_train, y_train)

print('Accuracy of XGB classifier on training set: {:.5f}'
      .format(xgb_model.score(X_train, y_train)))
```

```
Accuracy of XGB classifier on training set: 0.97929
```

```
print('Accuracy of XGB classifier on training set: {:.5f}'
      .format(xgb_model.score(X_test, y_test)))
```

```
Accuracy of XGB classifier on training set: 0.93460
```

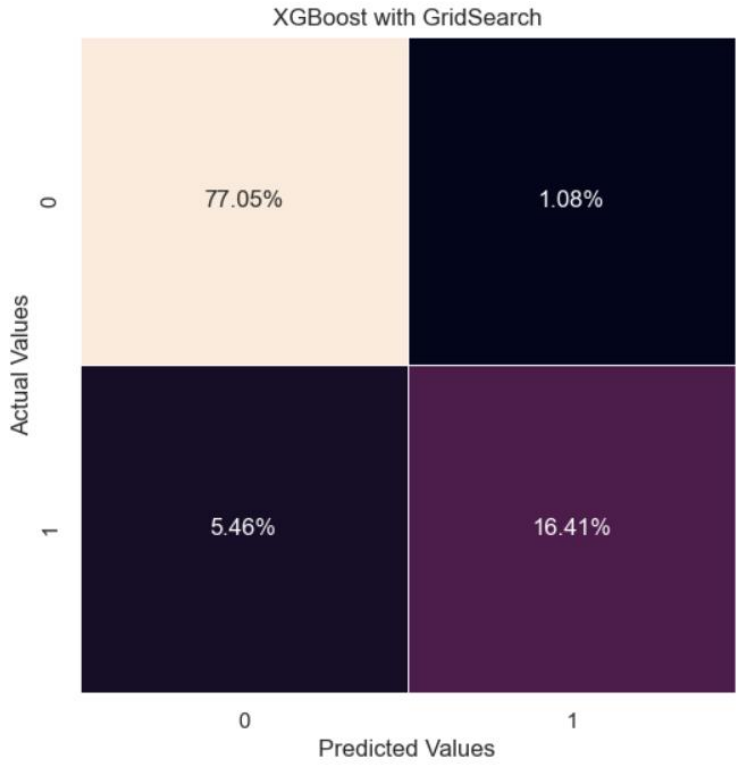
Sau khi kiểm tra ta thấy GridSearch không hiệu quả. Ta chọn ngẫu nhiên max_depth vẫn cho điểm cao hơn (0.9346 > 0.93224)

Trực quan hóa kết quả và phân loại_report:

```
y_pred = xgb_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[7493,  105],
       [ 531, 1596]], dtype=int64)
```

```
sns.heatmap(cm/np.sum(cm), annot=True, fmt='.2%', linewidth = 0.5,
            square = True, cbar = False)
plt.title('XGBoost with GridSearch')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```




```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.99      0.96      7598
           1       0.94      0.75      0.83      2127

    accuracy                           0.93      9725
   macro avg       0.94      0.87      0.90      9725
weighted avg       0.93      0.93      0.93      9725
```

Cuối cùng, sau khi training ba mô hình: Logistic, Decision Tree và XGBoost với Grid Search, chúng tôi quyết định chọn XGBoost với score của tệp train là 95% và 93% khi áp dụng cho tệp test.

Ta lưu mô hình lại

```
import pickle

path = "D:\Data science\Python\DA Python quest PC"
filename = "credit_analysis_xgbmodel.p"
pickle.dump(xgb_model, open(path+filename, 'wb'))
```

```
#Load model
model = pickle.load(open(path+filename, 'rb'))
```

```
model
```

