Topic : Predicting whether a customer will default on his/her credit card using Machine Learning(Classification)

Mihir Kulkarni, Ram Bakale

Data Science Trainees

Alma Better

Abstract:

Our study contains the findings done on the case of customers' default payments in Taiwan.In this we solved the problem of predicting the case of customers' default payments in Taiwan. From the perspective of risk management, the result of predictive accuracy of the estimated probability of default will be more valuable than the binary result of classification - credible or not credible clients. In this project, we found the efficacy of standard machine learning techniques namely Logistic Regression,SVC,Random Forest,XGBoost by implementing and analyzing their performance.

Problem statement:

This project is aimed at predicting the case of customers' default payments in Taiwan. From the perspective of risk management, the result of predictive accuracy of the estimated probability of default will be more valuable than the binary result of classification - credible or not credible clients.

Introduction to default of credit card clients Data set.

There is 1 .XLS file we have in our field of study namely ‘default of credit card clients’ we have dealt with 25 columns ,30000 rows. The dataset contains weather information such as ID, Limit balance , sex, education etc.

Attribute Information:

This research employed a binary variable, default payment (Yes = 1, No = 0), as the response variable. This study reviewed the literature and used the following 23 variables as explanatory variables:

- X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
- X2: Gender (1 = male; 2 = female).
- X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).

- X4: Marital status (1 = married; 2 = single; 3 = others).
- X5: Age (year).
- X6 - X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above.
- X12-X17: Amount of bill statement (NT dollar). X12 = amount of bill statement in September, 2005; X13 = amount of bill statement in August, 2005; . . .; X17 = amount of bill statement in April, 2005.
- X18-X23: Amount of previous payment (NT dollar). X18 = amount paid in September, 2005; X19 = amount paid in August, 2005; . . .;X23 = amount paid in April, 2005.

- Importing the libraries

- import numpy as np
- import pandas as pd
- import matplotlib.pyplot as plt
- import seaborn as sns
- %matplotlib inline
- import warnings
- warnings.filterwarnings('ignore')
- # Importing packages
- from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, roc_auc_score, confusion_matrix, roc_curve, auc

Loading the data

```
from google.colab import drive
 drive.mount('/content/drive')
```

 # Importing Dataset

file_path = '/content/drive/MyDrive/Credit Card Default Prediction/Data & Resources/default of credit card clients.xls'

top 5 rows of the given dataset

data.head()

bottom 5 rows of the given dataset

data.tail()

Preprocessing the dataset

In the real world the data has a lot of missing values and it is due to data corruption or failure to record the data. For that purpose it is very important to handle the missing values. Also many machine learning algorithms do not support missing values, that's why we check missing values first.

Checking the count of null values in our dataset.

There are no null values and duplicate values in our dataset.

<u>Exploratory Data Analysis</u>

this will count the values of column of the dataset

this is the dependent variable

df['Next_month_defaulter'].value_counts()

0 23364

1 6636

Name: Next_month_defaulter, dtype: int64

<u>Dependent Variable</u>

Replacing the values of 0 and 1 to string values for better understanding.

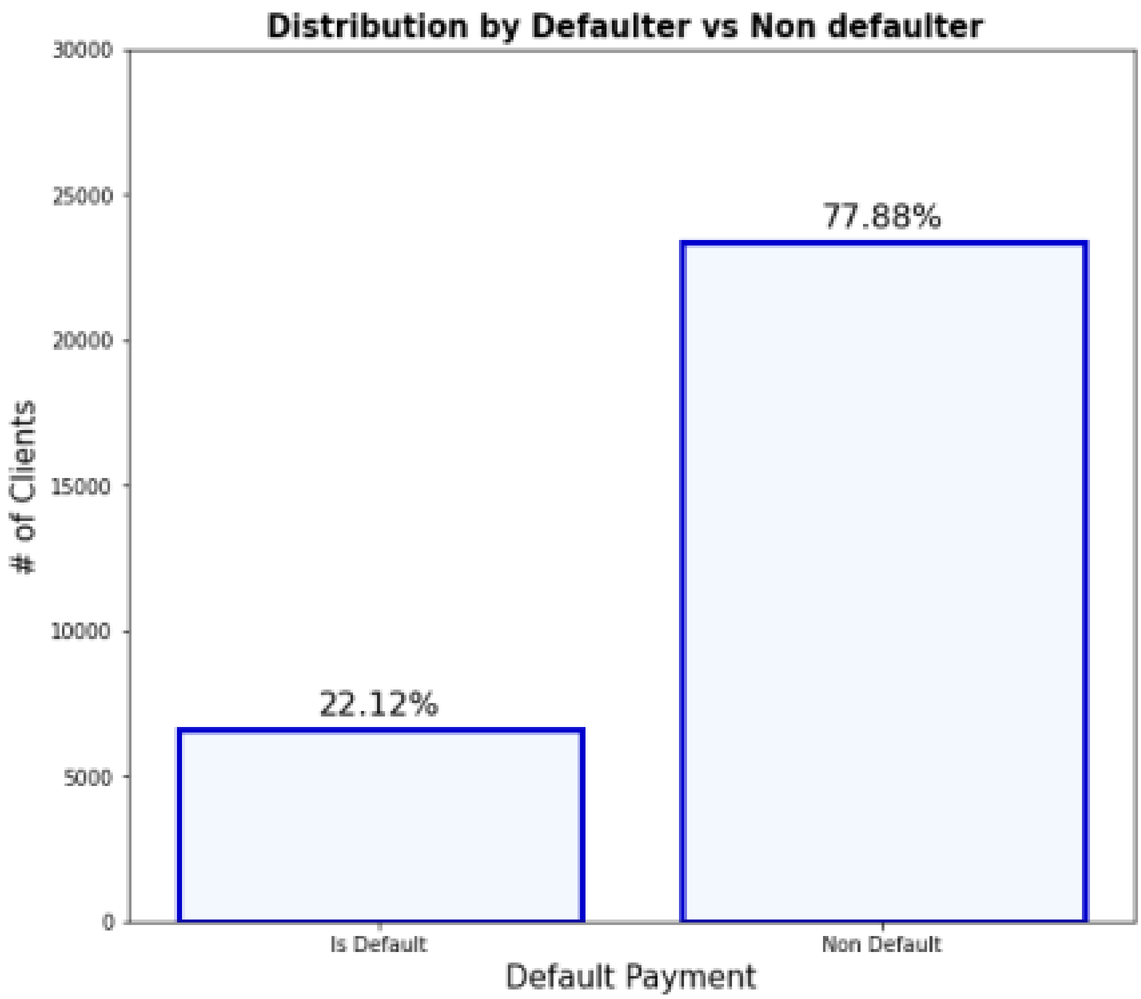
df['Defaulter'] = df.Next_month_defaulter.replace([1,0], ['Is Default', 'Non Default'])

