Project Title : Predicting whether a customer will default on his/her credit card

Problem Description

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. We can use the K-S chart to evaluate which customers will default on their credit card payments

Data Description

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.

Objective:

Objective of our project is to predict which customer might default in upcoming months. Before going any fudther let's have a quick look on defination of what actually meant by **Credit Card Default.**

> We are all aware what is **credit card**. It is type of payment payment card in which charges are made against a line of credit instead of the account holder's cash deposits. When someone uses a credit card to make a purchase, that person's account accrues a balance that must be paid off each month.

> **Credit card default** happens when you have become severely delinquent on your credit card payments.Missing credit card payments once or twice does not count as a default. A payment default occurs when you fail to pay the Minimum Amount Due on the credit card for a few consecutive months.

```
# Importing all libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')


from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, roc_auc_score, confusion_matrix, roc_curve, auc
```

```
from sklearn.svm import SVC
```

```
path = '../input/default-of-credit-card-clients-dataset/UCI_Credit_Card.csv'
df = pd.read_csv(path)
```

```
df
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT
0	1	20000.0	2	2	1	24	2	2	-1	-1	...	0.0	0.0	0.0	0.0	689.0	0
1	2	120000.0	2	2	2	26	-1	2	0	0	...	3272.0	3455.0	3261.0	0.0	1000.0	1000
2	3	90000.0	2	2	2	34	0	0	0	0	...	14331.0	14948.0	15549.0	1518.0	1500.0	1000
3	4	50000.0	2	2	1	37	0	0	0	0	...	28314.0	28959.0	29547.0	2000.0	2019.0	1200
4	5	50000.0	1	2	1	57	-1	0	-1	0	...	20940.0	19146.0	19131.0	2000.0	36681.0	10000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
29995	29996	220000.0	1	3	1	39	0	0	0	0	...	88004.0	31237.0	15980.0	8500.0	20000.0	5003
29996	29997	150000.0	1	3	2	43	-1	-1	-1	-1	...	8979.0	5190.0	0.0	1837.0	3526.0	8998
29997	29998	30000.0	1	2	2	37	4	3	2	-1	...	20878.0	20582.0	19357.0	0.0	0.0	22000
29998	29999	80000.0	1	3	1	41	1	-1	0	0	...	52774.0	11855.0	48944.0	85900.0	3409.0	1178
29999	30000	50000.0	1	2	1	46	0	0	0	0	...	36535.0	32428.0	15313.0	2078.0	1800.0	1430

30000 rows × 25 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 25 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   ID                          30000 non-null  int64  
 1   LIMIT_BAL                   30000 non-null  float64
 2   SEX                         30000 non-null  int64  
 3   EDUCATION                   30000 non-null  int64  
 4   MARRIAGE                    30000 non-null  int64  
 5   AGE                         30000 non-null  int64  
 6   PAY_0                       30000 non-null  int64  
 7   PAY_2                       30000 non-null  int64  
 8   PAY_3                       30000 non-null  int64  
 9   PAY_4                       30000 non-null  int64  
 10  PAY_5                       30000 non-null  int64  
 11  PAY_6                       30000 non-null  int64  
 12  BILL_AMT1                   30000 non-null  float64
 13  BILL_AMT2                   30000 non-null  float64
 14  BILL_AMT3                   30000 non-null  float64
 15  BILL_AMT4                   30000 non-null  float64
 16  BILL_AMT5                   30000 non-null  float64
 17  BILL_AMT6                   30000 non-null  float64
 18  PAY_AMT1                    30000 non-null  float64
 19  PAY_AMT2                    30000 non-null  float64
 20  PAY_AMT3                    30000 non-null  float64
 21  PAY_AMT4                    30000 non-null  float64
 22  PAY_AMT5                    30000 non-null  float64
 23  PAY_AMT6                    30000 non-null  float64
 24  default.payment.next.month  30000 non-null  int64  
dtypes: float64(13), int64(12)
memory usage: 5.7 MB
```

What we know about dataset :

We have records of 30000 customers. Below are the description of all features we have.

- ID: ID of each client
- LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit)
- SEX: Gender (1 = male, 2 = female)
- EDUCATION: (1 = graduate school, 2 = university, 3 = high school, 0,4,5,6 = others)
- MARRIAGE: Marital status (0 = others, 1 = married, 2 = single, 3 = others)
- AGE: Age in years

Scale for PAY_0 to PAY_6 : (-2 = No consumption, -1 = paid in full, 0 = use of revolving credit (paid minimum only), 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)

- PAY_0: Repayment status in September, 2005 (scale same as above)
- PAY_2: Repayment status in August, 2005 (scale same as above)
- PAY_3: Repayment status in July, 2005 (scale same as above)
- PAY_4: Repayment status in June, 2005 (scale same as above)
- PAY_5: Repayment status in May, 2005 (scale same as above)

- PAY_6: Repayment status in April, 2005 (scale same as above)
- BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
- BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)
- BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)
- BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)
- BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
- BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
- PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
- PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
- PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
- PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
- PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
- PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
- default.payment.next.month: Default payment (1=yes, 0=no)

In our dataset we got customer credit card transaction history for past 6 month , on basis of which we have to predict if cutomer will default or not.

So let's begin.

