

# Credit Risk Assessment Using Machine Learning Techniques

Credit is a must in financial systems. For all financial institutions, whose role is to allocate credit, it is necessary to fully understand the risk behind it and to correctly decide who to give credit and who not. To do so, they make use of credit scoring, which is one of the most successful application of statistical and operational research modelling in finance.

The aim of this thesis is to combine supervised and unsupervised machine learning models to predict the probability of default of a set of individuals who asked a loan to a bank, and to correctly classify them according to their individual propensity to default.

## Data Set Description

A significant problem for credit scoring models which must be pointed out is the unavailability of real-world credit data. The reason is that customer's credit data is confidential in most of the financial institutions. For these reasons we will use a public data set from Kaggle.

Kaggle is an online community of data scientists and machine learning practitioners that offers machine learning competitions and a public data platform.

The data set contains information on clients who have taken out a loan with an unspecified financial institution. The aims of the analysis is to search for statistical relationships that could give us some insights about the the risk of credit and to develop some algorithms to predict credit default. Given that the context of the data set is not fully explained, we are not going to take into account external macroeconomic events, which could completely change the results of the analysis. For instance, different countries could have different loan's requirement, or different phases of the economic cycle could end up in a very different scenario.

Another important fact to which one must draw attention is that we will assume the data set to be not distorted by a credit score system. This means that no systematic screening of the costumers' credit standing had been implemented until the date of data retrieval.

The data set contains 12 attributes and 32581 observations, and its structure is resumed in Table below. The potential covariates describe both customers and loans characteristics. As we can see there are both numerical and categorical variables. Let us see a little more in details which are the levels of the categorical ones.

VARIABLE	DESCRIPTION	TYPE
<i>person_age</i>	Customers age (in years)	numerical
<i>person_income</i>	Annual income (in dollars)	numerical
<i>person_home_ownership</i>	Home ownership	categorical
<i>person_emp_length</i>	Employment length (in years)	numerical
<i>loan_intent</i>	Loan intent	categorical
<i>loan_grade</i>	Loan grade	categorical
<i>loan_amnt</i>	Loan amount (in dollars)	numerical
<i>loan_int_rate</i>	Loan interest rate	numerical

## 1. Import library

In [ ]:

```
# Import scikit-plot
import scikitplot as skplt
import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
import pandas as pd
import numpy as np
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
from matplotlib import rcParams
from optbinning import OptimalBinning

from sklearn.model_selection import train_test_split, learning_curve, RandomizedSearchCV
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report
```

In [118]:

```
df = pd.read_csv('credit_risk_dataset.csv')
df.head()
```

Out[118]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_gr
0	22	59000		RENT	123.0	PERSONAL
1	21	9600		OWN	5.0	EDUCATION
2	25	9600	MORTGAGE		1.0	MEDICAL
3	23	65500		RENT	4.0	MEDICAL
4	24	54400		RENT	8.0	MEDICAL

In [119]:

```
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
```

## 2. Dataset information

In [120]:

```
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
```

In [121]:

```
df.describe()
```

Out[121]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status
count	32581.000000	3.258100e+04	31686.000000	32581.000000	29465.000000	32581.000000
mean	27.734600	6.607485e+04	4.789686	9589.371106	11.011695	0.218162
std	6.348078	6.198312e+04	4.142630	6322.086646	3.240459	0.413006
min	20.000000	4.000000e+03	0.000000	500.000000	5.420000	0.000000
25%	23.000000	3.850000e+04	2.000000	5000.000000	7.900000	0.000000
50%	26.000000	5.500000e+04	4.000000	8000.000000	10.990000	0.000000
75%	30.000000	7.920000e+04	7.000000	12200.000000	13.470000	0.000000
max	144.000000	6.000000e+06	123.000000	35000.000000	23.220000	1.000000

### 3. Exploratory data analysis

#### 3.1 Explore data

In [122]:

```
def plot_distributions_int64(feature):
    _, axes = plt.subplots(1, 2, figsize=(12,4))
    sns.histplot(df[feature], kde=True, label='Train', stat='density',
                 discrete=True, color='pink', alpha=0.6, ax=axes[0])
    axes[0].legend()
    axes[0].set_title('Distribution')

    sns.boxplot(x='loan_status', y=feature, data=df, ax=axes[1], palette=['seagreen', 'tan'])
```

In [123]:

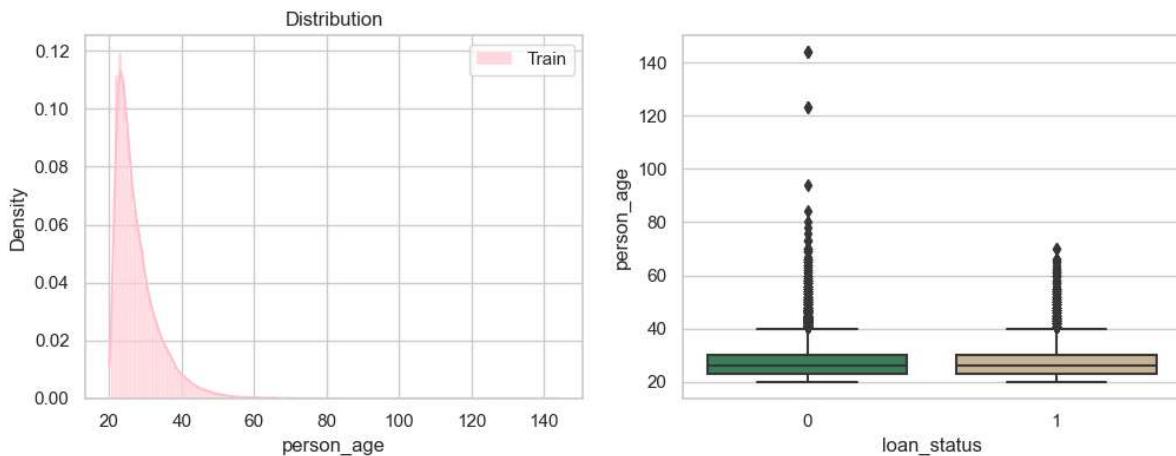
```
def plot_distributions_float(feature):

    _, axes = plt.subplots(1, 2, figsize=(12,4))
    sns.kdeplot(df[feature].apply(np.log1p), label='Train',
                color='black', alpha=0.5, ax=axes[0])
    #, shape=True, edgecolor='k'
    axes[0].legend()
    axes[0].set_title('Distribution')
    axes[0].set_xlabel('{}'.format(feature))
    sns.boxplot(x='loan_status', y=df[feature],
                data=df, ax=axes[1], palette=['seagreen', 'tan'])
```

In [124]:

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

"()" distributions



In [125]:

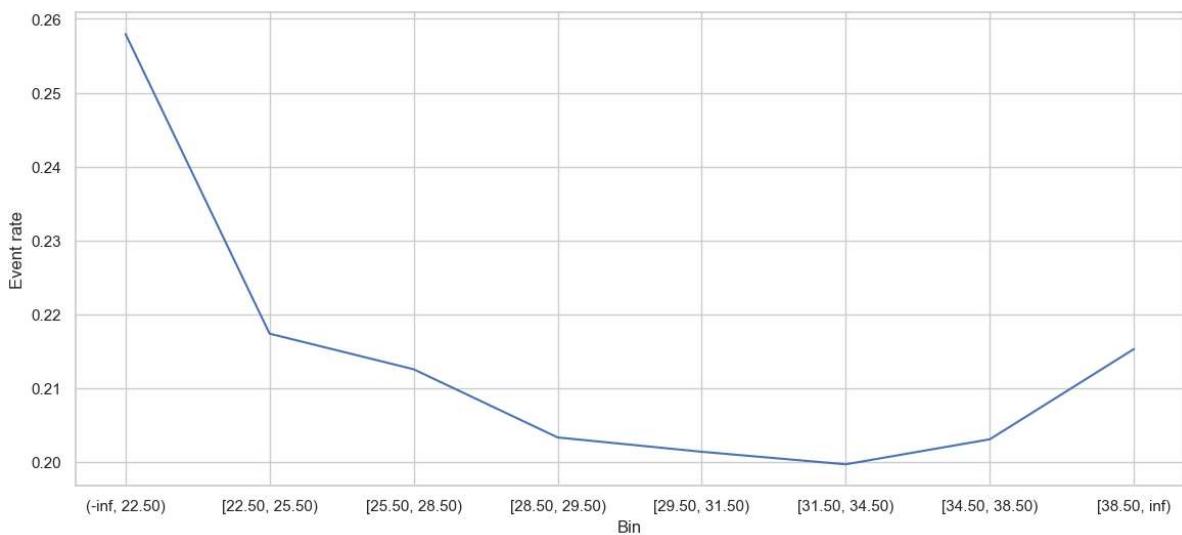
```
x = df['person_age'].values
y = df['loan_status']
#Look model for bin in next variable loan_int_rate
optb3= OptimalBinning(name='Age', dtype="numerical", solver ='cp')
optb3.fit(x, y)
age = optb3.binning_table.build()
age_bin = age[['Bin', 'Count', "Non-event", 'Event' , 'Event rate']] .loc[0:8]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

Out[125]:

	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

In [126]:

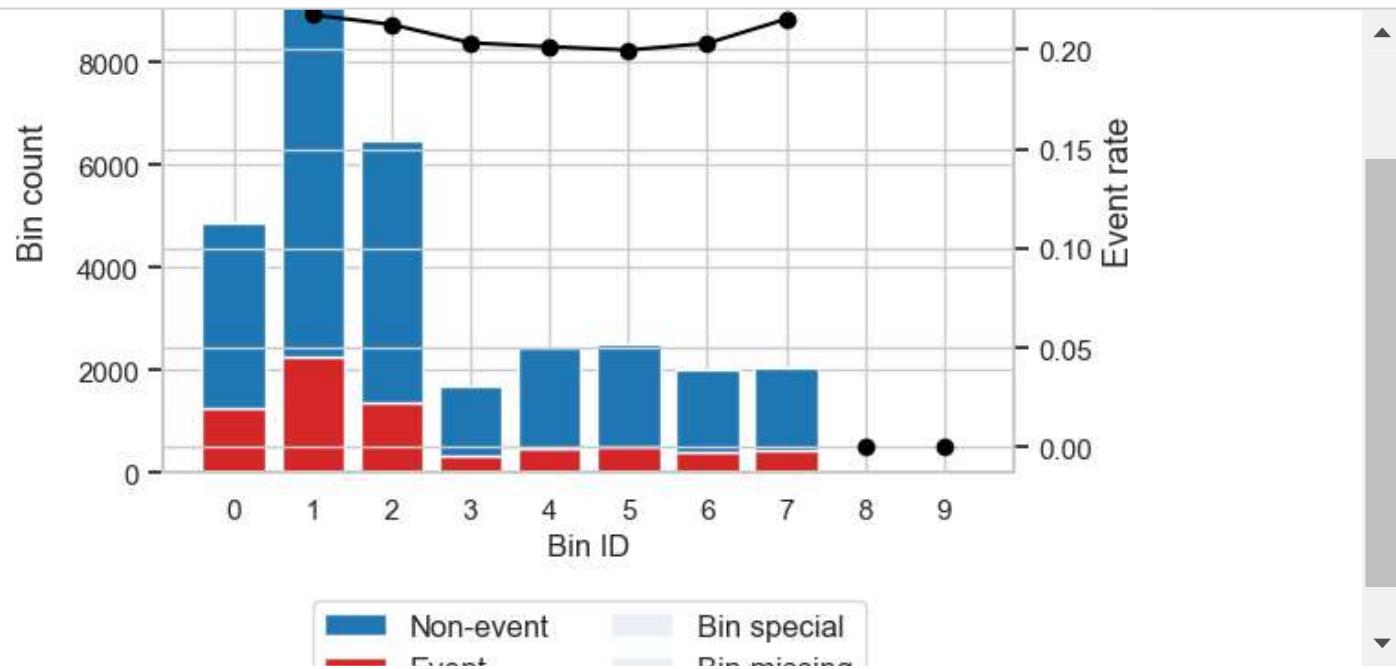
```
rcParams['figure.figsize'] = 14,6  
sns.lineplot(data=age_bin.loc[0:7], x='Bin', y='Event rate');
```



For person\_age, the loan\_status is negatively correlated with this indicator (age): it increases along the increase in this ratio. However, after 34.5 yrs, the trend is converse, this indicator is not really important.

In [127]:

```
rcParams['figure.figsize'] = 6,4  
optb3.binning_table.plot(metric='event_rate')
```

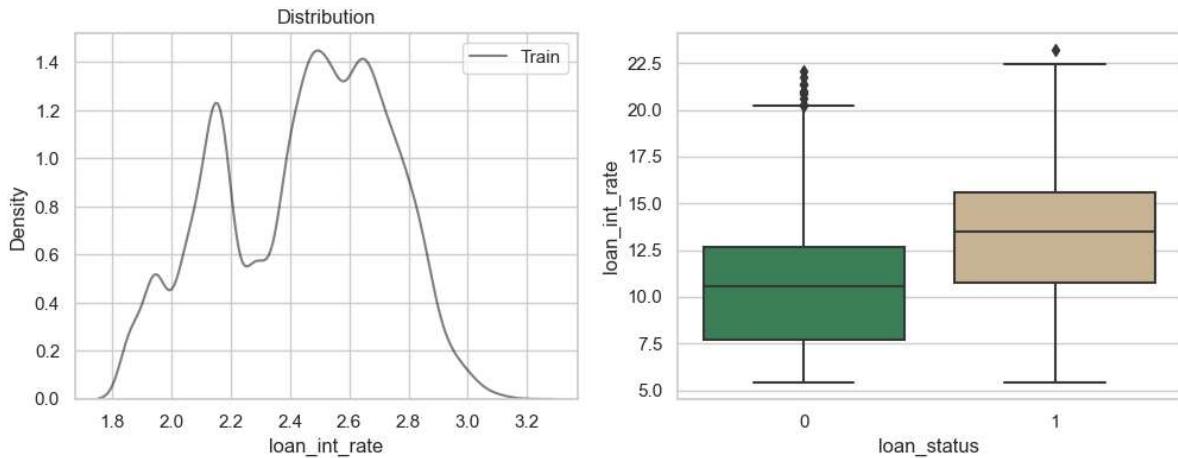


Float variable

In [128]:

```
plot_distributions_float('loan_int_rate')
```

"()" distributions



In [129]:

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

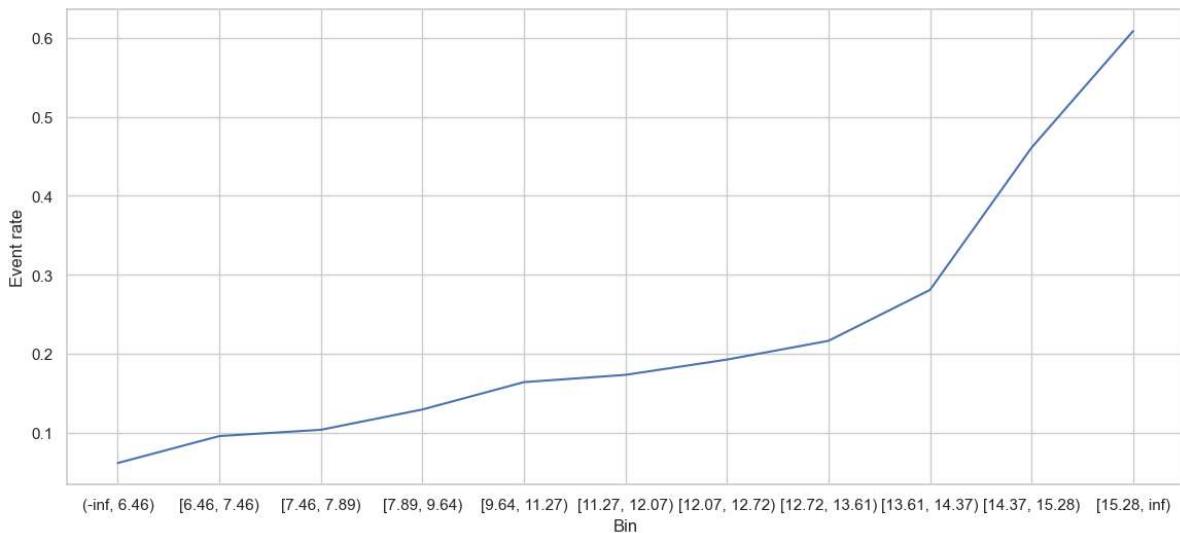
optb3 = OptimalBinning(name='Interest', dtype="numerical", solver ='cp')
optb3.fit(x, y)
var = optb3.binning_table.build()
var_bin = var[['Bin', 'Count', "Non-event", 'Event' , 'Event rate']] .loc[0:12]
var_bin.loc[0:12] # don't show Special and Missing
```

Out[129]:

	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

In [130]:

```
#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');
```



The higher Interest rate is, the higher default rate is.

## Categorical Variable

In [131]:

```
# df_geo = df[df['loan_status']==1].groupby(by=['person_home_ownership']).agg('count')

var = 'loan_status' #except feature(below), all variance is True
def df_distribution_categorical(feature):
    df_feature = df.groupby(by=[feature]).agg('count')
```

In [132]:

```
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

**we continue to describe dataframe for every catalog in variable**

In [17]:

```
df_m = df[df['person_home_ownership']=='MORTGAGE']
df_m.describe()
```

Out[17]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status
count	13444.000000	1.344400e+04	13093.000000	13444.000000	12122.000000	13444.000000
mean	27.980735	8.112712e+04	5.883907	10574.460726	10.488001	0.125707
std	6.295921	7.876525e+04	4.383934	6789.579286	3.298739	0.331537
min	20.000000	7.904000e+03	0.000000	500.000000	5.420000	0.000000
25%	24.000000	5.000000e+04	3.000000	5400.000000	7.510000	0.000000
50%	26.000000	6.900000e+04	5.000000	9200.000000	10.380000	0.000000
75%	30.000000	9.600000e+04	8.000000	14756.250000	12.965000	0.000000
max	144.000000	6.000000e+06	123.000000	35000.000000	22.480000	1.000000

In [18]:

```
df_o = df[df['person_home_ownership']=='OTHER']
df_o.describe()
```

Out[18]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	lo
count	107.000000	107.000000	107.000000	107.000000	94.000000	107.000000	107.000000
mean	26.803738	76387.803738	3.682243	11074.532710	12.030638	0.308411	0.308411
std	6.366835	82204.071525	3.869567	6012.921406	2.624237	0.464011	0.464011
min	21.000000	11389.000000	0.000000	1000.000000	7.400000	0.000000	0.000000
25%	23.000000	40052.000000	1.000000	6000.000000	9.630000	0.000000	0.000000
50%	24.000000	59000.000000	3.000000	10000.000000	11.890000	0.000000	0.000000
75%	28.000000	82328.000000	5.500000	15000.000000	14.260000	1.000000	1.000000
max	48.000000	600000.000000	24.000000	25000.000000	17.900000	1.000000	1.000000

In [19]:

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

Out[19]:

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

In [20]:

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

Out[20]:

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

Age and income are not too different the loan\_status is clear with the highest rate belong to RENT.

## Historical default

In [21]:

```
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

In [22]:

```
df_h = df[df['cb_person_default_on_file']=='Y']
df_h.describe()
```

Out[22]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status
count	5745.000000	5.745000e+03	5629.000000	5745.000000	5213.000000	5745.000000
mean	27.814273	6.559078e+04	4.542548	10123.359443	14.513800	0.378068
std	6.261130	5.710100e+04	4.256252	6853.530975	1.727252	0.484947
min	20.000000	4.000000e+03	0.000000	750.000000	10.280000	0.000000
25%	23.000000	3.600000e+04	2.000000	5000.000000	13.230000	0.000000
50%	26.000000	5.400000e+04	4.000000	8400.000000	14.260000	0.000000
75%	30.000000	7.800000e+04	7.000000	14400.000000	15.620000	1.000000
max	70.000000	2.039784e+06	123.000000	35000.000000	23.220000	1.000000

In [23]:

```
df_h = df[df['cb_person_default_on_file']=='N']
df_h.describe()
```

Out[23]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status
count	26836.000000	2.683600e+04	26057.000000	26836.000000	24252.000000	26836.000000
mean	27.717544	6.617848e+04	4.843075	9475.055895	10.258913	0.183932
std	6.366524	6.297963e+04	4.115808	6196.560537	2.985542	0.387436
min	20.000000	4.080000e+03	0.000000	500.000000	5.420000	0.000000
25%	23.000000	3.900000e+04	2.000000	5000.000000	7.660000	0.000000
50%	26.000000	5.520000e+04	4.000000	8000.000000	10.360000	0.000000
75%	30.000000	8.000000e+04	7.000000	12000.000000	12.180000	0.000000
max	144.000000	6.000000e+06	123.000000	35000.000000	21.740000	1.000000

The people who has positive historical default have double higher rate than negative one.

## 3.2. Missing value

### Drop duplicates

In [24]:

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

In [25]:

```
df[dups]
```

Out[25]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan_gr
15975	23	42000	RENT	5.0	VENTURE	
15989	23	90000	MORTGAGE	7.0	EDUCATION	
15995	24	48000	MORTGAGE	4.0	MEDICAL	
16025	24	10000	RENT	8.0	PERSONAL	
16028	23	100000	MORTGAGE	7.0	EDUCATION	
...	...	...	...	...	...	...
32010	42	39996	MORTGAGE	2.0	HOMEIMPROVEMENT	
32047	36	250000	RENT	2.0	DEBTCONSOLIDATION	
32172	49	120000	MORTGAGE	12.0	MEDICAL	
32259	39	40000	OWN	4.0	VENTURE	

In [26]:

```
#test duplicates
df.query("person_age==23 & person_income==42000 &
person_home_ownership=='RENT' & loan_int_rate==9.99")
```

Out[26]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_intent	loan
6464	23	42000	RENT	5.0	VENTURE	
15975	23	42000	RENT	5.0	VENTURE	

In [27]:

```
df.shape
```

Out[27]:

```
(32581, 12)
```

In [28]:

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

In [29]:

```
df.shape
```

Out[29]:

```
(32416, 12)
```

## Handling NaN values

In [30]:

```
df.isna().sum()
```

Out[30]:

```
person_age           0  
person_income        0  
person_home_ownership 0  
person_emp_length    887  
loan_intent          0  
loan_grade           0  
loan_amnt            0  
loan_int_rate        3095  
loan_status          0  
loan_percent_income   0  
cb_person_default_on_file 0  
cb_person_cred_hist_length 0  
dtype: int64
```

In [31]:

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

Out[31]:

```
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
```

In [32]:

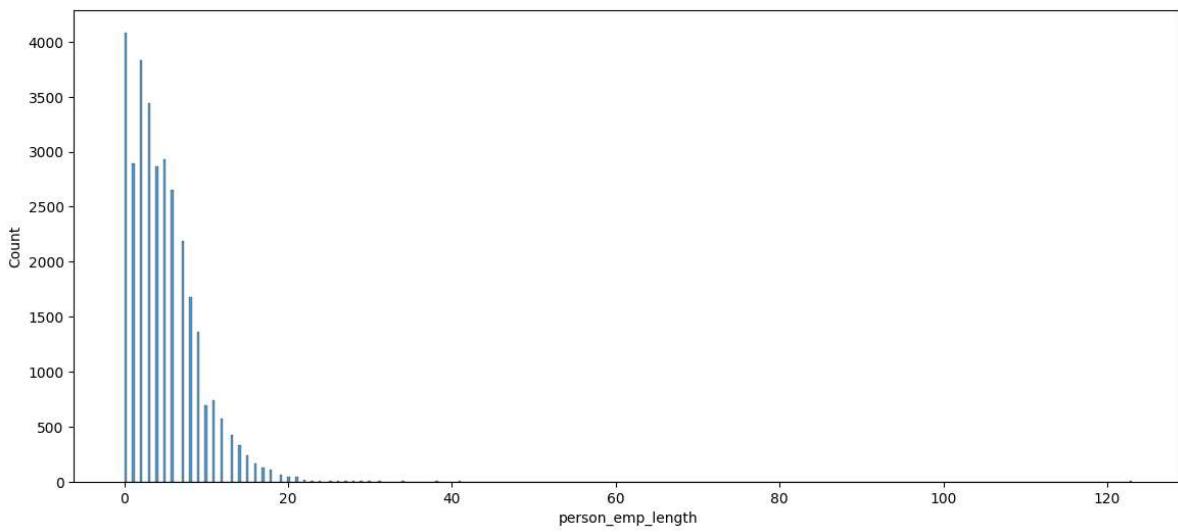
```
#Should we impute mean, mode, or median?  
df['person_emp_length'].describe()
```

Out[32]:

```
count    31529.00000  
mean      4.79051  
std       4.14549  
min      0.00000  
25%     2.00000  
50%     4.00000  
75%     7.00000  
max     123.00000  
Name: person_emp_length, dtype: float64
```

In [33]:

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



In [34]:

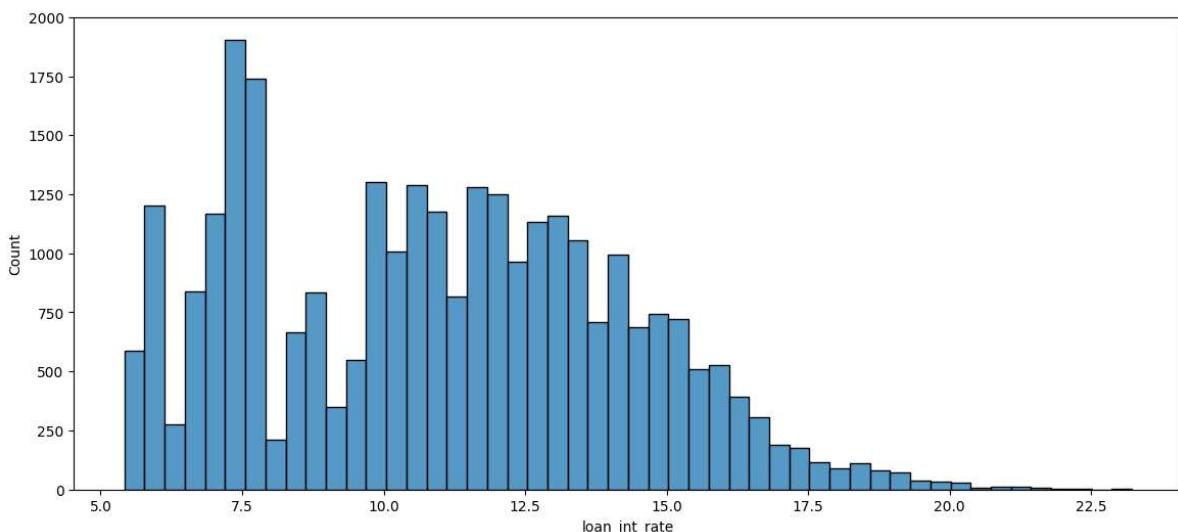
```
df['loan_int_rate'].describe()
```

Out[34]:

```
count    29321.000000  
mean      11.017265  
std       3.241680  
min       5.420000  
25%      7.900000  
50%     10.990000  
75%     13.470000  
max     23.220000  
Name: loan_int_rate, dtype: float64
```

In [35]:

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



The distributions of 2 variables person\_emp\_length and loan\_int\_rate are skewed, have outliers. Therefore, median imputation is preferred. However,

In [36]:

```
#df.fillna(df.median(), inplace=True) for numeric dataset  
#df.loan_int_rate.fillna(df.loan_int_rate.mode()[0], inplace=True) #if mode() add[0]  
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()
```

Out[36]:

```
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
```

Because loan\_percent\_income = loan\_amnt/person\_income, we drop loan\_percent\_income column.

In [37]:

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

Out[37]:

	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

In [38]:

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

### 3.3 Detecting and handling outliers

#### Outliers

Because we have already performed boxplots which contain outliers of numeric variable on Exploration data above, we only define function to find whisker and handle outliers in this step.