#plotting the count plot to visualize the data distribution
plt.figure(figsize=(9,8))

ax = sns.countplot(x="Defaulter", data=df,color = 'aliceblue', edgecolor = 'medium blue',lw =3)

plt.xlabel("Default Payment", font size= 15)

```
plt.ylabel("# of Clients", font size= 15)

plt.ylim(0,30000) # making the y-axis limit to 30,000

plt.title('Distribution by Defaulter vs Non defaulter ',weight ='bold', font size=
15)

for p in ax.patches: # This step is used for showing the percentage on the graph

    height = p.get_height()

                    ax.text(p.get_x()+p.get_width()/2, height+500,
            '{:1.2f}"%'.format(height/df.shape[0]*100),ha = "center",
fontsize= 16)
```



From the above plot we can see that Defaulters are less than the Non Defaulters. Approx 78% are Non Defaulters and 22% are Defaulters.

Independent variable

The categorical features in dataset that are:

SEX

EDUCATION
MARRIAGE

Age

checking their relation with dependent variable.

SEX

```
print('SEX column distribution : 1=Male, 2=Female')
df['SEX'].value_counts()
```

SEX column distribution : 1=Male, 2=Female

2 18112

1 11888

Name: SEX, dtype: int64

Education

```
print('EDUCATION column distribution : 1=Graduate school, 2=University,
3=High school, 4=Others, 5=unknown, 6=unknown')
df['EDUCATION'].value_counts()
```

2 14030

1 10930

3 4917

4 123

Name: EDUCATION, dtype: int64

MARRIAGE

```
print('MARRIAGE column distribution : 1=Married, 2=Single, 3=Others')
df['MARRIAGE'].value_counts()
```