First we will check if we have any null values

```
df.isnull().sum()
```

```
ID                            0
LIMIT_BAL                     0
SEX                           0
EDUCATION                     0
MARRIAGE                      0
AGE                           0
PAY_0                         0
PAY_2                         0
PAY_3                         0
PAY_4                         0
PAY_5                         0
PAY_6                         0
BILL_AMT1                     0
BILL_AMT2                     0
BILL_AMT3                     0
BILL_AMT4                     0
BILL_AMT5                     0
BILL_AMT6                     0
PAY_AMT1                      0
PAY_AMT2                      0
PAY_AMT3                      0
PAY_AMT4                      0
PAY_AMT5                      0
PAY_AMT6                      0
default.payment.next.month    0
dtype: int64
```

```
df.describe()
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	...
mean	15000.500000	167484.322667	1.603733	1.853133	1.551867	35.485500	-0.016700	-0.133767	-0.166200	-0.220667	...
std	8660.398374	129747.661567	0.489129	0.790349	0.521970	9.217904	1.123802	1.197186	1.196868	1.169139	...
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.000000	-2.000000	-2.000000	-2.000000	-2.000000	...
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000	28.000000	-1.000000	-1.000000	-1.000000	-1.000000	...
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.000000	0.000000	0.000000	0.000000	0.000000	...
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.000000	0.000000	0.000000	0.000000	0.000000	...
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.000000	8.000000	8.000000	8.000000	8.000000	...

8 rows × 25 columns

Exploratory Data Analysis

Dependent Variable:

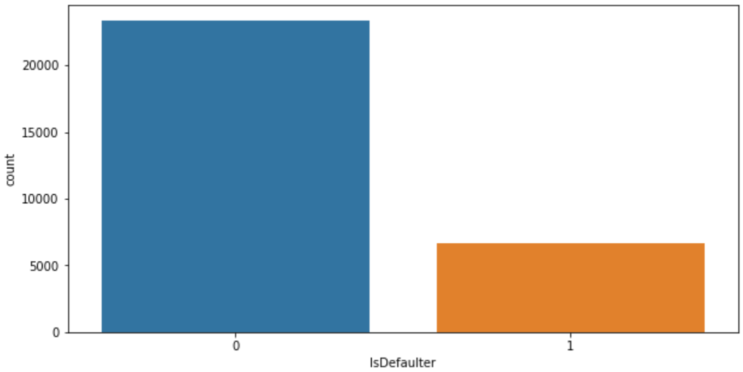
```
#renaming for better convinience
df['IsDefaulter'] =df ['default.payment.next.month']
df.drop('default.payment.next.month',axis = 1)
# df.rename({'default.payment.next.month' : 'IsDefaulter'}, inplace=True)
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AM
0	1	20000.0	2	2	1	24	2	2	-1	-1	...	0.0	0.0	0.0	0.0	689.0	0
1	2	120000.0	2	2	2	26	-1	2	0	0	...	3272.0	3455.0	3261.0	0.0	1000.0	1000
2	3	90000.0	2	2	2	34	0	0	0	0	...	14331.0	14948.0	15549.0	1518.0	1500.0	1000
3	4	50000.0	2	2	1	37	0	0	0	0	...	28314.0	28959.0	29547.0	2000.0	2019.0	1200
4	5	50000.0	1	2	1	57	-1	0	-1	0	...	20940.0	19146.0	19131.0	2000.0	36681.0	10000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
29995	29996	220000.0	1	3	1	39	0	0	0	0	...	88004.0	31237.0	15980.0	8500.0	20000.0	5003
29996	29997	150000.0	1	3	2	43	-1	-1	-1	-1	...	8979.0	5190.0	0.0	1837.0	3526.0	8998
29997	29998	30000.0	1	2	2	37	4	3	2	-1	...	20878.0	20582.0	19357.0	0.0	0.0	22000
29998	29999	80000.0	1	3	1	41	1	-1	0	0	...	52774.0	11855.0	48944.0	85900.0	3409.0	1178
29999	30000	50000.0	1	2	1	46	0	0	0	0	...	36535.0	32428.0	15313.0	2078.0	1800.0	1430

30000 rows × 25 columns

```
plt.figure(figsize=(10,5))
sns.countplot(x = 'IsDefaulter', data = df)
```

```
<AxesSubplot:xlabel='IsDefaulter', ylabel='count'>
```




```
df['IsDefaulter'].value_counts()
```

```
0    23364
1     6636
Name: IsDefaulter, dtype: int64
```

As we can see from above graph that both classes are not in proportion and we have imbalanced dataset.

Independent Variable:

Categorical Features

We have few categorical features in our dataset. Let'Check how they are related with out target class.

SEX

- 1 - Male
- 2 - Female

```
df['SEX'].value_counts()
```

```
2    18112
1    11888
Name: SEX, dtype: int64
```

Education

> 1 = graduate school; 2 = university; 3 = high school; 4 = others

```
df['EDUCATION'].value_counts()
```

```
2    14030
1    10585
3     4917
```

```
5      280
4      123
6       51
0       14
Name: EDUCATION, dtype: int64
```

As we can see in dataset we have values like 5,6,0 as well for which we are not having description so we can add up them in 4, which is Others.

```
fil = (df['EDUCATION'] == 5) | (df['EDUCATION'] == 6) | (df['EDUCATION'] == 0)
df.loc[fil, 'EDUCATION'] = 4
df['EDUCATION'].value_counts()
```

```
2    14030
1    10585
3     4917
4      468
Name: EDUCATION, dtype: int64
```

Marriage

> 1 = married; 2 = single; 3 = others

```
df['MARRIAGE'].value_counts()
```

```
2    15964
1    13659
3      323
0       54
Name: MARRIAGE, dtype: int64
```

We have few values for 0, which are not determined . So I am adding them in Others category.

```
fil = df['MARRIAGE'] == 0
df.loc[fil, 'MARRIAGE'] = 3
df['MARRIAGE'].value_counts()
```

```
2    15964
1    13659
3      377
Name: MARRIAGE, dtype: int64
```

Plotting our categorical features

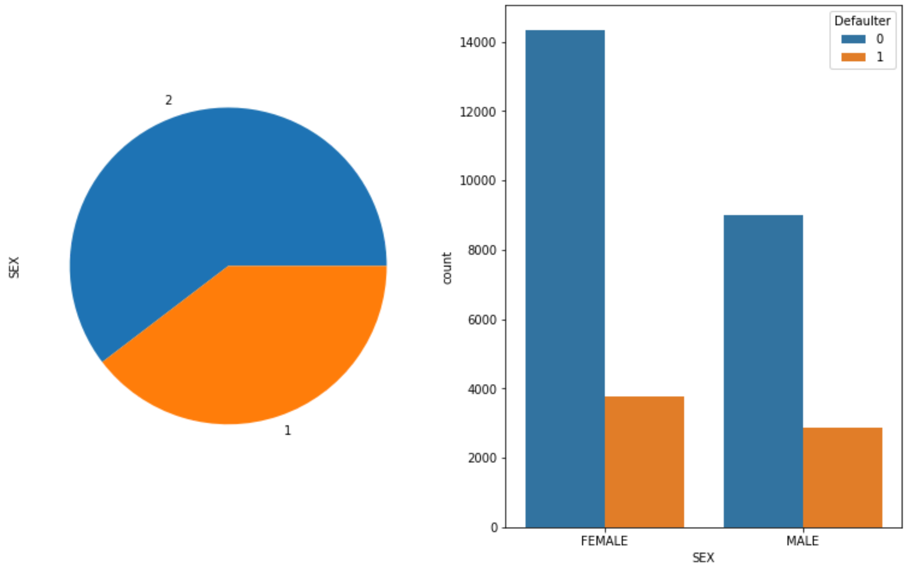
```
categorical_features = ['SEX', 'EDUCATION', 'MARRIAGE']
```

```
df_cat = df[categorical_features]
df_cat['Defaulter'] = df['IsDefaulter']
```