6. PLOT ROC CURVE

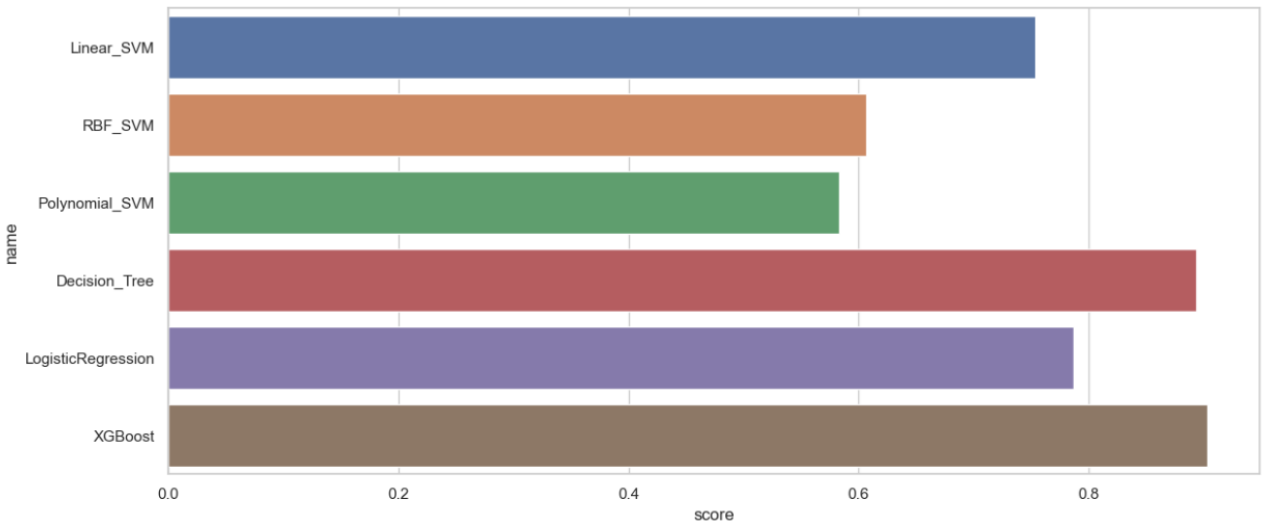
Vẽ đường cong roc cho các mô hình này

Tiến hành làm mẫu và tính điểm

```
names = ["Linear_SVM", "RBF_SVM","Polynomial_SVM", "Decision_Tree", "LogisticRegression", "XGBoost"]

classifiers = [
    SVC(kernel="linear", C=0.025, probability=True),
    SVC(kernel="rbf", C=1, gamma=2, probability=True),
    SVC(kernel="poly", degree=3, C=0.025, probability=True),
    DecisionTreeClassifier(max_depth=8),
    LogisticRegression(),
    xgb.XGBClassifier(max_depth=8, learning_rate=0.1 ,n_jobs=-1)]
```

```
scores = []
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    scores.append(score)
```