MARRIAGE column distribution : 1=Married, 2=Single, 3=Others

2 15964

1 13659

3 323

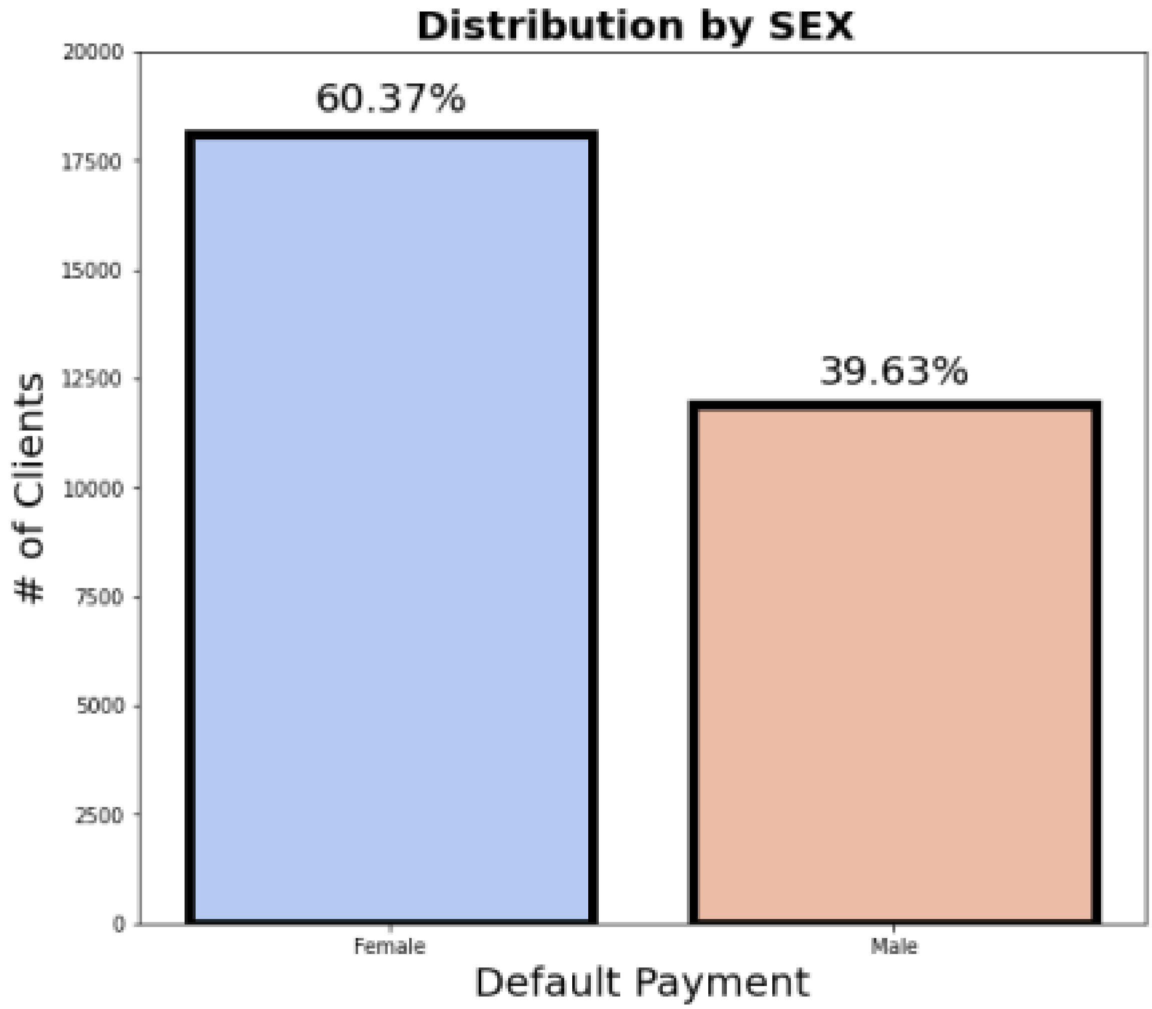
0 54

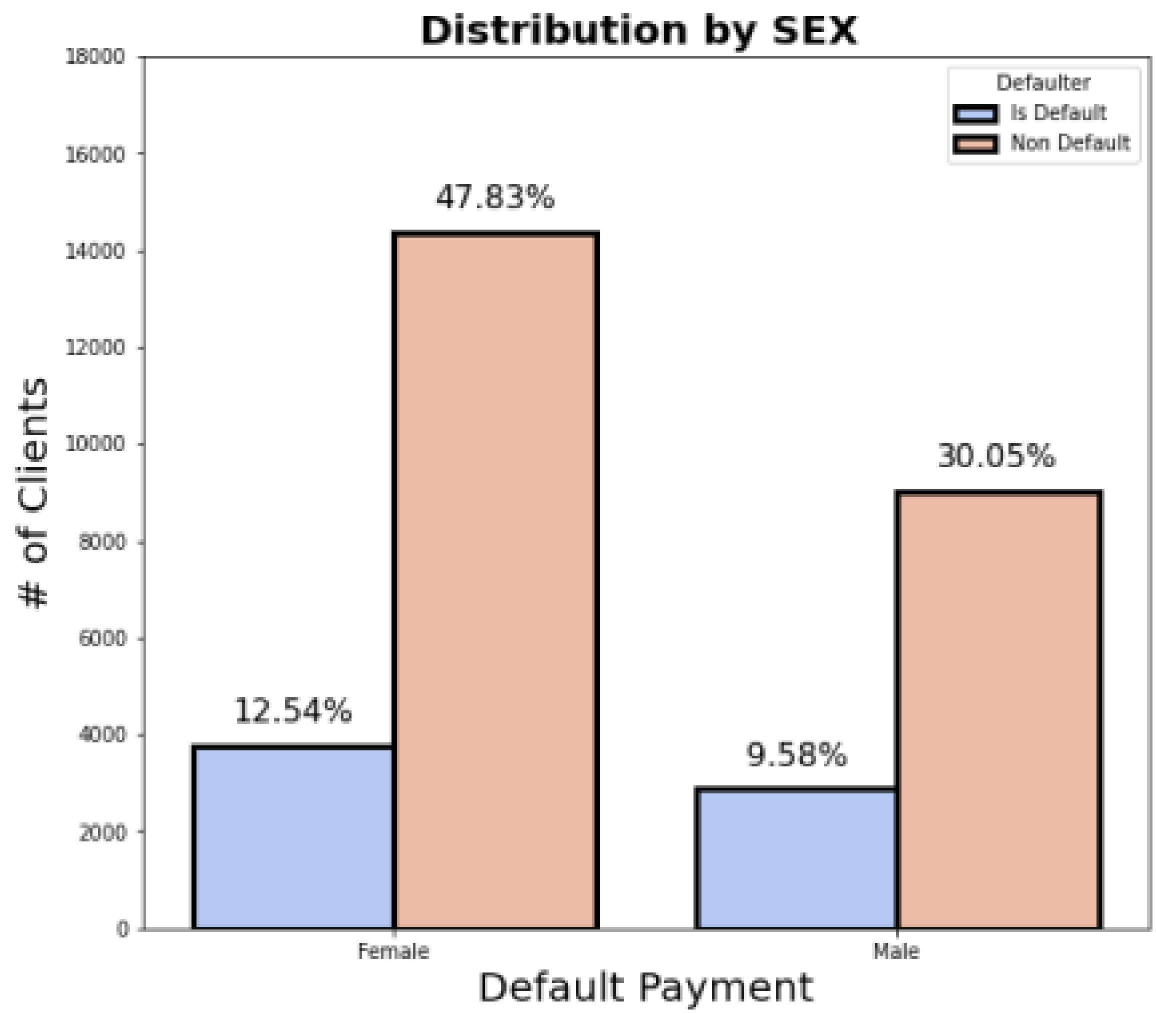
Name: MARRIAGE, dtype: int64

Based on categorical features we can say whether a customer is default or not.

```
#plotting graph for SEX feature

plt.figure(figsize=(9,8))

ax = sns.countplot(x="SEX", data=df_category, palette = 'coolwarm',
edgecolor = 'black',lw =5)

plt.xlabel("Default Payment", fontsize= 20)
plt.ylabel("# of Clients", fontsize= 20)

plt.ylim(0,20000)

plt.title('Distribution by SEX ',weight ='bold', fontsize= 20) for p in
ax.patches:

    height = p.get_height()

    ax.text(p.get_x()+p.get_width()/2, height+500,
'{:1.2f}"%'.format(height/df.shape[0]*100),ha = "center", fontsize= 20)


# plotting graph for SEX [Is Defaulter or not]

plt.figure(figsize=(9,8))

ax = sns.countplot(x="SEX", data=df_category, palette = 'coolwarm',
hue="Defaulter" ,edgecolor = 'black',lw =3)

plt.xlabel("Default Payment", fontsize= 20)

plt.ylabel("# of Clients", fontsize= 20)
```

```
plt.ylim(0,18000)

plt.title('Distribution by SEX ',weight ='bold', fontsize= 20) for p in

ax.patches:

    height = p.get_height()

    ax.text(p.get_x()+p.get_width()/2, height+500,
'{:1.2f}"%'.format(height/df.shape[0]*100),ha = "center", fontsize= 16)
```