In [39]:

```
df_copy = df.copy()
```

In [133]:

```
#Find outliers:

def detect_outlier(col):

# first quantile/percentile
    first_qrt = np.quantile(df_copy[col], 0.25)
# third quantile/percentile
    third_qrt = np.quantile(df_copy[col], 0.75)

    lower_whisker = first_qrt - (third_qrt -first_qrt )*1.5
    print("lower_whisker:", lower_whisker)
    upper_whisker = third_qrt + (third_qrt -first_qrt )*1.5
    print("upper whisker:", upper_whisker)
```

In [134]:

```
#plot boxplot and find the upper and Lower whisker to prepare to handle outliers by using scipy.stats
def boxplot(col):
    plt.figure(figsize=(8,5))
    sns.boxplot(y= col, data = df_copy)
```

In [135]:

```
#handle outliers by using scipy.stats, not delete them
import scipy.stats

df_cleaned = df_copy.copy()
def new_df(col, lower_lim, upper_lim):
    df_cleaned[col] = scipy.stats.mstats.winsorize(df_cleaned[col], limits = [lower_lim, upper_lim])
```

### Check with loan\_int\_rate

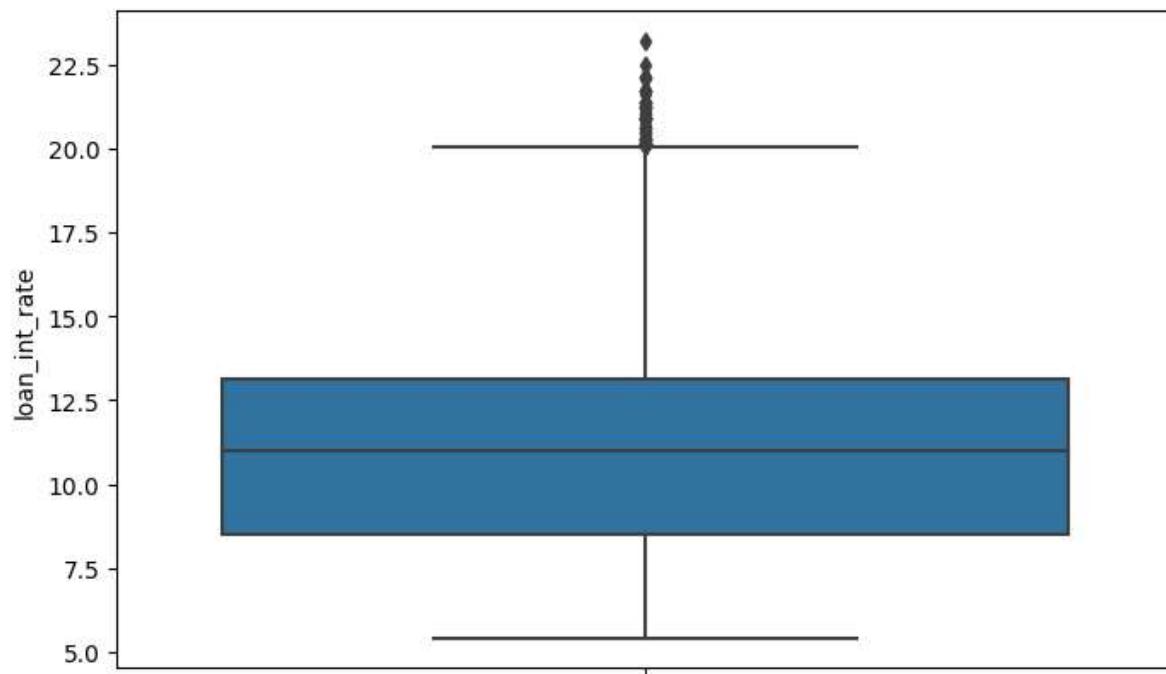
In [43]:

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

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

In [44]:

```
#plot boxplot and find the upper and Lower whisker to prepare to handle outliers by using scipy.stats  
boxplot('loan_int_rate')
```

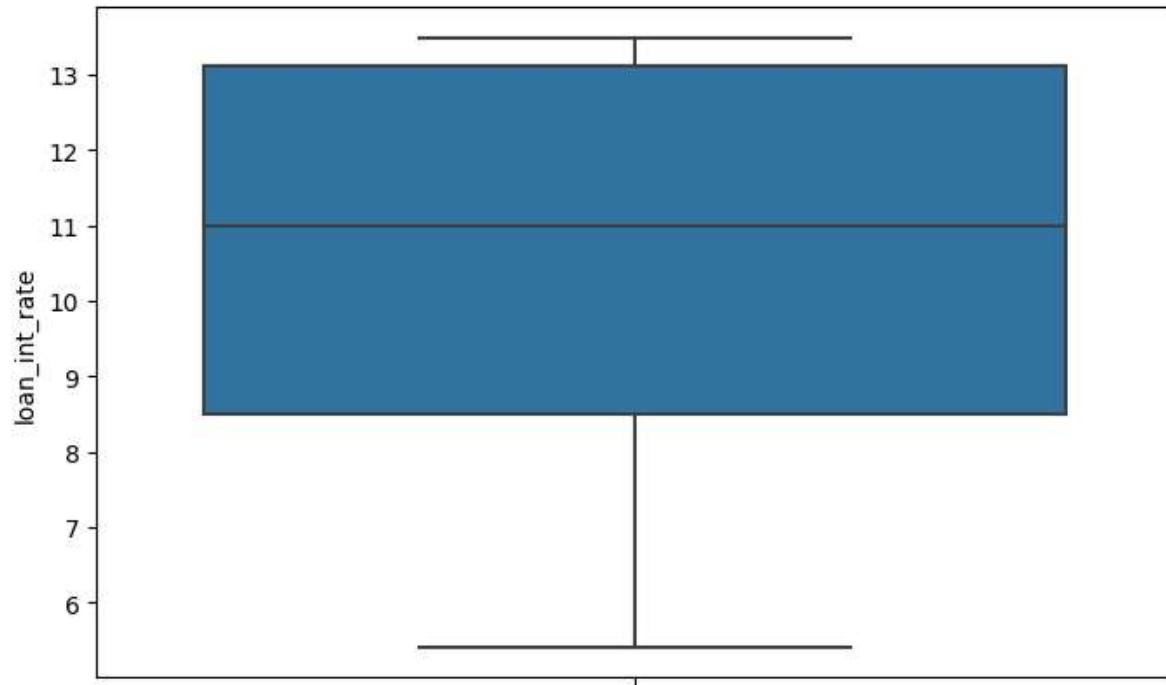


In [45]:

```
#handle outliers by using scipy.stats  
new_df('loan_int_rate',0, 0.22)
```

In [46]:

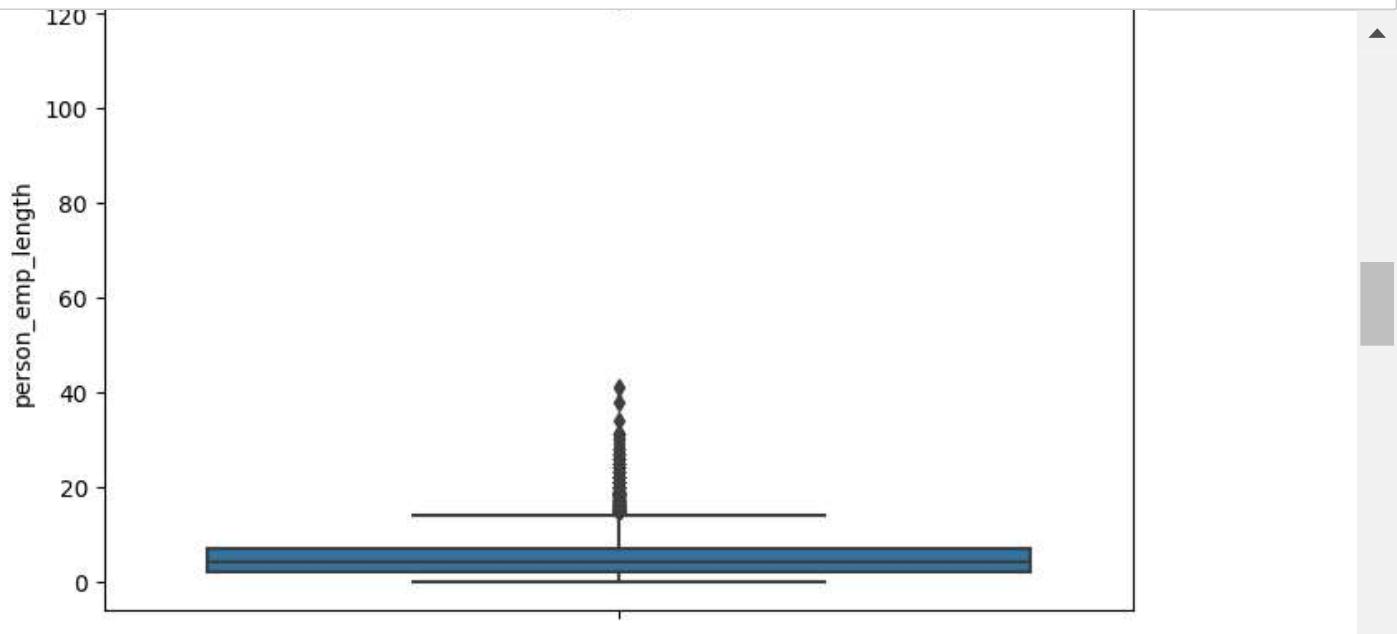
```
boxplot(df_cleaned['loan_int_rate']) #df_cleaned not df
```



We do these for all numeric columns.