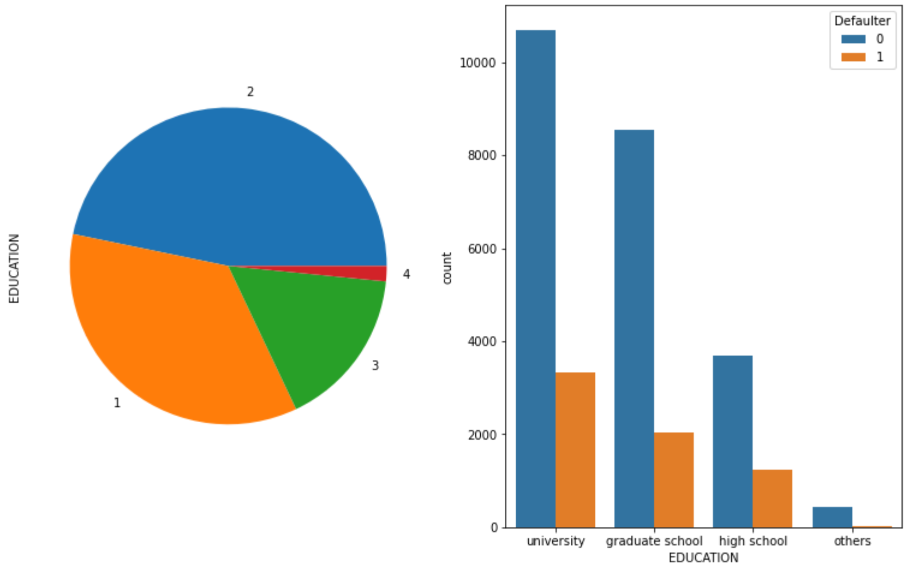
```
df_cat.replace({'SEX': {1 : 'MALE', 2 : 'FEMALE'}, 'EDUCATION' : {1 : 'graduate school', 2 : 'university', 3 : 'high school', 4 : 'others'}, 'MARRIAGE'
```

```
for col in categorical_features:
  plt.figure(figsize=(10,5))
  fig, axes = plt.subplots(ncols=2,figsize=(13,8))
  df[col].value_counts().plot(kind="pie",ax = axes[0],subplots=True)
  sns.countplot(x = col, hue = 'Defaulter', data = df_cat)
```

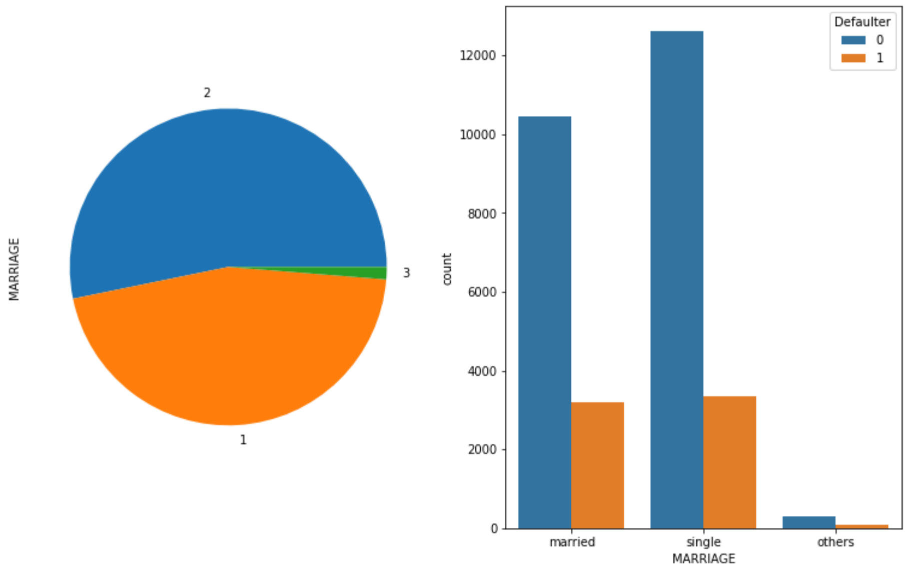
```
<Figure size 720x360 with 0 Axes>
```




```
<Figure size 720x360 with 0 Axes>
```




```
<Figure size 720x360 with 0 Axes>
```



Below are few observations for categorical features:

- There are more females credit card holder,so no. of defaulter have high proportion of females.
- No. of defaulters have a higher proportion of educated people (graduate school and university)
- No. of defaulters have a higher proportion of Singles.

Limit Balance

```
df['LIMIT_BAL'].max()
```

```
1000000.0
```

```
df['LIMIT_BAL'].min()
```