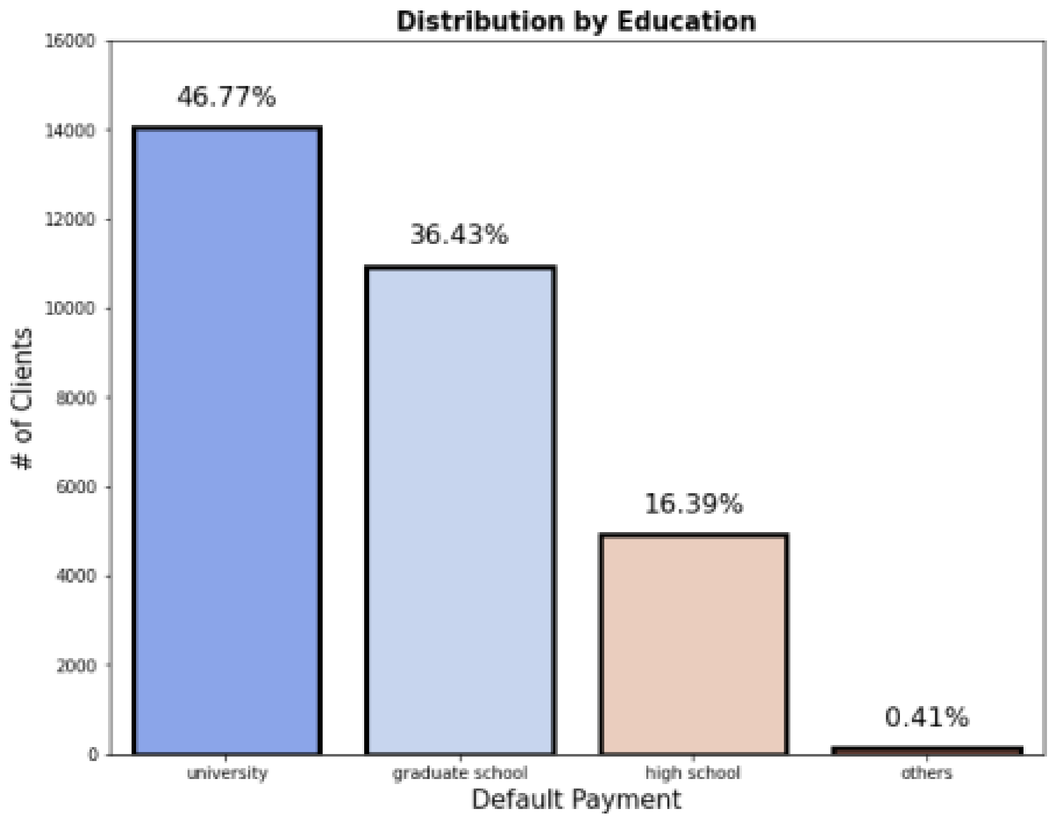


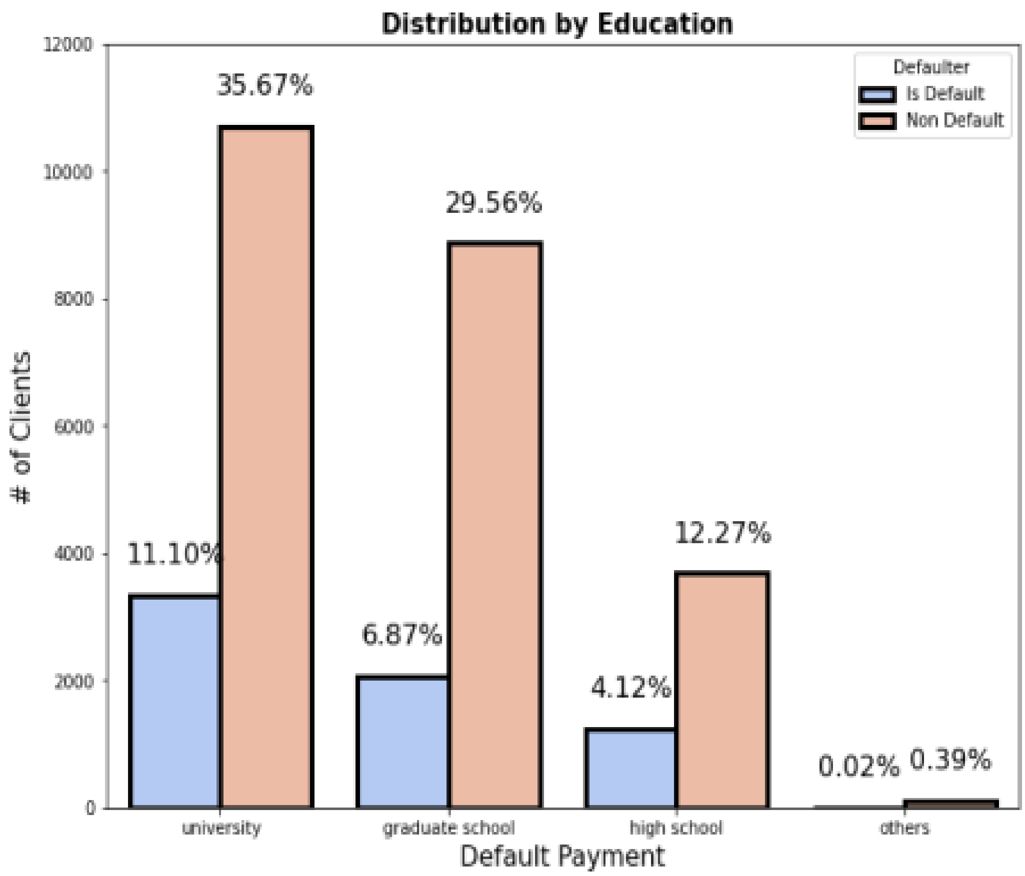
From the above data analysis we can say that Number of Male credit holders is less than Female.