In [47]:

```
col =['person_age', 'person_income', 'person_emp_length', 'loan_amnt',
      'loan_int_rate','cb_person_cred_hist_length']
for c in col:
    plt.figure(figsize=(2,2))
    boxplot(c)  #this is function we define: def boxplot(col)
```



In [48]:

```
col =['person_age', 'person_income', 'person_emp_length', 'loan_amnt',
      'loan_int_rate','loan_status','cb_person_cred_hist_length']
detect_outlier('person_age')
print('\n')
detect_outlier('person_income')
print('\n')
detect_outlier('person_emp_length')
print('\n')
detect_outlier('loan_amnt')
print('\n')
detect_outlier('loan_int_rate')
print('\n')
detect_outlier('cb_person_cred_hist_length')
```

```
lower_whisker: 12.5
upper whisker: 40.5
Number of outliers: 1491
% outliers: 4.599580454096742 %
```

```
lower_whisker: -22472.0
upper whisker: 140232.0
Number of outliers: 1478
% outliers: 4.559476801579467 %
```

```
lower_whisker: -5.5
upper whisker: 14.5
Number of outliers: 852
% outliers: 2.6283316880552814 %
```

```
lower_whisker: -5875.0
upper whisker: 22125.0
```

We can see at Person\_age: whisker: 12.5-40.5 (not min=12.5 and max=40.5), % outliers= 4.599%. Looking to boxplot distribution at above code, outliers of person\_age is on the upper part, so are all variables, therefore, we set lower\_lim=0, upper\_lim=0.046. With loan\_int\_rate, the level 20% is realistic, make sense in life, so we don't delete these outliers.

In [49]:

```
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)
```

In [50]:

```
df_cleaned
```

Out[50]:

	person_age	person_income	person_home_ownership	person_emp_length	loan_in
0	22	59000	RENT	14.0	PERSO
1	21	9600	OWN	5.0	EDUCAT
2	25	9600	MORTGAGE	1.0	MEDI
3	23	65500	RENT	4.0	MEDI
4	24	54400	RENT	8.0	MEDI
...	...	...	...	...	...
32576	40	53000	MORTGAGE	1.0	PERSO
32577	40	120000	MORTGAGE	4.0	PERSO
32578	40	76000	RENT	3.0	HOMEIMPROVEM
32579	40	140000	MORTGAGE	5.0	PERSO
32580	40	42000	RENT	2.0	MEDI

32416 rows × 11 columns

The shape of dataframe and unique values of variables are unchanged.

In [51]:

```
for col in df_cleaned.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  
cb_person_default_on_file : 2  
cb_person_cred_hist_length : 29
```

## Convert categorical variable to numeric type

In [52]:

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

In [53]:

```
df_cleaned_dum
```

Out[53]:

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

## 4. Modelling

In [62]:

```
# X and y will be thought of as the entire data
X = df_cleaned_dum.drop('loan_status', axis=1).copy()
y = df_cleaned_dum['loan_status']
#y = y[X.index]
X.shape, y.shape
```

Out[62]:

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

In [63]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.3,
                                                    stratify=df['loan_status'], shuffle=True)
```

In [64]:

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

In [65]:

```
y_test.value_counts(normalize=True) # independent variable is quite imbalance, we can use s...
```

Out[65]:

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

In [60]:

```
y = y[X.index]  
X = X[:1000]  
y = y[:1000]
```

## 4.1. LogisticRegression

In [197]:

```
logreg = LogisticRegression(random_state=0).fit(X_train, y_train)
```

In [198]:

```
# the accuracy of the model  
logreg.score(X_train, y_train)
```

Out[198]:

```
0.8539509056454101
```

In [199]:

```
logreg.score(X_test, y_test)
```

Out[199]:

```
0.856452442159383
```

In [200]:

```
# Let's check the model to predict  
logreg.predict(X_train[:20])
```

Out[200]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],  
      dtype=int64)
```

In [201]:

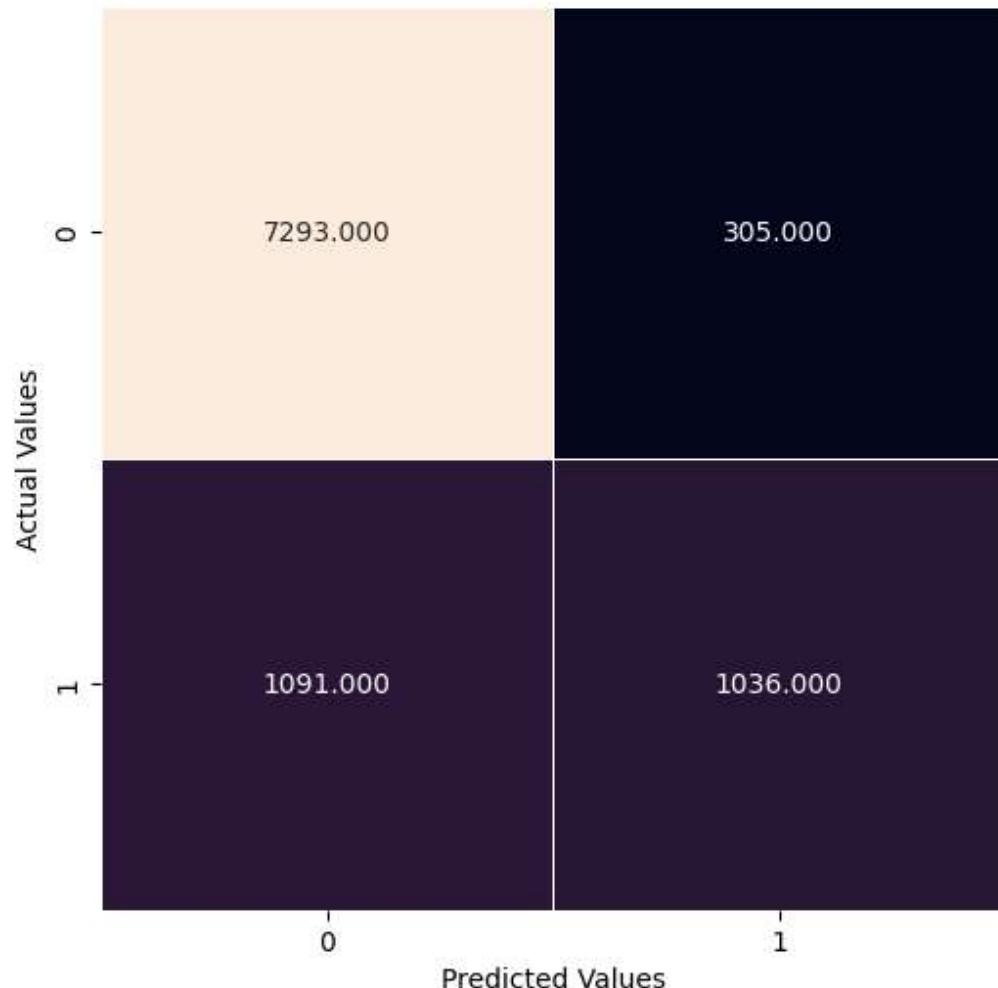
```
y_pred = logreg.predict(X_test)
```

In [202]:

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

In [203]:

```
sns.heatmap(cm, annot= True, fmt = '0.3f', linewidth = 0.5,
             square = True, cbar = False)
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```



In [204]:

```
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

In [205]:

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

Out[205]:

0.856452442159383

## 4.2. Decision Tree Classifier and GridSearchCV

In [206]:

```
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, random_state=5)
```

In [207]:

```
from sklearn.tree import DecisionTreeClassifier  
  
dt = DecisionTreeClassifier()  
dt.fit(X_train, y_train)
```

Out[207]:

DecisionTreeClassifier()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [208]:

```
dt.score(X_train, y_train)
```

Out[208]:

1.0

In [209]:

```
dt.score(X_test, y_test)
```

Out[209]:

0.8827763496143959

In [210]:

```
dt1 = DecisionTreeClassifier(max_depth=8)  
dt1.fit(X_train, y_train)
```

Out[210]:

DecisionTreeClassifier(max\_depth=8)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [211]:

```
print("training data score: " + str(dt1.score(X_train, y_train)))  
print('test data score ' + str(dt1.score(X_test, y_test)))
```

training data score: 0.9176325415362919  
test data score 0.9112596401028278

## #### Using DecisionTree model with GridSearchCV

In [212]:

```
from sklearn.model_selection import GridSearchCV
parameters = {'max_depth': [3, 4, 5, 6, 7], 'max_leaf_nodes': [2, 3, 4, 5, 6]}
dt2 = DecisionTreeClassifier()
dt2 = GridSearchCV(dt2, parameters)
```

In [213]:

```
dt2.fit(X_train, y_train)
```

Out[213]:

```
GridSearchCV(estimator=DecisionTreeClassifier(),
             param_grid={'max_depth': [3, 4, 5, 6, 7],
                          'max_leaf_nodes': [2, 3, 4, 5, 6]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [214]:

```
print("training data score: " + str(dt2.score(X_train, y_train)))
print('test data score ' + str(dt2.score(X_test, y_test)))
```

```
training data score: 0.8359702084526905
test data score 0.836401028277635
```

In [215]:

```
dt2.best_params_
```

Out[215]:

```
{'max_depth': 3, 'max_leaf_nodes': 5}
```

In [216]:

```
dt3 = DecisionTreeClassifier(max_depth=3, max_leaf_nodes=5)
dt3.fit(X_train, y_train)
```

Out[216]:

```
DecisionTreeClassifier(max_depth=3, max_leaf_nodes=5)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [217]:

```
print("training data score: " + str(dt3.score(X_train, y_train)))
print('test data score ' + str(dt3.score(X_test, y_test)))
```

```
training data score: 0.8359702084526905
test data score 0.836401028277635
```

Using GridSearchCV makes our model worse (score go down from 91.2% to 83.6%)

In [218]:

```
from sklearn.metrics import confusion_matrix
y_pred = dt3.predict(X_test)
confusion_matrix(y_test, y_pred)
```

Out[218]:

```
array([[7249,  321],
       [1270,  885]], dtype=int64)
```

In [219]:

```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
tn, fp, fn, tp
```

Out[219]:

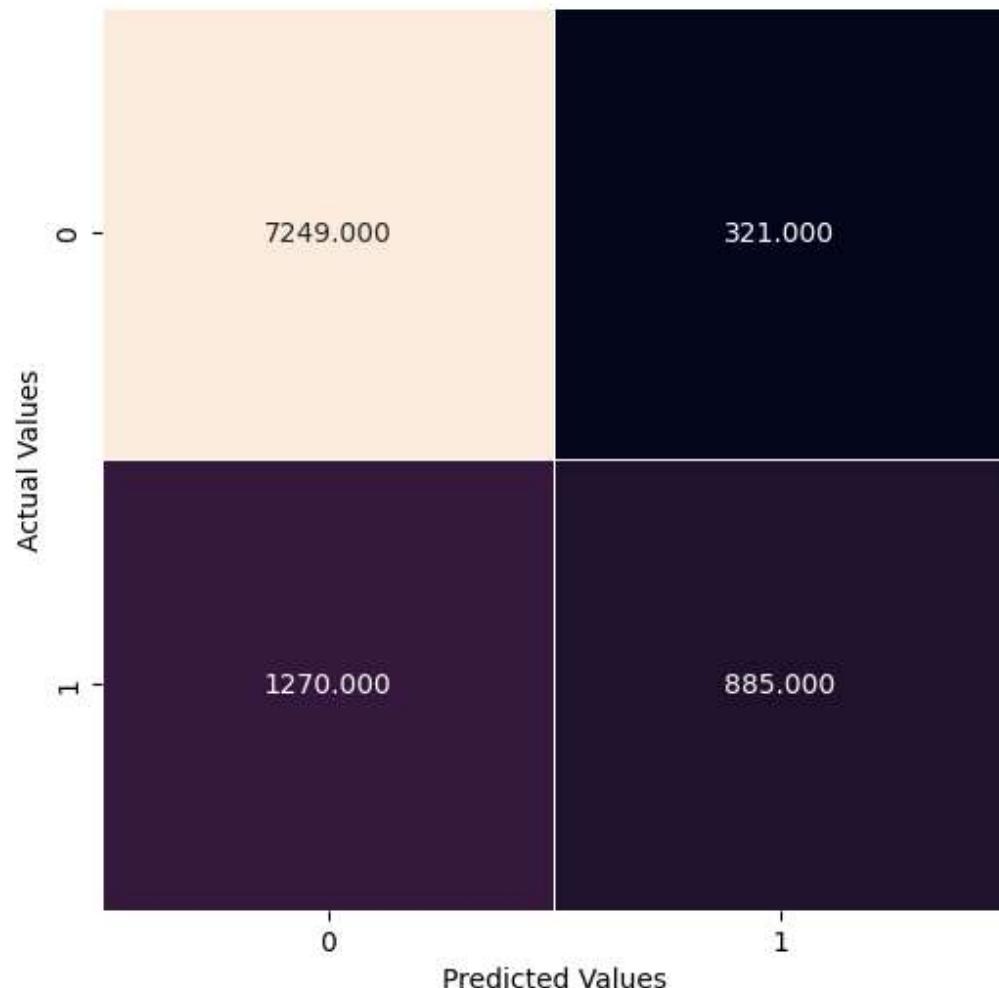
```
(7249, 321, 1270, 885)
```

In [220]:

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

In [221]:

```
sns.heatmap(cm, annot= True, fmt = '0.3f', linewidth = 0.5,
            square = True, cbar = False)
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```

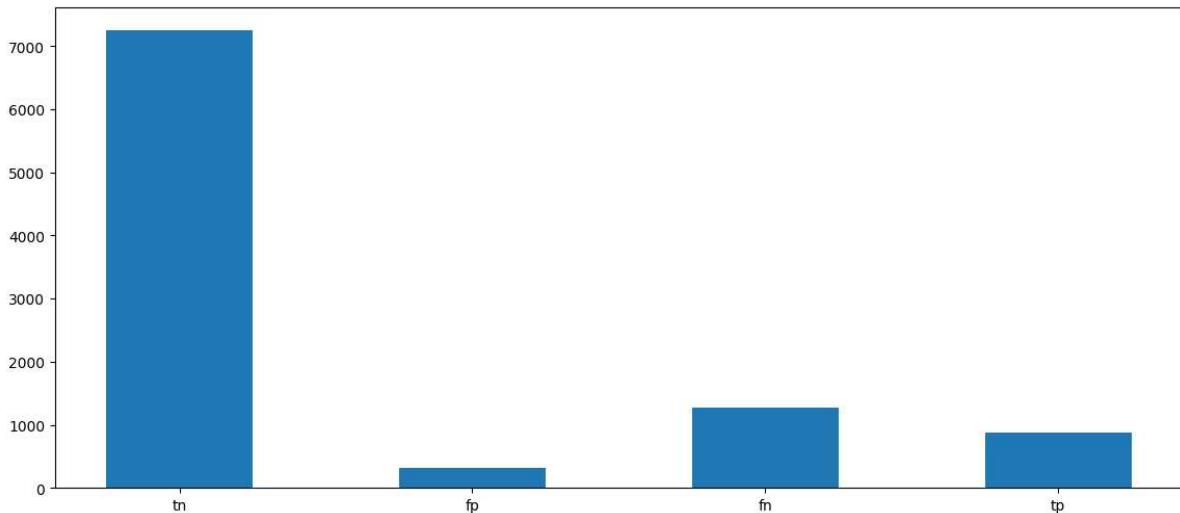


In [222]:

```
#change from array to series
cm = cm.ravel()
cm = pd.Series(cm)
```

In [223]:

```
para = ['tn', 'fp', 'fn', 'tp']
plt.bar(para, cm, width = 0.5,data=None)
plt.show()
```



In [224]:

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

	precision	recall	f1-score	support
0	0.85	0.96	0.90	7570
1	0.73	0.41	0.53	2155
accuracy			0.84	9725
macro avg	0.79	0.68	0.71	9725
weighted avg	0.82	0.84	0.82	9725

In [225]:

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

Out[225]:

0.836401028277635

**check with max\_depth=8**

In [226]:

```
dt1 = DecisionTreeClassifier(max_depth=8)
dt1.fit(X_train, y_train)
```

Out[226]:

```
DecisionTreeClassifier(max_depth=8)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [227]:

```
print("training data score: " + str(dt1.score(X_train, y_train)))
print('test data score ' + str(dt1.score(X_test, y_test)))
```

```
training data score: 0.9175884712000353
test data score 0.912185089974293
```

In [228]:

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

Out[228]:

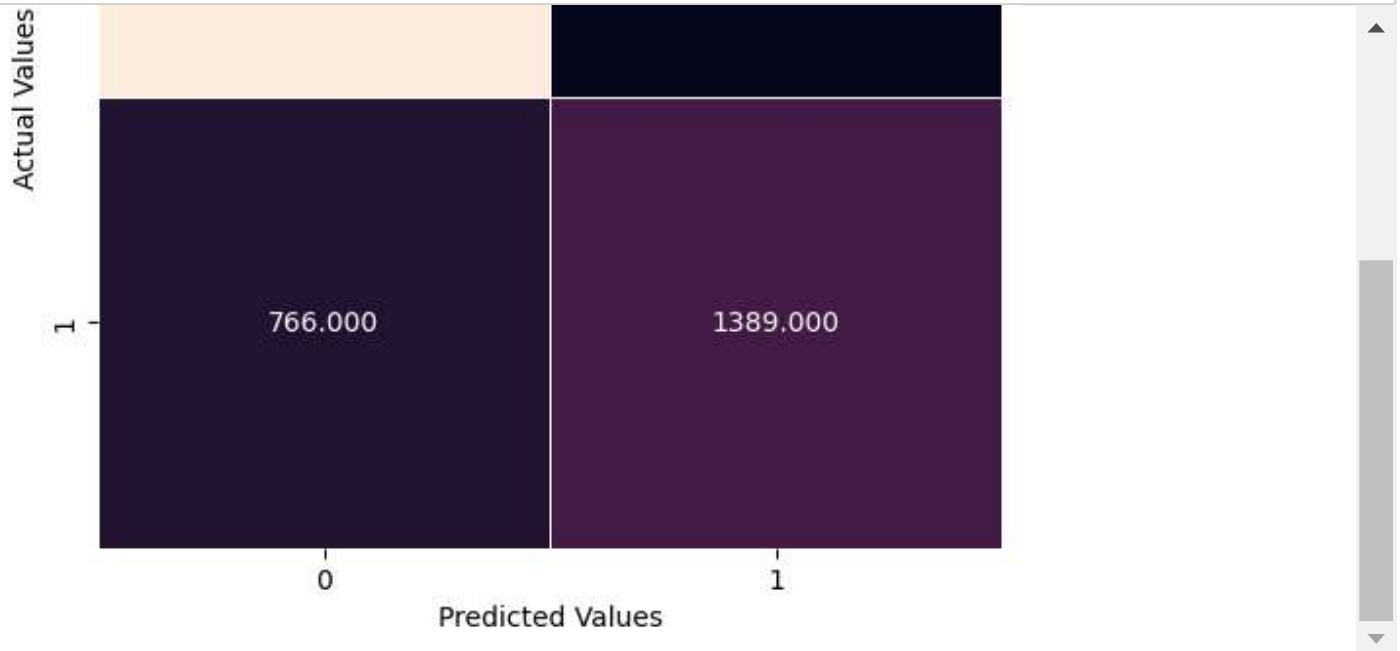
```
array([[7482,    88],
       [ 766, 1389]], dtype=int64)
```

In [229]:

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

In [230]:

```
sns.heatmap(cm, annot= True, fmt = '0.3f', linewidth = 0.5,
             square = True, cbar = False)
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```



With the parameter max\_depth=8, we get 768 fn and 95 fp, both lower when applying best\_params

In [231]:

```
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

In [232]:

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

Out[232]:

0.912185089974293

## 4.3. Support Vector Machine

In [ ]:

```
from sklearn import svm
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
#svc = svm.SVC().fit(X_train, y_train)
svc = SVC(C = 0.1, kernel='poly', gamma='auto').fit(X_train, y_train)
```

In [ ]:

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

print('Accuracy of SVM classifier on test set: {:.5f}'
      .format(svc.score(X_test, y_test)))
```

In [ ]:

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

In [ ]:

```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
tn, fp, fn, tp
```

In [ ]:

```
sns.heatmap(cm, annot= True, fmt = '0.3f', linewidth = 0.5,
            square = True, cbar = False)
plt.title('Support Vector Machine')
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```

In [ ]:

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

## 4.4. XGBoost algorithm

In [72]:

```
# import Libraries for ML
import xgboost as xgb
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

In [71]:

```
#XGBoost Multiclassification Model
#max_depth is depth of decision tree, learning_rate: tỷ lệ học
default_xgb_model = xgb.XGBClassifier(max_depth=8, learning_rate=0.1, n_jobs=-1).fit(X_train,
print('Accuracy of XGB classifier on training set: {:.5f}'
      .format(default_xgb_model.score(X_train, y_train)))
print('Accuracy of XGB classifier on test set: {:.5f}'
      .format(default_xgb_model.score(X_test, y_test)))
```

Accuracy of XGB classifier on training set: 0.93469  
Accuracy of XGB classifier on test set: 0.93069

In [92]:

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

In [93]:

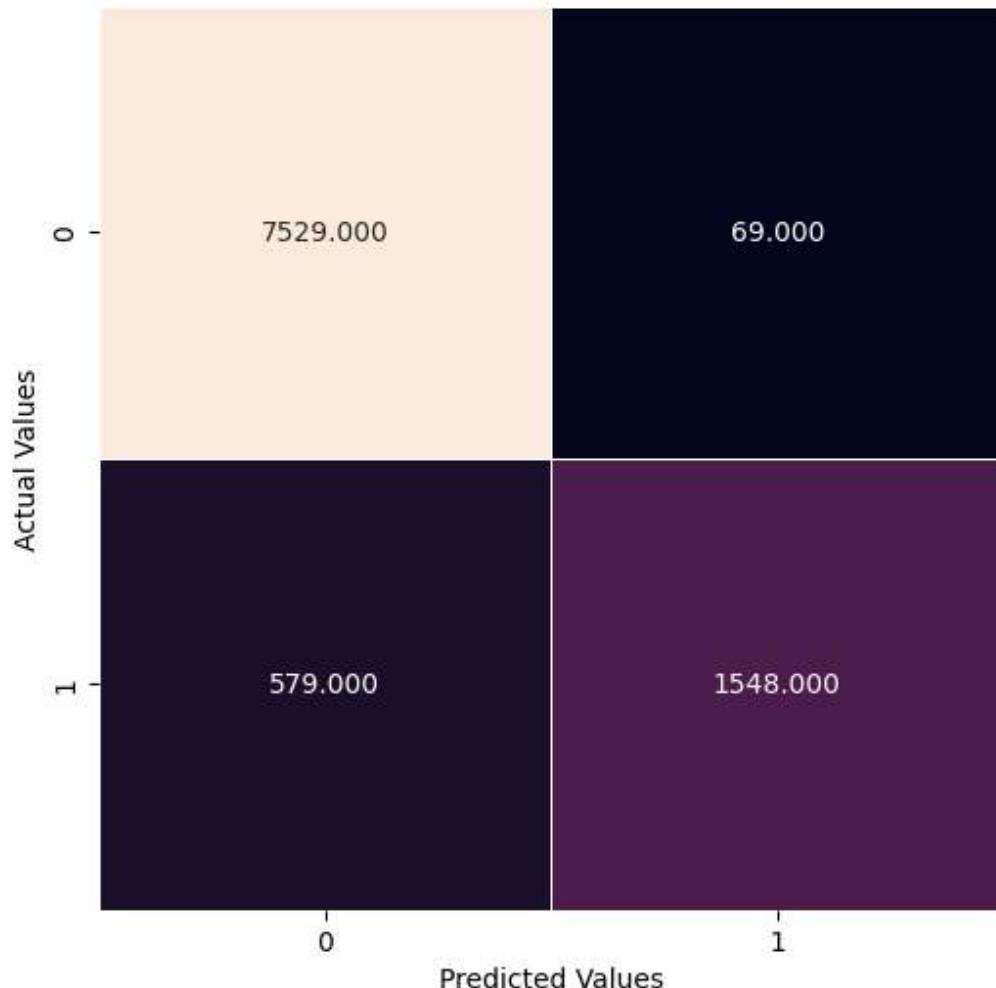
```
tn, fp, fn, tp = confusion_matrix(y_test, y_pred).ravel()
tn, fp, fn, tp
```

Out[93]:

(7529, 69, 579, 1548)

In [94]:

```
sns.heatmap(cm, annot= True, fmt = '0.3f', linewidth = 0.5,
             square = True, cbar = False)
plt.ylabel('Actual Values')
plt.xlabel('Predicted Values')
plt.show()
```



In [95]:

```
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

In [96]:

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

Out[96]:

0.9333676092544987

## #### Apply XGBoost with GridSearch

In [95]:

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

In [96]:

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

In [97]:

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

Out[97]:

```
GridSearchCV(estimator=XGBClassifier(base_score=None, booster=None,
                                      callbacks=None, colsample_bylevel=None,
                                      colsample_bynode=None,
                                      colsample_bytree=None,
                                      early_stopping_rounds=None,
                                      enable_categorical=False, eval_metric=None
                                      e,
                                      feature_types=None, gamma=None,
                                      gpu_id=None, grow_policy=None,
                                      importance_type=None,
                                      interaction_constraints=None,
                                      learning_rate=None, max_bin=None,
                                      max_cat_threshold=None,
                                      max_cat_to_onehot=None,
                                      max_delta_step=None, max_depth=None,
                                      max_leaves=None, min_child_weight=None,
                                      missing=nan, monotone_constraints=None,
                                      n_estimators=100, n_jobs=None,
                                      num_parallel_tree=None, predictor=None,
                                      random_state=None, ...),
                                      param_grid={'max_depth': [4, 5, 6, 7, 8, 9]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [98]:

```
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

In [99]:

```
default_xgb_model.best_params_
```

Out[99]:

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

## check GridSearch if it is better or not

In [109]:

```
# import Libraries for ML
import xgboost as xgb
from sklearn.model_selection import KFold, cross_val_score, train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

In [107]:

```
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

In [108]:

```
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

In [110]:

```
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

In [111]:

```
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

## Visualize the result and classification\_report

In [121]:

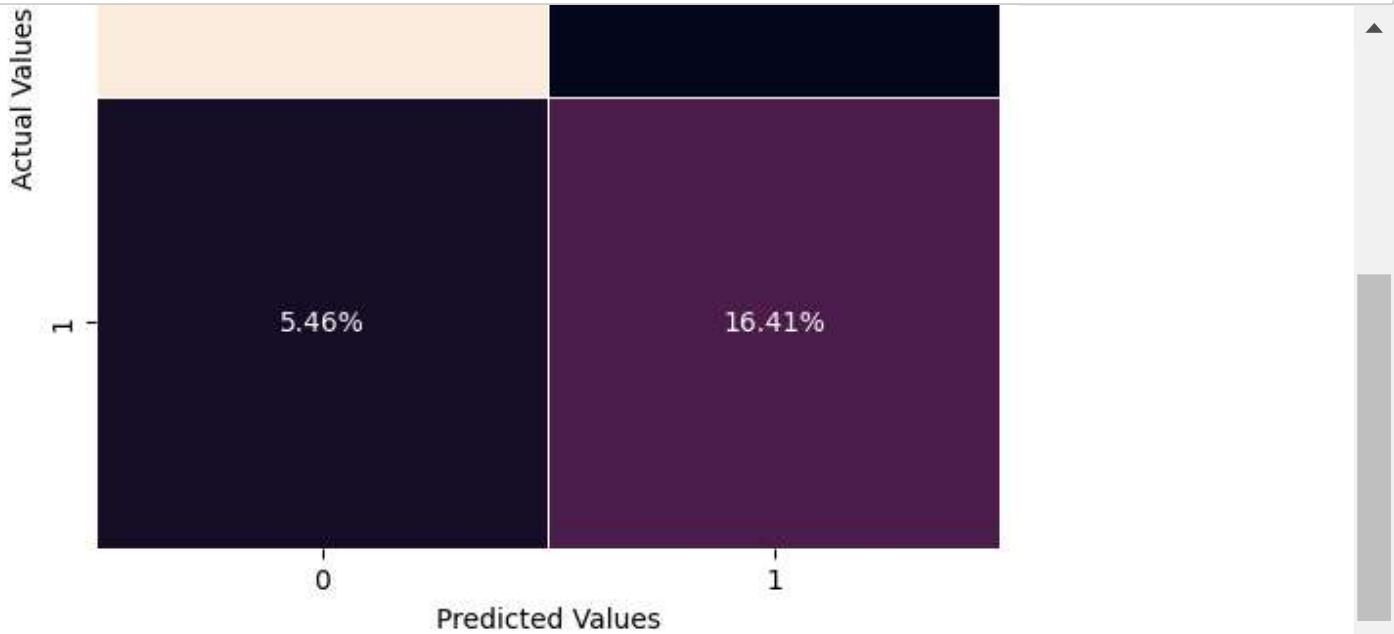
```
y_pred = xgb_model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[121]:

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

In [123]:

```
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()
```



In [124]:

```
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

Finally, after training four model: Logistic, Support Vector Machine, Decision Tree & XGBoost with GridSearch, we decide to choose XGBoost with the accuracy score of train 95% and 93% with test.

## 5. Save model

In [128]:

```
import pickle

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

In [129]:

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

In [130]:

```
model
```

Out[130]:

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=8, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.