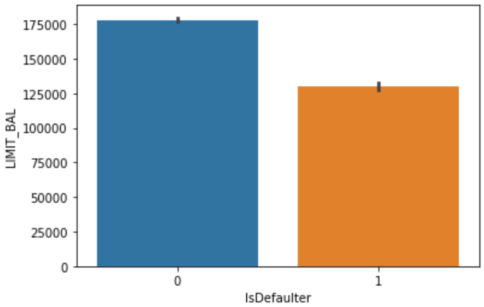
```
10000.0
```

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

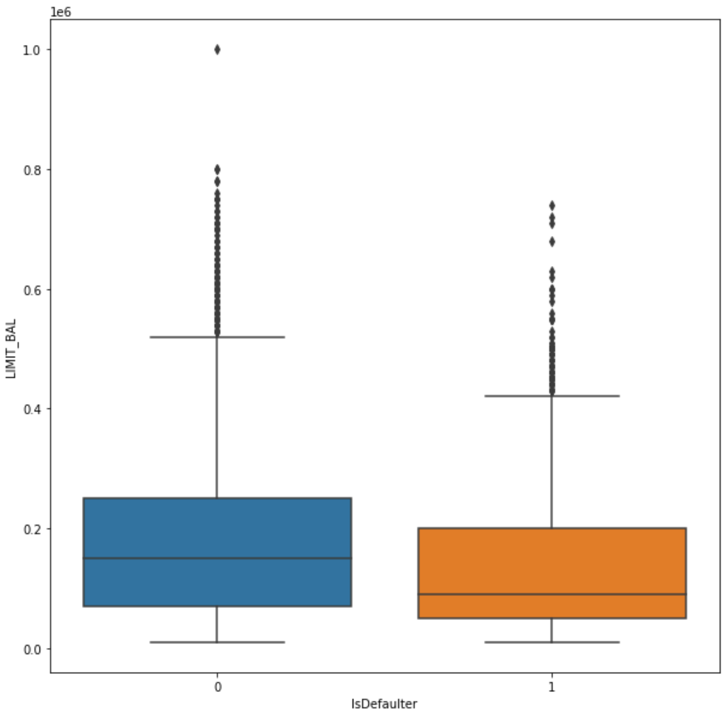
```
count      30000.000000
mean      167484.322667
std       129747.661567
min        10000.000000
25%        50000.000000
50%       140000.000000
75%       240000.000000
max      1000000.000000
Name: LIMIT_BAL, dtype: float64
```

```
sns.barplot(x='IsDefaulter', y='LIMIT_BAL', data=df)
```

```
<AxesSubplot:xlabel='IsDefaulter', ylabel='LIMIT_BAL'>
```




```
plt.figure(figsize=(10,10))
ax = sns.boxplot(x="IsDefaulter", y="LIMIT_BAL", data=df)
```




```
#renaming columns

df.rename(columns={'PAY_0':'PAY_SEPT','PAY_2':'PAY_AUG','PAY_3':'PAY_JUL','PAY_4':'PAY_JUN','PAY_5':'PAY_MAY','PAY_6':'PAY_APR'},inplace=True)
df.rename(columns={'BILL_AMT1':'BILL_AMT_SEPT','BILL_AMT2':'BILL_AMT_AUG','BILL_AMT3':'BILL_AMT_JUL','BILL_AMT4':'BILL_AMT_JUN','BILL_AMT5':'BILL_AMT_M
df.rename(columns={'PAY_AMT1':'PAY_AMT_SEPT','PAY_AMT2':'PAY_AMT_AUG','PAY_AMT3':'PAY_AMT_JUL','PAY_AMT4':'PAY_AMT_JUN','PAY_AMT5':'PAY_AMT_MAY','PAY_A
```

```
df.head()
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_SEPT	PAY_AUG	PAY_JUL	PAY_JUN	...	BILL_AMT_MAY	BILL_AMT_APR	PAY_AMT_SEPT	PAY_AMT_AUG	PAY
0	1	20000.0	2	2	1	24	2	2	-1	-1	...	0.0	0.0	0.0	689.0	
1	2	120000.0	2	2	2	26	-1	2	0	0	...	3455.0	3261.0	0.0	1000.0	
2	3	90000.0	2	2	2	34	0	0	0	0	...	14948.0	15549.0	1518.0	1500.0	
3	4	50000.0	2	2	1	37	0	0	0	0	...	28959.0	29547.0	2000.0	2019.0	
4	5	50000.0	1	2	1	57	-1	0	-1	0	...	19146.0	19131.0	2000.0	36681.0	

5 rows × 26 columns

AGE

Plotting graph of number of ages of all people with credit card irrespective of gender.

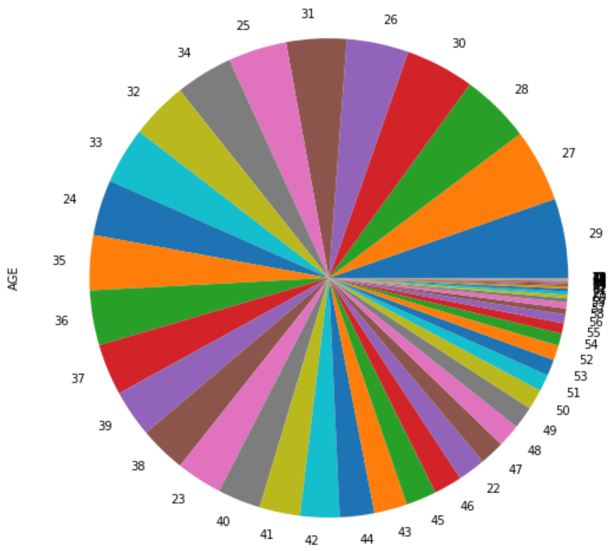
```
df['AGE'].value_counts()
```

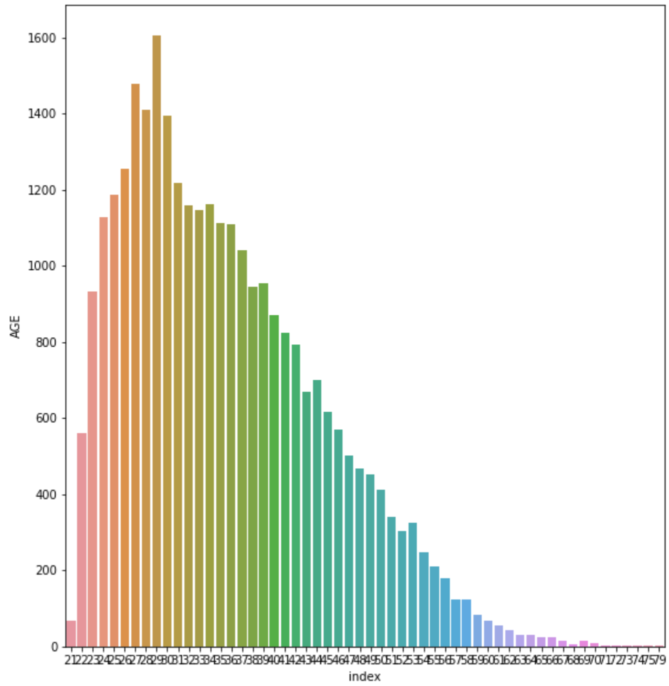
```
29    1605
27    1477
28    1409
30    1395
26    1256
31    1217
25    1186
34    1162
32    1158
33    1146
24    1127
35    1113
36    1108
37    1041
39     954
38     944
23     931
40     870
41     824
42     794
44     700
43     670
45     617
46     570
22     560
47     501
48     466
49     452
50     411
51     340
53     325
52     304
54     247
55     209
56     178
58     122
57     122
59      83
60      67
21      67
61      56
62      44
63      31
64      31
66      25
65      24
67      16
69      15
70      10
68       5
73       4
72       3
75       3
71       3
79       1
74       1
Name: AGE, dtype: int64
```

```
df['AGE']=df['AGE'].astype('int')
```

```
fig, axes = plt.subplots(ncols=2,figsize=(20,10))
Day_df=df['AGE'].value_counts().reset_index()
df['AGE'].value_counts().plot(kind="pie",ax = axes[0],subplots=True)
sns.barplot(x='index',y='AGE',data=Day_df,ax = axes[1],orient='v')
```

```
<AxesSubplot:xlabel='index', ylabel='AGE'>
```



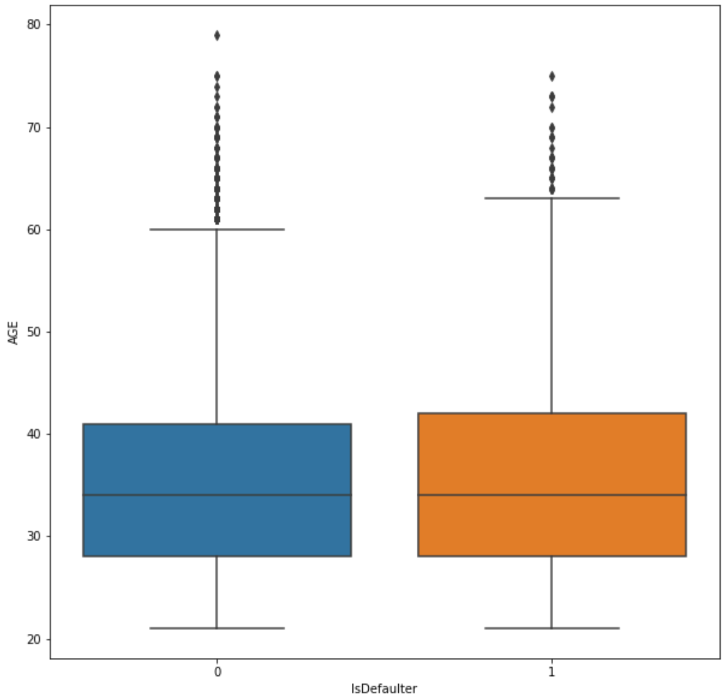