Approx 40% are male and 60% are Female and in that 10% are default from male & 13% are default from female.

```
print(df_category['EDUCATION'].value_counts(),'\n')

print(df_category.groupby(['EDUCATION',
'Defaulter']).size().unstack())



#plotting graph for Education
plt.figure(figsize=(10,8))

ax = sns.countplot(x="EDUCATION", data=df_category, palette = 'coolwarm',
edgecolor = 'black',lw =3)

plt.xlabel("Default Payment", font size= 15)

plt.ylabel("# of Clients", font size= 15)

plt.ylim(0,16000)
```

```
plt.title('Distribution by Education',weight ='bold', font size= 15) for p in ax.patches:

    height = p.get_height()

    ax.text(p.get_x()+p.get_width()/2, height+500,
'{:1.2f}"%'.format(height/df.shape[0]*100),ha = "center", fontsize= 16)


# plotting graph for Education [university, graduate school,
highschool, others]

plt.figure(figsize=(10,8))

ax = sns.countplot(x="EDUCATION", data=df_category, palette = 'coolwarm',
hue="Defaulter" ,edgecolor = 'black',lw =3)

plt.xlabel("Default Payment", font size= 15)

plt.ylabel("# of Clients", font size= 15)

plt.ylim(0,12000)

    plt.title('Distribution by Education ',weight ='bold', font size= 15)
for p in ax.patches:

    height = p.get_height()

    ax.text(p.get_x()+p.get_width()/2, height+500,
'{:1.2f}"%'.format(height/df.shape[0]*100),ha = "center", fontsize= 16)
```





From the above data analysis we can say that

1 - graduate school

2 - university

3 - high school

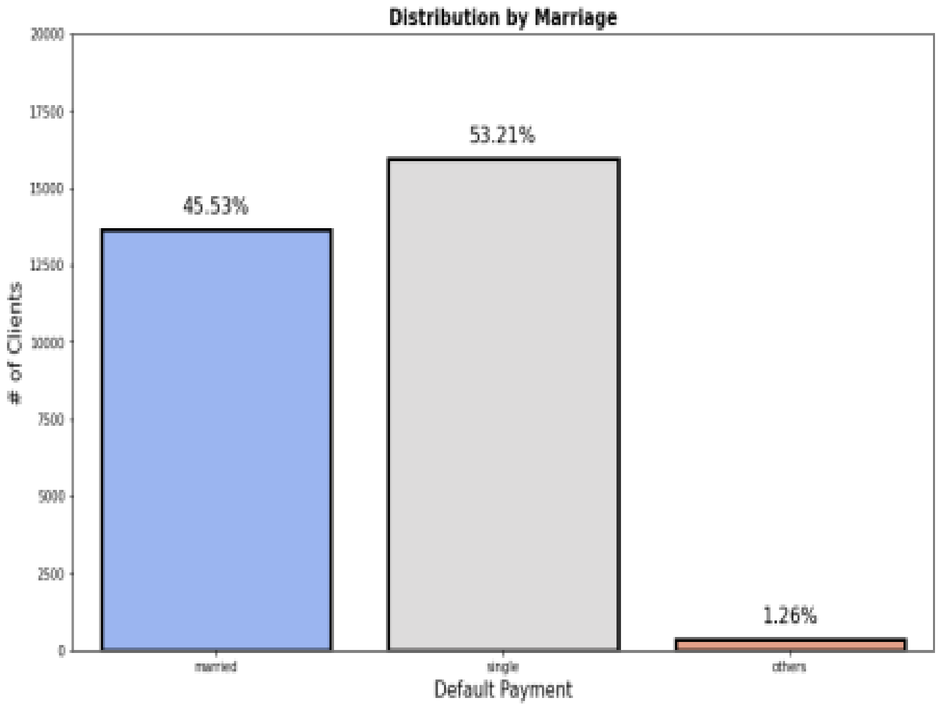
4 - others

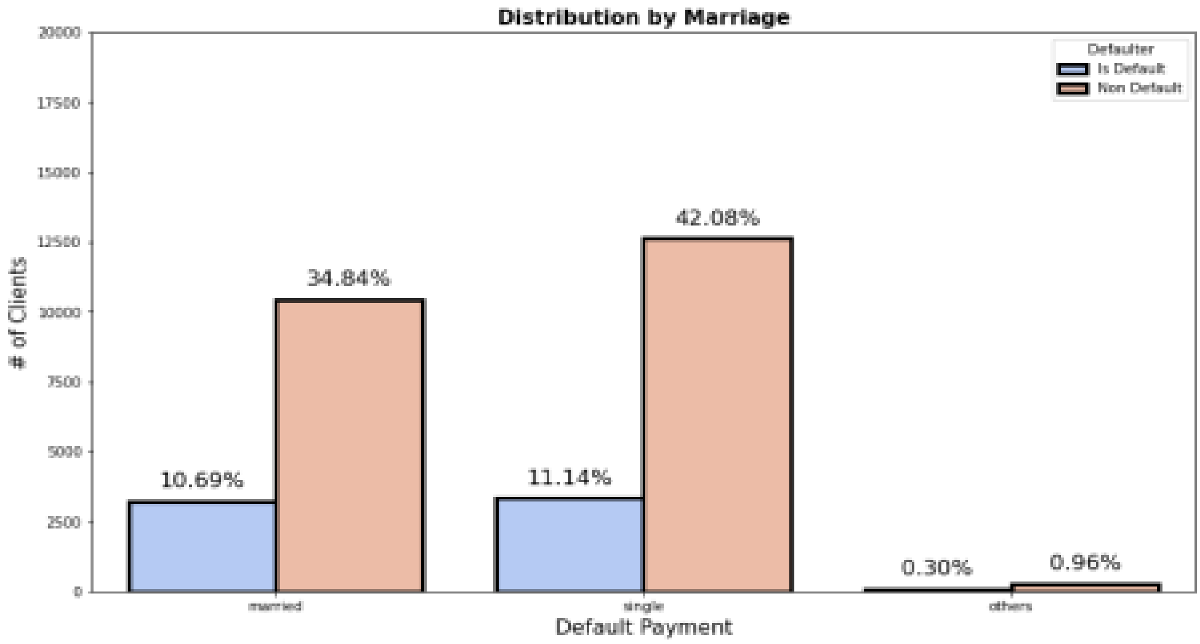
More credit card holders are university students followed by Graduates and then High school students.