On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

Finally, after training four model: Logistic, Support Vector Machine, Decision Tree & XGBoost with GridSearch, we decide to choose XGBoost with the accuracy score of train 95% and 93% with test.

## 6. Plot roc curve for these models

start from step: from sklearn.preprocessing import StandardScaler

In [66]:

```
import xgboost as xgb
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
```

## 6.1. StandardScaler

In [96]:

```
# X and y will be thought of as the entire data
X = df_cleaned_dum.drop('loan_status', axis=1).copy()
y = df_cleaned_dum['loan_status']
#y = y[X.index]
X.shape, y.shape
```

Out[96]:

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

In [97]:

```
y = y[X.index]
X = X[:1000]
y = y[:1000]
X.shape, y.shape
```

Out[97]:

```
((1000, 25), (1000,))
```

In [98]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, test_size=0.3, shu
```

In [99]:

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

In [100]:

```
y_test.value_counts(normalize=True) # independent variable is quite imbalance, we can use sar
```

Out[100]:

```
loan_status
1    0.583333
0    0.416667
Name: proportion, dtype: float64
```

## 6.2. Conduct model and calculate accuracy score

In [101]:

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

In [104]:

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

In [105]:

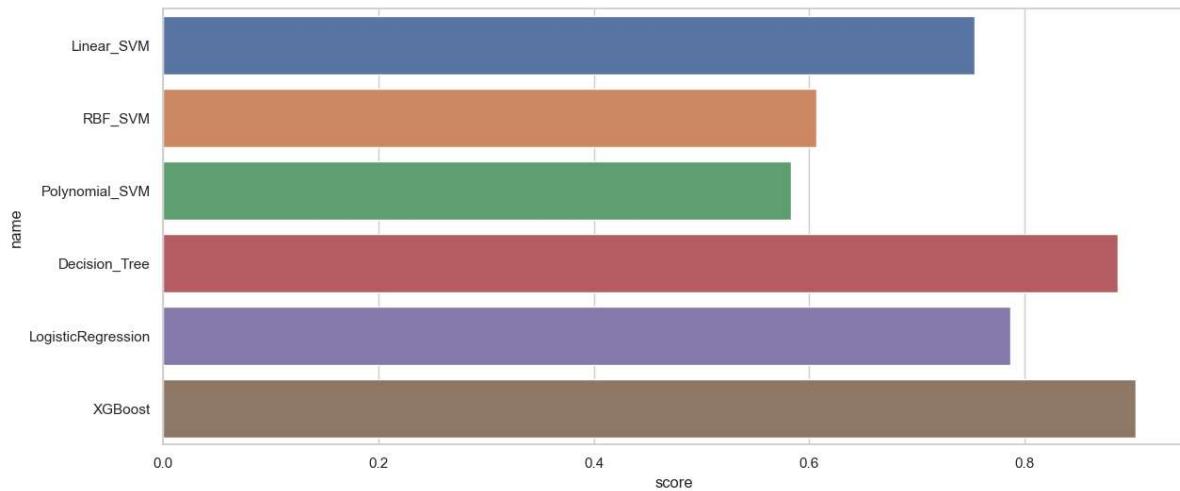
```
import pandas as pd
import seaborn as sns
df = pd.DataFrame()
df['name'] = names
df['score'] = scores
df
```

Out[105]:

	name	score
0	Linear_SVM	0.753333
1	RBF_SVM	0.606667
2	Polynomial_SVM	0.583333
3	Decision_Tree	0.886667
4	LogisticRegression	0.786667
5	XGBoost	0.903333

In [106]:

```
sns.set(style="whitegrid")
ax = sns.barplot(y="name", x="score", data=df)
```



### 6.3. Plot roc curve by calculate probability

In [109]:

```
svm_probs_lr = svm_probs_lr[:, 1]
svm_probs_rbf = svm_probs_rbf[:, 1]
svm_probs_pl = svm_probs_pl[:, 1]
dt_probs = dt_probs[:, 1]
log_probs = log_probs[:, 1]
xgb_probs = xgb_probs[:, 1]
```

In [110]:

```
from sklearn.metrics import roc_curve, roc_auc_score

svm_lr_auc = roc_auc_score(y_test, svm_probs_lr)
svm_rbf_auc = roc_auc_score(y_test, svm_probs_rbf)
svm_pl_auc = roc_auc_score(y_test, svm_probs_pl)
dt_auc = roc_auc_score(y_test, dt_probs)
log_auc = roc_auc_score(y_test, log_probs)
xgb_auc = roc_auc_score(y_test, xgb_probs)
```

In [111]:

```
print('Linear_SVM: AUROC = %.3f' % (svm_lr_auc))
print('RBF_SVM: AUROC = %.3f' % (svm_rbf_auc))
print('Polynomial_SVM: AUROC = %.3f' % (svm_pl_auc))
print('Decision Tree: AUROC = %.3f' % (dt_auc))
print('Logistics: AUROC = %.3f' % (log_auc))
print('XGBoost: AUROC = %.3f' % (xgb_auc))
```

Linear\_SVM: AUROC = 0.848  
RBF\_SVM: AUROC = 0.757  
Polynomial\_SVM: AUROC = 0.776  
Decision Tree: AUROC = 0.879  
Logistics: AUROC = 0.863  
XGBoost: AUROC = 0.962

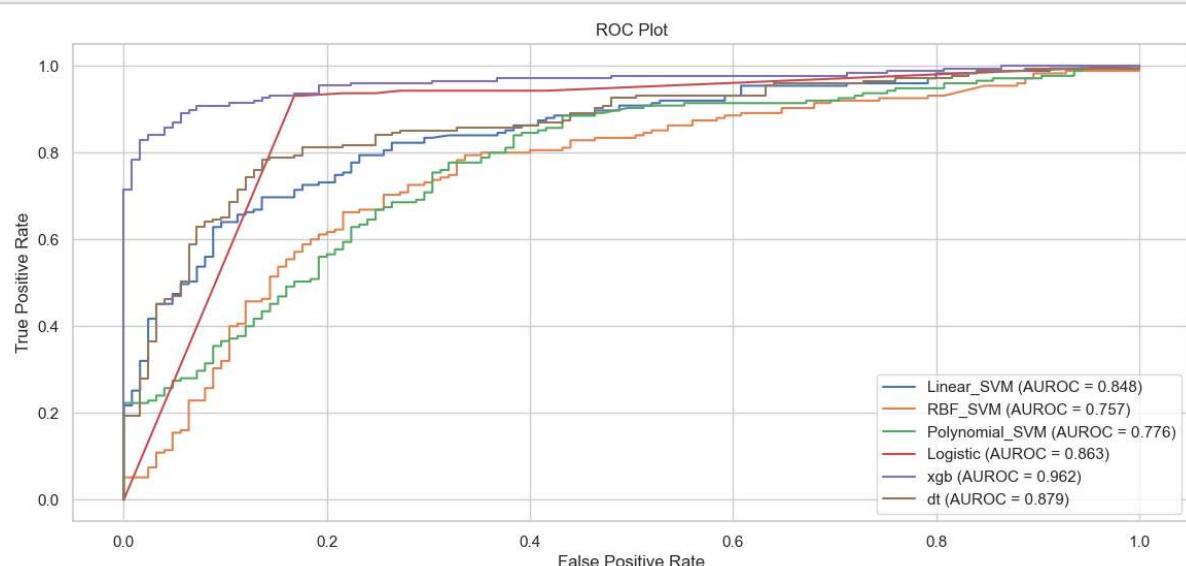
In [112]:

```
svm_fpr_lr, svm_tpr_lr, _ = roc_curve(y_test, svm_probs_lr)
svm_fpr_rbf, svm_tpr_rbf, _ = roc_curve(y_test, svm_probs_rbf)
svm_fpr_pl, svm_tpr_pl, _ = roc_curve(y_test, svm_probs_pl)
logistic_fpr, logistic_tpr, _ = roc_curve(y_test, dt_probs)
dt_fpr, dt_tpr, _ = roc_curve(y_test, log_probs)
xgb_fpr, xgb_tpr, _ = roc_curve(y_test, xgb_probs)
```

In [136]:

```
plt.plot(svm_fpr_lr, svm_tpr_lr, linestyle='-', label='Linear_SVM (AUROC = %0.3f)' % svm_lr_auc)
plt.plot(svm_fpr_rbf, svm_tpr_rbf, linestyle='-', label='RBF_SVM (AUROC = %0.3f)' % svm_rbf_auc)
plt.plot(svm_fpr_pl, svm_tpr_pl, linestyle='-', label='Polynomial_SVM (AUROC = %0.3f)' % svm_pl_auc)
plt.plot(logistic_fpr, logistic_tpr, linestyle='-', label='Logistic (AUROC = %0.3f)' % log_auc)
plt.plot(xgb_fpr, xgb_tpr, linestyle='-', label='xgb (AUROC = %0.3f)' % xgb_auc)
plt.plot(dt_fpr, dt_tpr, linestyle='-', label='dt (AUROC = %0.3f)' % dt_auc)

# Title
plt.title('ROC Plot')
# Axis Labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# Show Legend
plt.legend() #
# Show plot
plt.show()
```



In six choosen model, we decide to determine XGBoost model with the highest AUROC = 0.962.