```
df.groupby('IsDefaulter')['AGE'].mean()
```

```
IsDefaulter
0    35.417266
1    35.725738
Name: AGE, dtype: float64
```

```
df = df.astype('int')
```

```
plt.figure(figsize=(10,10))
ax = sns.boxplot(x="IsDefaulter", y="AGE", data=df)
```

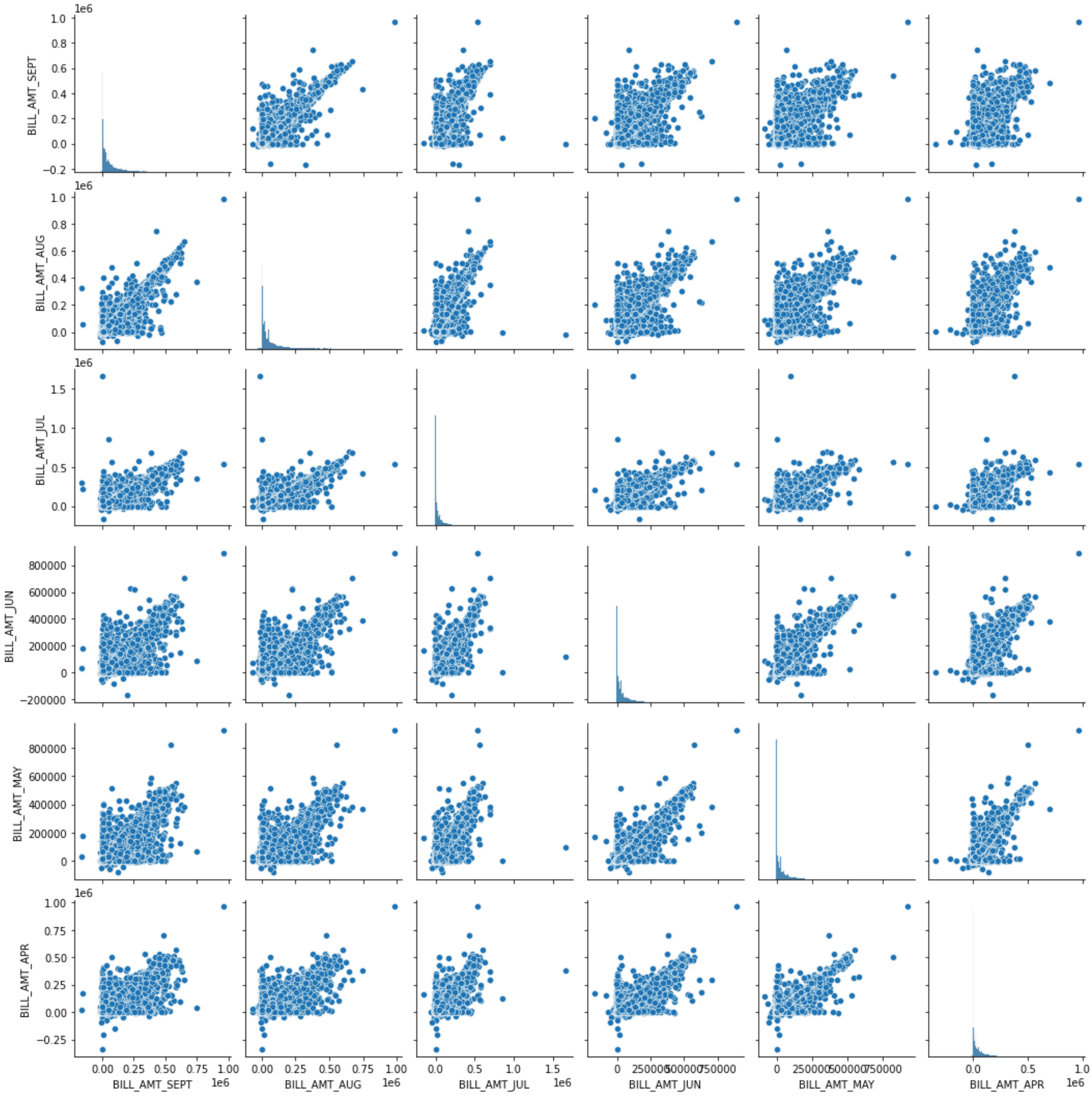


Bill Amount