Vẽ đường cong roc bằng cách tính xác suất chính xác bằng hàm predict_proba

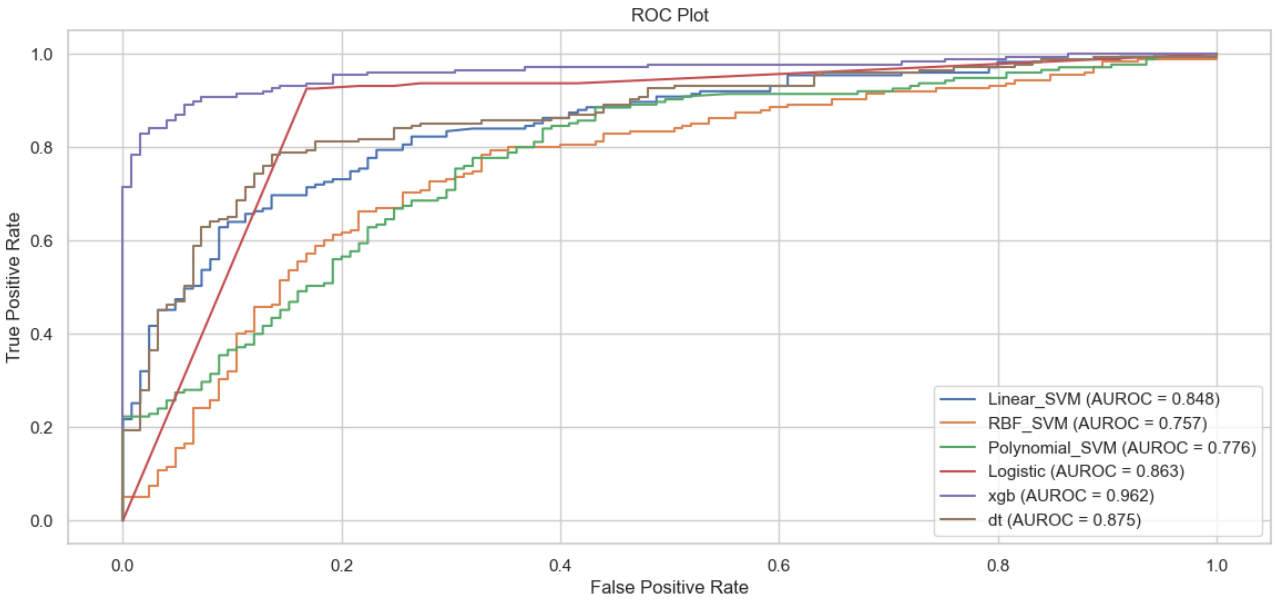
```
SVC_model_lr = SVC(kernel="linear", C=0.025, probability=True).fit(X_train, y_train)
SVC_model_rbf =  SVC(kernel="rbf", C=1, gamma=2, probability=True).fit(X_train, y_train)
SVC_model_pl = SVC(kernel="poly", degree=3, C=0.025, probability=True).fit(X_train, y_train)
dt_model = DecisionTreeClassifier(max_depth=8).fit(X_train, y_train)
log_model = LogisticRegression().fit(X_train, y_train)
xgb_model = xgb.XGBClassifier(max_depth=5, learning_rate=0.1 ,n_jobs=-1).fit(X_train, y_train)

svm_probs_lr = SVC_model_lr.predict_proba(X_test)
svm_probs_rbf = SVC_model_rbf.predict_proba(X_test)
svm_probs_pl = SVC_model_pl.predict_proba(X_test)
dt_probs = dt_model.predict_proba(X_test)
log_probs = log_model.predict_proba(X_test)
xgb_probs = xgb_model.predict_proba(X_test)
```

Ta tính điểm roc_auc_score của từng mô hình

```
Linear_SVM: AUROC = 0.848
RBF_SVM: AUROC = 0.757
Polynomial_SVM: AUROC = 0.776
Decision Tree: AUROC = 0.875
Logistics: AUROC = 0.863
XGBoost: AUROC = 0.962
```

Biểu diễn đường ROC của mỗi mô hình:



Trong sáu mô hình được thực hiện, chúng ta quyết định chọn mô hình XGBoost có AUROC = 0.962 cao nhất.

7. DEVELOP PROJECT WITH THRESHOLDING

Thresholding là kỹ thuật để chọn threshold tốt nhất, với bài toán này, chúng ta sẽ trade-off 1 trong 2 lựa chọn sau:

Higher Precision and Lower Recall

- Khi threshold đặt mức ngưỡng cao để đánh dấu default, nó sẽ đánh dấu các khách hàng là default chỉ khi nó rất chắc chắn về tính default của chúng. Điều này dẫn đến một tỷ lệ cao các khách hàng được đánh dấu là default đúng (true positives) trong số tất cả các khách hàng được đánh dấu là default (bao gồm cả true positives và false positives), nghĩa là chính xác cao. Tuy nhiên, hệ thống có thể bỏ sót một số khách hàng default (false negatives). **An toàn là trên hết**