From university 11% are default, from graduate 7% are default, and from high school 4% are default.

```
print(df_category['MARRIAGE'].value_counts(),'\n')
print(df_category.groupby(['MARRIAGE',
'Defaulter']).size().unstack())
#plotting graph for Marriage

plt.figure(figsize=(14,8))

ax = sns.countplot(x="MARRIAGE", data=df_category, palette = 'coolwarm',
edgecolor = 'black',lw =3)

plt.xlabel("Default Payment", font size= 15)

plt.ylabel("# of Clients", font size= 15)

plt.ylim(0,20000)

plt.title('Distribution by Marriage',weight ='bold', font size= 15) for p in ax.patches:

    height = p.get_height()

    ax.text(p.get_x()+p.get_width()/2, height+500,
'{:1.2f}"%'.format(height/df.shape[0]*100),ha = "center", fontsize= 16)



# plotting graph for Marriage [married, single, others]

plt.figure(figsize=(14,8))

ax = sns.countplot(x="MARRIAGE", data=df_category, palette = 'coolwarm',
hue="Defaulter" ,edgecolor = 'black',lw =3)

plt.xlabel("Default Payment", font size= 15)
plt.ylabel("# of Clients", font size= 15)

plt.ylim(0,20000)

plt.title('Distribution by Marriage',weight ='bold', font size= 15)
```

```
for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2, height+500,
'{:1.2f}"%'.format(height/df.shape[0]*100),ha = "center", fontsize= 16)
```





From the above data analysis we can say that

1 - Married

2 - Single

3 - Others

More credit card holders are single as compared to married people.

From single 11% are defaulters and from married approx 11% are defaulters.

```
#plotting the count plot to visualize the data distribution with respect to Age

plt.figure(figsize=[24, 8])
sns.countplot(x = 'AGE', hue = 'Defaulter', data =df, palette = 'husl', edgecolor =
'blue',lw=3)
```

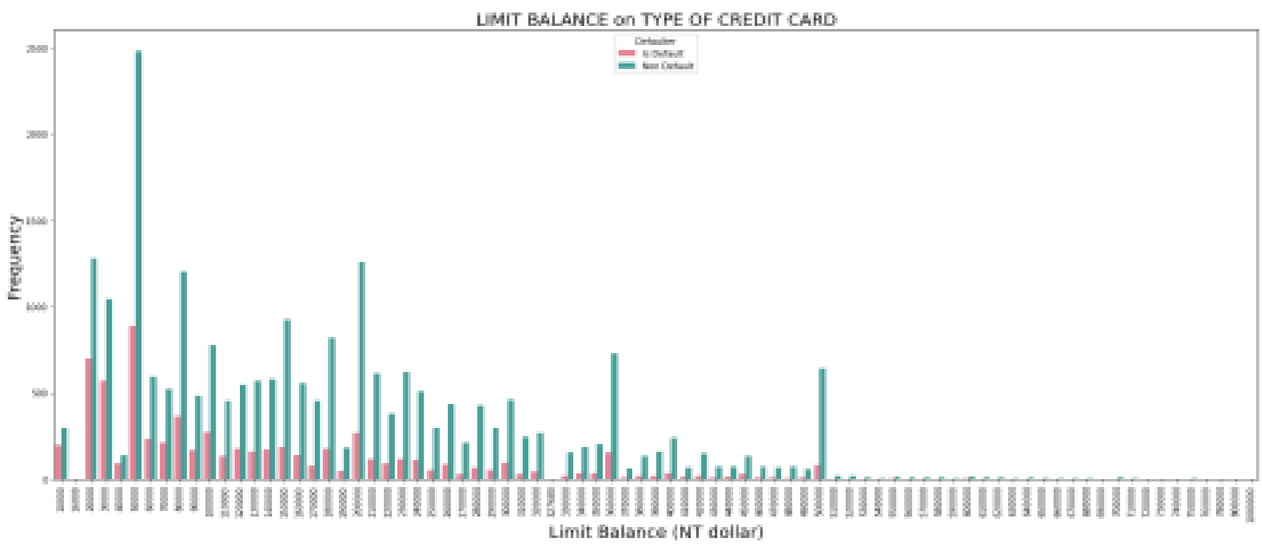
From the above graph we can say that

More credit card holders between 26-32 years and 29 years age are the highest users of credit cards.