```
bill_amnt_df = df[['BILL_AMT_SEPT', 'BILL_AMT_AUG', 'BILL_AMT_JUL', 'BILL_AMT_JUN', 'BILL_AMT_MAY', 'BILL_AMT_APR']]
```

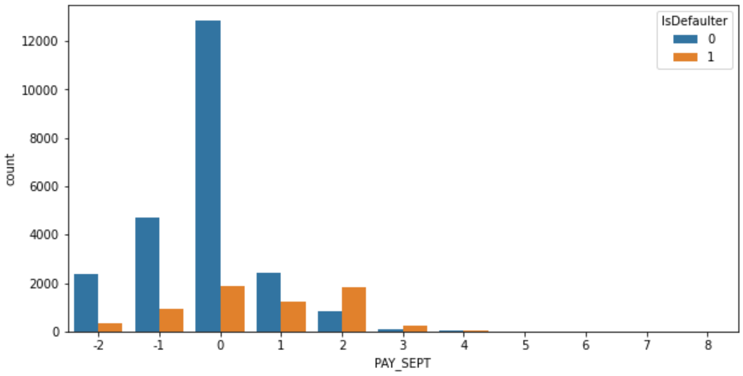
```
sns.pairplot(data = bill_amnt_df)
```

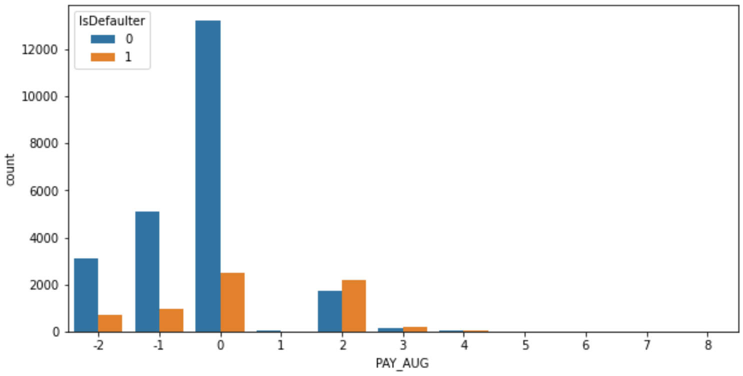
```
<seaborn.axisgrid.PairGrid at 0x7fe50e04e750>
```

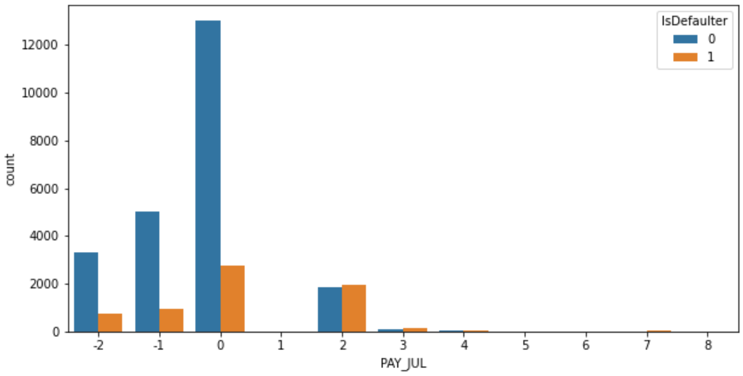


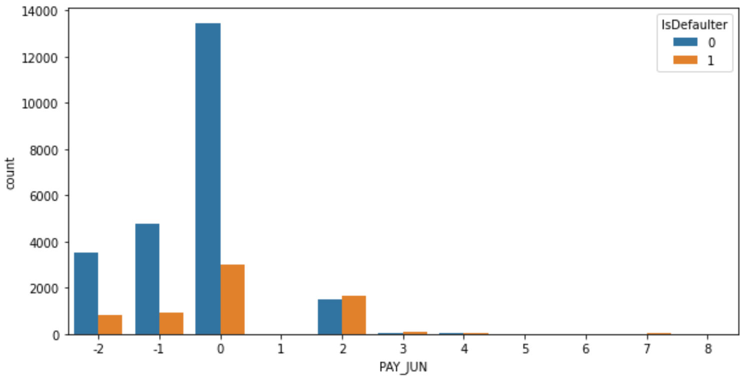
History payment status

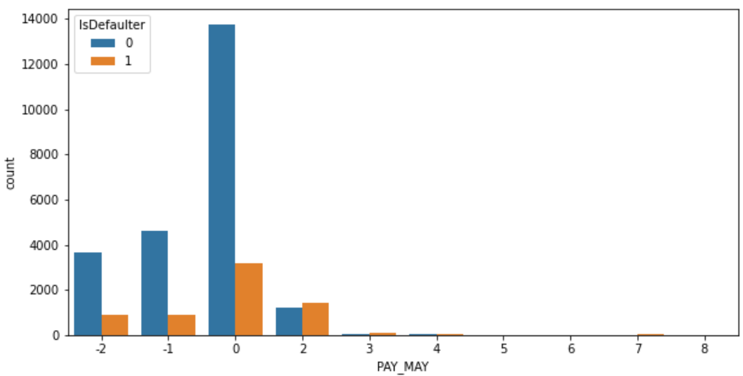
```
pay_col = ['PAY_SEPT',  'PAY_AUG',  'PAY_JUL',  'PAY_JUN',  'PAY_MAY',  'PAY_APR']
for col in pay_col:
  plt.figure(figsize=(10,5))
  sns.countplot(x = col, hue = 'IsDefaulter', data = df)
```


IsDefaulter
0
1
count
PAY_SEPT

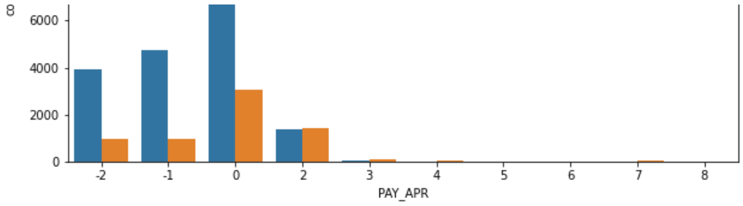

IsDefaulter
0
1
count
PAY_AUG


IsDefaulter
0
1
count
PAY_JUL


IsDefaulter
0
1
count
PAY_JUN


IsDefaulter
0
1
count
PAY_MAY


IsDefaulter
0
1

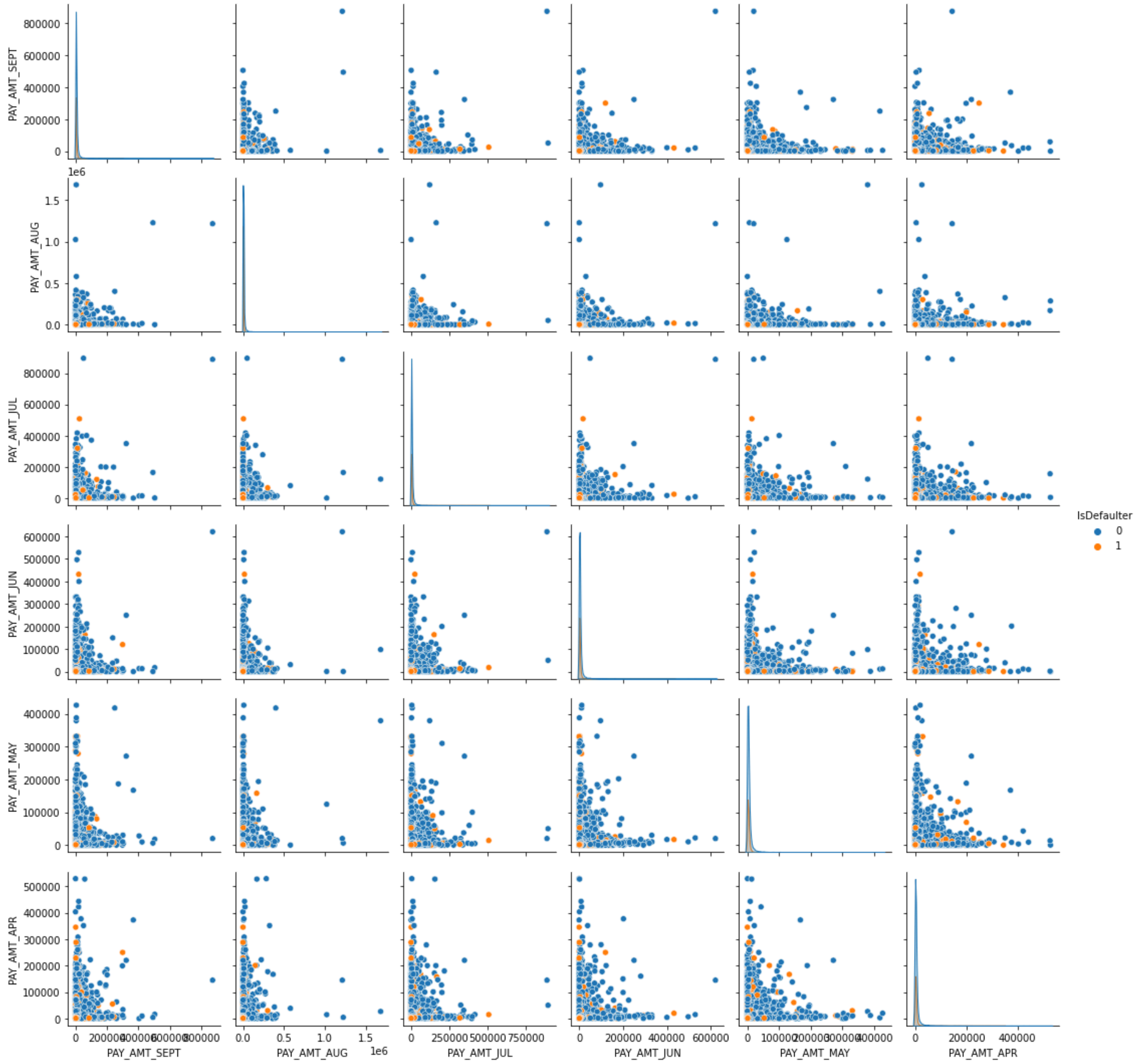


Paid Amount

```
pay_amnt_df = df[['PAY_AMT_SEPT',  'PAY_AMT_AUG',  'PAY_AMT_JUL',  'PAY_AMT_JUN',  'PAY_AMT_MAY',  'PAY_AMT_APR', 'IsDefaulter']]
```

```
sns.pairplot(data = pay_amnt_df, hue='IsDefaulter')
```

```
<seaborn.axisgrid.PairGrid at 0x7fe5030f47d0>
```




```
df.shape
```

```
(30000, 26)
```

As we have seen earlier that we have imbalanced dataset. So to remediate Imbalance we are using SMOTE(Synthetic Minority Oversampling Technique)