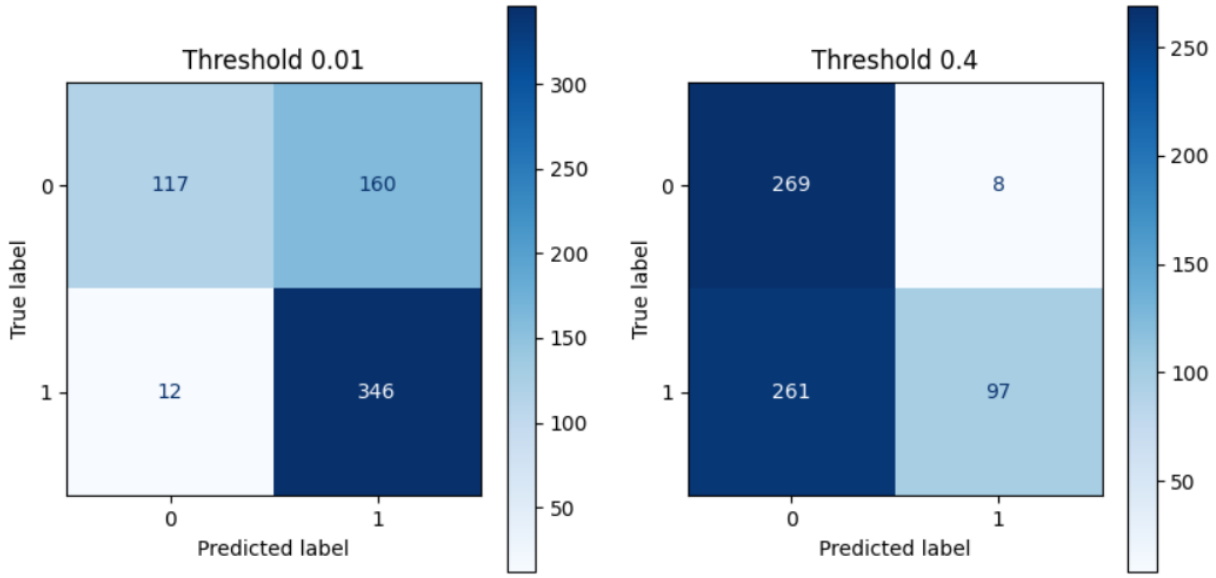
Higher Recall and Lower Precision

- Khi threshold đặt mức ngưỡng thấp, nó có khả năng phát hiện nhiều khách hàng default, bao gồm cả những khách hàng có khả năng là default nhỏ. Model có khả năng phát hiện được nhiều khách hàng default, giảm nguy cơ bỏ sót khách hàng default (false negatives). Tuy nhiên, do hệ thống đánh dấu các khách hàng có khả năng là default nhỏ, có thể dẫn đến một số lượng khách hàng không default bị đánh dấu là default sai (false positives). **Thà bắt nhầm còn hơn bỏ sót**

Tuy nhiên bài toán này rất quan trọng việc nhầm lẫn KH xấu thành tốt (false negative) nên chúng ta sẽ ưu tiên giảm false negative xuống thấp nhất có thể hay là chọn High Recall and Lower Precision xuống. Ví dụ:

```
plt.style.use('default')
fig, axs = plt.subplots(1,2, figsize=(10, 5))

disp = metrics.ConfusionMatrixDisplay.from_predictions(y_test, y_pred_thres_1pct,cmap=plt.cm.Blues,ax=axs[0])
disp = metrics.ConfusionMatrixDisplay.from_predictions(y_test, y_pred_thres_40pct,cmap=plt.cm.Blues,ax=axs[1])
axs[0].set_title('Threshold 0.01')
axs[1].set_title('Threshold 0.4')
plt.show()
```



Với threshold = 0.01, false negative chỉ có 12 khách hàng, trong khi con số này là 261 khách hàng với threshold = 0.4 hay default threshold = 0.5. Nhưng đổi lại số khách hàng bị nhầm là default lên đến 160 khách hàng so với 8 người.

Ta vẽ bảng so sánh precision và recall giữa 2 mức threshold.

metric	tp	fp	tp+fp	precision	recall
threshold					
0.01	346	160	506	0.683794	0.96648
0.40	97	8	105	0.923810	0.27095

Ta có thể thấy trade-off giữa precision và recall khi chọn threshold, và đây là tiền đề để làm các bài toán scoring sâu hơn.

Việc ưu tiên giảm False Negative là quan trọng trong bài toán này tương đương với việc giảm threshold thấp hơn (và cũng đồng thời giảm Precision xuống)

Lúc này chúng ta sẽ giải thích cho business stake-holder cặn kẽ và hiểu xem họ mong muốn gì để điều chỉnh precision và recall cho phù hợp với yêu cầu của business.

Tham khảo file gốc tại đây: (Credit Scoring)