Those above 60 years old rarely use the credit card. The number of

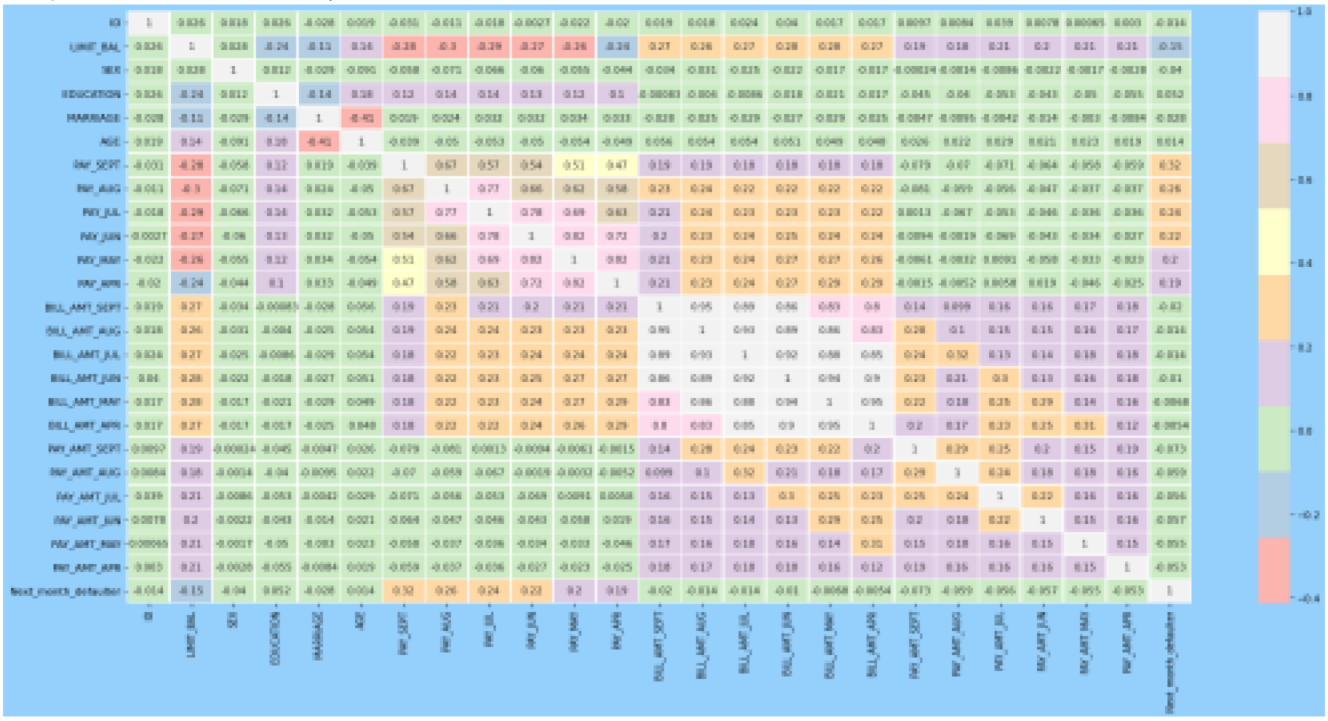
Defaulters is between 27-29 years.

```
#plotting the count plot to visualize the data distribution with respect to Limit
Balance

plt.figure(figsize=[25, 10])

sns.countplot(x = 'LIMIT_BAL', hue = 'Defaulter',data =df, palette = 'husl')

plt.xticks(rotation = 90)

plt.xlabel('Limit Balance (NT dollar)', SIZE=20)
plt.ylabel('Frequency', SIZE=20)

plt.title('LIMIT BALANCE on TYPE OF CREDIT CARD', SIZE=20)
```



Checking correlation between the variables

```
plt.figure (figsize= (24,12),edgecolor='k',facecolor='xkcd:light blue')

correlation= df.corr()

sns.heatmap(correlation, annot=True,
cmap='Pastel1',linewidth=.6)
```



There are negatively correlated features like age and marriage.

FEATURE ENGINEERING

ONE HOT ENCODING

One hot encoding is a process by which categorical variables are converted into numerical variables so that they can be provided to ML algorithms. We are performing one hot encoding on 'EDUCATION', 'MARRIAGE', and 'SEX'.

Creating the Dependent and Independent Variables:

```
X = df.drop(['Next_month_defaulter'] , axis=1)

y = df['Next_month_defaulter']

# using lambda function

X = X.apply(lambda x : (x-np.mean(x))/np.std(x))
```

Splitting the dataset into training and test sets.

```
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=42, stratify = y)
```

Checking the shape of train dataset

```
print(X_train.shape,y_train.shape)
```

(20883, 28) (20883,)

```
# Check the shape of test dataset
print(X_test.shape, y_test.shape)
```

(8950, 28) (8950,)

Oversampling

As there is imbalance in the dataset so we have to apply Random Over Sampling to balance it.

Performance Metrics

Precision is a good metric to use when the cost of false positive(FP) is high.

Precision = TP / (TP + FP)

Recall is a good metric to use when the cost associated with false negative(FN) is high.

Recall = TP / (TP + FN)

F1-score is a weighted average of precision and recall. Thus, it considers FP and FN. This metric is very useful when we have uneven class distribution, as it seeks a balance between precision and recall.

F1-score = 2 (precision recall) / (precision + recall)

Logistic Regression

Implementing Logistic Regression

```
from sklearn.linear_model import LogisticRegression from
sklearn.model_selection import GridSearchCV

param_grid = {'penalty':['l1','l2'], 'C' : [0.001, 0.01, 0.1, 1, 10, 100, 1000] } #set
```

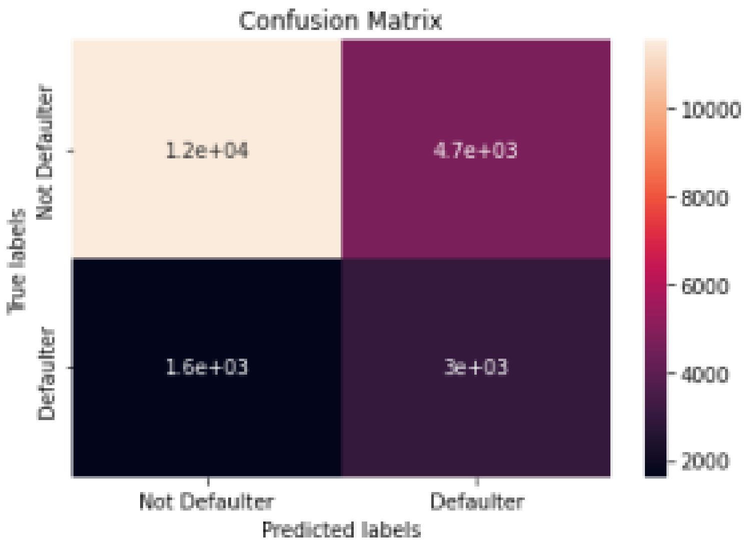
the parameter

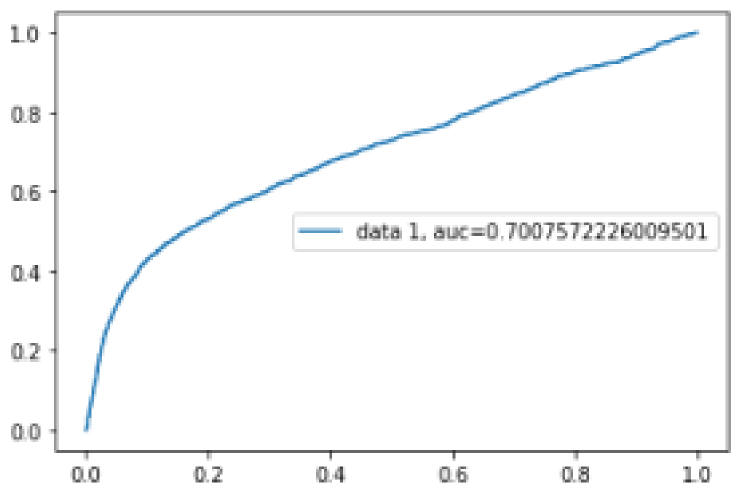
The accuracy on train data is 0.6970262893262462