```
from imblearn.over_sampling import SMOTE
```

```
smote = SMOTE()
```

```
# fit predictor and target variable
```

```
x_smote, y_smote = smote.fit_resample(df.iloc[:,0:-1], df['IsDefaulter'])

print('Original dataset shape', len(df))
print('Resampled dataset shape', len(y_smote))
```

```
Original dataset shape 30000
Resampled dataset shape 46728
```

```
x_smote
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_SEPT	PAY_AUG	PAY_JUL	PAY_JUN	...	BILL_AMT_JUN	BILL_AMT_MAY	BILL_AMT_APR	PAY_AMT_:
0	1	20000	2	2	1	24	2	2	-1	-1	...	0	0	0	
1	2	120000	2	2	2	26	-1	2	0	0	...	3272	3455	3261	
2	3	90000	2	2	2	34	0	0	0	0	...	14331	14948	15549	1
3	4	50000	2	2	1	37	0	0	0	0	...	28314	28959	29547	2
4	5	50000	1	2	1	57	-1	0	-1	0	...	20940	19146	19131	2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
46723	20094	50000	1	2	1	32	1	2	0	0	...	25764	25264	25747	
46724	12912	90000	2	2	1	31	0	0	0	0	...	68726	64845	66464	3
46725	14122	240000	1	2	1	29	-1	-1	-1	-1	...	3822	3444	3646	4
46726	5969	120000	2	1	2	27	0	-1	-1	-1	...	0	240	120	
46727	17790	30000	1	2	1	30	3	2	2	7	...	2398	2398	2398	

46728 rows × 25 columns

```
columns = list(df.columns)
```

```
columns.pop()
```

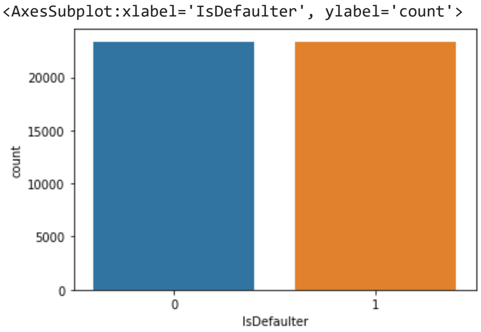
```
'IsDefaulter'
```

```
balance_df = pd.DataFrame(x_smote, columns=columns)
```

```
balance_df['IsDefaulter'] = y_smote
```

```
sns.countplot('IsDefaulter', data = balance_df)
```

```
<AxesSubplot:xlabel='IsDefaulter', ylabel='count'>
```




```
balance_df[balance_df['IsDefaulter']==1]
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_SEPT	PAY_AUG	PAY_JUL	PAY_JUN	...	BILL_AMT_MAY	BILL_AMT_APR	PAY_AMT_SEPT	PAY_AMT_
0	1	20000	2	2	1	24	2	2	-1	-1	...	0	0	0	0
1	2	120000	2	2	2	26	-1	2	0	0	...	3455	3261	0	10
13	14	70000	1	2	2	30	1	2	2	0	...	36137	36894	3200	
16	17	20000	1	1	2	24	0	0	2	2	...	17905	19104	3200	
21	22	120000	2	2	1	39	-1	-1	-1	-1	...	632	316	316	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
46723	20094	50000	1	2	1	32	1	2	0	0	...	25264	25747	0	14
46724	12912	90000	2	2	1	31	0	0	0	0	...	64845	66464	3000	32
46725	14122	240000	1	2	1	29	-1	-1	-1	-1	...	3444	3646	4293	24
46726	5969	120000	2	1	2	27	0	-1	-1	-1	...	240	120	120	1
46727	17790	30000	1	2	1	30	3	2	2	7	...				

23364 rows × 26 columns

Feature Engineering

```
df_fr = balance_df.copy()
```

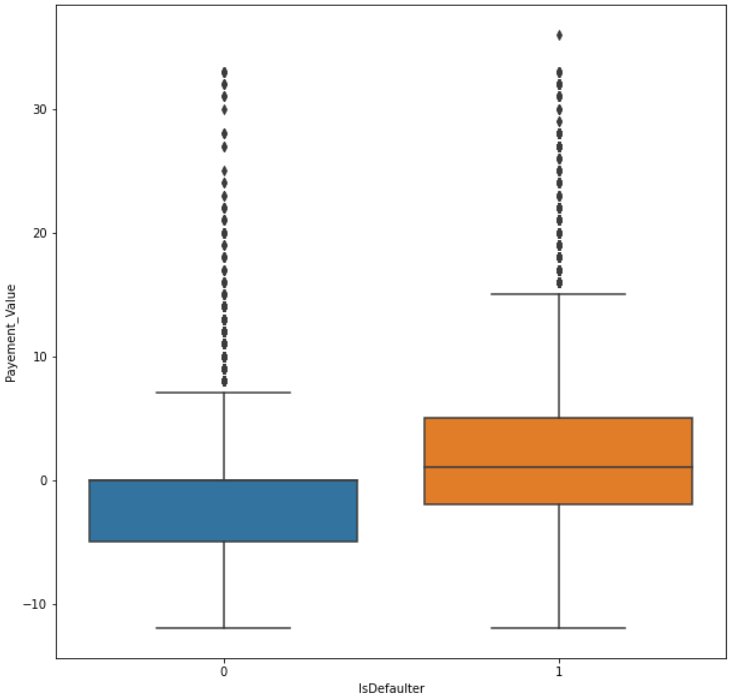
```
df_fr['Payement_Value'] = df_fr['PAY_SEPT'] + df_fr['PAY_AUG'] + df_fr['PAY_JUL'] + df_fr['PAY_JUN'] + df_fr['PAY_MAY'] + df_fr['PAY_APR']
```

```
df_fr.groupby('IsDefaulter')['Payement_Value'].mean()
```

```
IsDefaulter
0   -1.980140
1    1.656608
Name: Payement_Value, dtype: float64
```

```
plt.figure(figsize=(10,10))
sns.boxplot(data = df_fr, x = 'IsDefaulter', y = 'Payement_Value' )
```

```
<AxesSubplot:xlabel='IsDefaulter', ylabel='Payement_Value'>
```




```
df_fr['Dues'] = (df_fr['BILL_AMT_APR']+df_fr['BILL_AMT_MAY']+df_fr['BILL_AMT_JUN']+df_fr['BILL_AMT_JUL']+df_fr['BILL_AMT_SEPT'])-(df_fr['PAY_AMT_APR']+
```

```
df_fr.groupby('IsDefaulter')['Dues'].mean()
```

```
IsDefaulter
0    187742.051532
1    198226.812789
Name: Dues, dtype: float64
```

```
df_fr['EDUCATION'].unique()
```

```
array([2, 1, 3, 4])
```

```
df_fr['EDUCATION']=np.where(df_fr['EDUCATION'] == 6, 4, df_fr['EDUCATION'])
df_fr['EDUCATION']=np.where(df_fr['EDUCATION'] == 0, 4, df_fr['EDUCATION'])
```

```
df_fr['MARRIAGE'].unique()
```

```
array([1, 2, 3])
```

```
df_fr['MARRIAGE']=np.where(df_fr['MARRIAGE'] == 0, 3, df_fr['MARRIAGE'])
```

```
df_fr.replace({'SEX': {1 : 'MALE', 2 : 'FEMALE'}, 'EDUCATION' : {1 : 'graduate school', 2 : 'university', 3 : 'high school', 4 : 'others'}, 'MARRIAGE'
```