The accuracy on test data is 0.6815642458100558

The accuracy on test data is 0.6815642458100558
The precision on test data is 0.5989924433249371
The recall on test data is 0.3666358310206599
The f1 on test data is 0.454858454475899
The roc_score on test data is 0.6135789983910028

Get the confusion matrix for both train and test





Implementing SVC

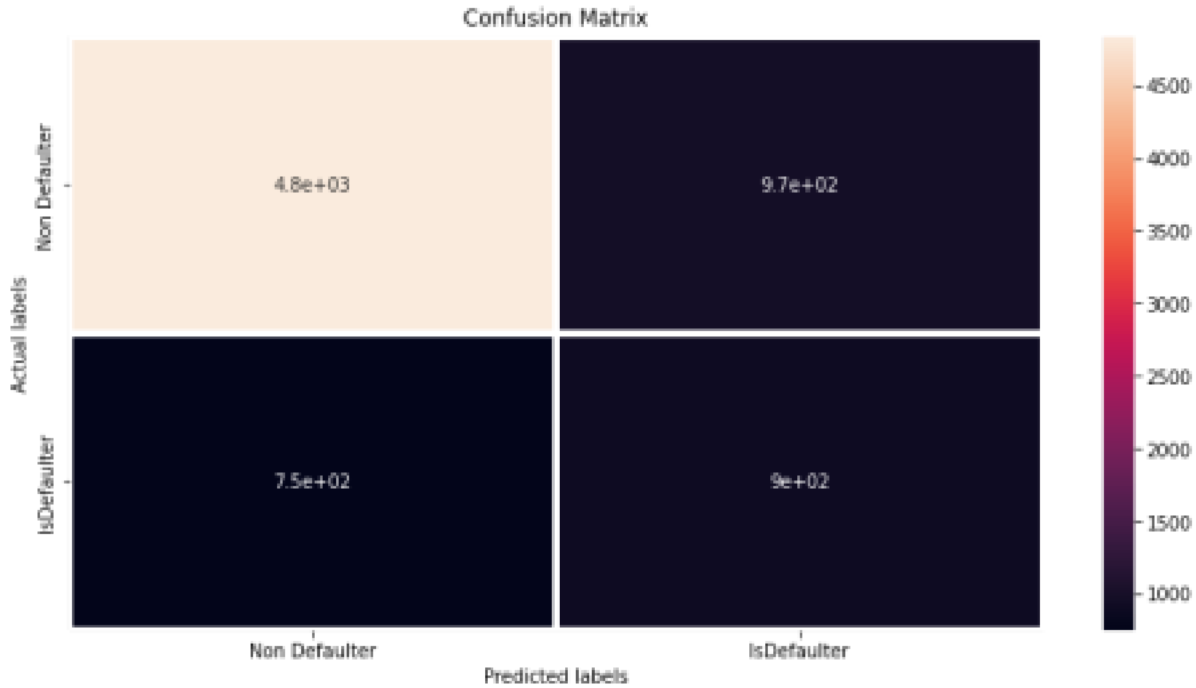
from sklearn import svm

#Create a svm Classifier

svm_model = svm.SVC(kernel = 'rbf')
The accuracy on train data is 0.7341086702356778

The accuracy on test data is 0.7696742190642177

Get the confusion matrix for svm

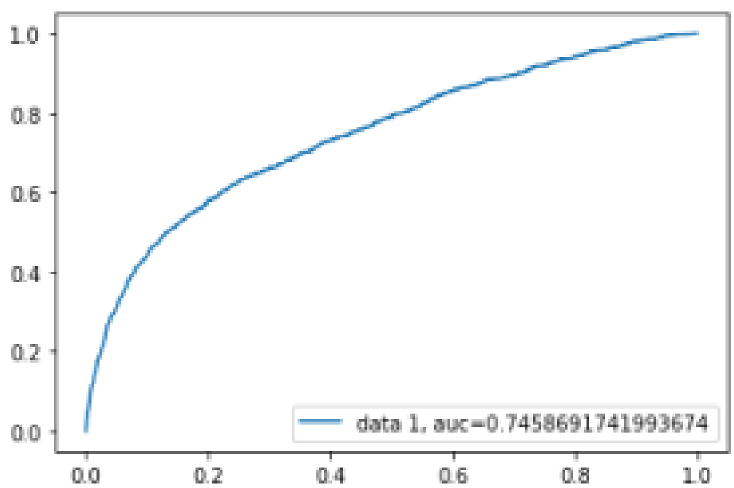


Implementing Random Forest

from sklearn.tree import DecisionTreeClassifier

The accuracy on train data is 0.9953517475641369

The accuracy on test data is 0.8078830942485588



Implementing XGBoost

The accuracy on train data is 0.8448198802181103

The accuracy on test data is 0.8108325512803325

Hyperparameter Tuning

The accuracy on train data is 0.8245284705461696

The accuracy on test data is 0.8212897171202574

Conclusion

1.There are no null values and duplicate values in our dataset.

2 .Defaulters are less than the Non Defaulters. Approx 78% are Non Defaulters and 22% are Defaulters.

3.Number of Male credit holders is less than Female.

4.Approx 40% are male and 60% are Female and in that 10% are default from male & 13% are default from female.

5.We use box plot to detect outliers.

6.We implemented ML models and found that best accuracy, F1 score is obtained from random forest classifier, XGB classifier.

7.With RF classifier we get test accuracy of 81%.

8.With logistic regression test accuracy is 68%.

9.With SVC test accuracy is 77%.

10.With XG Boost test accuracy is 82%.

11. Therefore we can conclude based on test accuracy, F1 score, recall XG Boost classifier, RF Classifiers are the best model to predict credit card default.