```
df_fr.head()
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_SEPT	PAY_AUG	PAY_JUL	PAY_JUN	...	PAY_AMT_SEPT	PAY_AMT_AUG	PAY_AMT_JUL	PAY_AMT_JUN	P
0	1	20000	FEMALE	university	married	24	2	2	-1	-1	...	0	689	0	0	
1	2	120000	FEMALE	university	single	26	-1	2	0	0	...	0	1000	1000	1000	
2	3	90000	FEMALE	university	single	34	0	0	0	0	...	1518	1500	1000	1000	
3	4	50000	FEMALE	university	married	37	0	0	0	0	...	2000	2019	1200	1100	
4	5	50000	MALE	university	married	57	-1	0	-1	0	...	2000	36681	10000	9000	

5 rows × 28 columns

One Hot Encoding

```
df_fr = pd.get_dummies(df_fr,columns=['EDUCATION','MARRIAGE'])
```

```
df_fr.head()
```

	ID	LIMIT_BAL	SEX	AGE	PAY_SEPT	PAY_AUG	PAY_JUL	PAY_JUN	PAY_MAY	PAY_APR	...	IsDefaulter	Payement_Value	Dues	EDUCATION_graduate school
0	1	20000	FEMALE	24	2	2	-1	-1	-2	-2	...	1	-2	3913	0
1	2	120000	FEMALE	26	-1	2	0	0	0	2	...	1	3	10352	0
2	3	90000	FEMALE	34	0	0	0	0	0	0	...	0	0	76608	0
3	4	50000	FEMALE	37	0	0	0	0	0	0	...	0	0	174713	0
4	5	50000	MALE	57	-1	0	-1	0	0	0	...	0	-2	44620	0

5 rows × 33 columns

```
df_fr.drop(['EDUCATION_others','MARRIAGE_others'],axis = 1, inplace = True)
```

```
df_fr = pd.get_dummies(df_fr, columns = ['PAY_SEPT',   'PAY_AUG',  'PAY_JUL',  'PAY_JUN',  'PAY_MAY',  'PAY_APR'], drop_first = True )
```

```
df_fr.head()
```

	ID	LIMIT_BAL	SEX	AGE	BILL_AMT_SEPT	BILL_AMT_AUG	BILL_AMT_JUL	BILL_AMT_JUN	BILL_AMT_MAY	BILL_AMT_APR	...	PAY_APR_-1	PAY_APR_0	PAY_
0	1	20000	FEMALE	24	3913	3102	689	0	0	0	...	0	0	
1	2	120000	FEMALE	26	2682	1725	2682	3272	3455	3261	...	0	0	
2	3	90000	FEMALE	34	29239	14027	13559	14331	14948	15549	...	0	1	
3	4	50000	FEMALE	37	46990	48233	49291	28314	28959	29547	...	0	1	
4	5	50000	MALE	57	8617	5670	35835	20940	19146	19131	...	0	1	

5 rows × 85 columns

```
# LABEL ENCODING FOR SEX
encoders_nums = {
                 "SEX":{"FEMALE": 0, "MALE": 1}
}
df_fr = df_fr.replace(encoders_nums)
```

```
df_fr.head()
```

	ID	LIMIT_BAL	SEX	AGE	BILL_AMT_SEPT	BILL_AMT_AUG	BILL_AMT_JUL	BILL_AMT_JUN	BILL_AMT_MAY	BILL_AMT_APR	...	PAY_APR_-1	PAY_APR_0	PAY_APR_
0	1	20000	0	24	3913	3102	689	0	0	0	...	0	0	
1	2	120000	0	26	2682	1725	2682	3272	3455	3261	...	0	0	
2	3	90000	0	34	29239	14027	13559	14331	14948	15549	...	0	1	
3	4	50000	0	37	46990	48233	49291	28314	28959	29547	...	0	1	
4	5	50000	1	57	8617	5670	35835	20940	19146	19131	...	0	1	

5 rows × 85 columns

```
df_fr.drop('ID',axis = 1, inplace = True)
```

```
df_fr.to_csv('Final_df.csv')
```

```
df_fr = pd.read_csv('./Final_df.csv')
```

```
df_fr.head()
```

	Unnamed: 0	LIMIT_BAL	SEX	AGE	BILL_AMT_SEPT	BILL_AMT_AUG	BILL_AMT_JUL	BILL_AMT_JUN	BILL_AMT_MAY	BILL_AMT_APR	...	PAY_APR_-1	PAY_APR_0
0	0	20000	0	24	3913	3102	689	0	0	0	...	0	0
1	1	120000	0	26	2682	1725	2682	3272	3455	3261	...	0	0
2	2	90000	0	34	29239	14027	13559	14331	14948	15549	...	0	1
3	3	50000	0	37	46990	48233	49291	28314	28959	29547	...	0	1
4	4	50000	1	57	8617	5670	35835	20940	19146	19131	...	0	1

5 rows × 85 columns

```
df_fr.drop(['Unnamed: 0'],axis = 1, inplace = True)
```

Implementing Logistic Regression

Logistic Regression is one of the simplest algorithms which estimates the relationship between one dependent binary variable and independent variables, computing the probability of occurrence of an event. The regulation parameter C controls the trade-off between increasing complexity (overfitting) and keeping the model simple (underfitting). For large values of C, the power of regulation is reduced and the model increases its complexity, thus overfitting the data.

```
df_log_reg = df_fr.copy()
```

```
df_log_reg.head()
```

	LIMIT_BAL	SEX	AGE	BILL_AMT_SEPT	BILL_AMT_AUG	BILL_AMT_JUL	BILL_AMT_JUN	BILL_AMT_MAY	BILL_AMT_APR	PAY_AMT_SEPT	...	PAY_APR_-1	PAY_APR_
0	20000	0	24	3913	3102	689	0	0	0	0	...	0	
1	120000	0	26	2682	1725	2682	3272	3455	3261	0	...	0	
2	90000	0	34	29239	14027	13559	14331	14948	15549	1518	...	0	
3	50000	0	37	46990	48233	49291	28314	28959	29547	2000	...	0	
4	50000	1	57	8617	5670	35835	20940	19146	19131	2000	...	0	

5 rows × 84 columns

```
X = df_log_reg.drop(['IsDefaulter','Payement_Value','Dues'],axis=1)
y = df_log_reg['IsDefaulter']
```

```
columns = X.columns
```

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42, stratify = y)
```

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

```
grid_lr_clf = GridSearchCV(LogisticRegression(), param_grid, scoring = 'accuracy', n_jobs = -1, verbose = 3, cv = 3)
grid_lr_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 14 candidates, totalling 42 fits
GridSearchCV(cv=3, estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
                         'penalty': ['l1', 'l2']},
             scoring='accuracy', verbose=3)
```

```
optimized_clf = grid_lr_clf.best_estimator_
```

```
grid_lr_clf.best_params_
```

```
{'C': 0.01, 'penalty': 'l2'}
```

```
grid_lr_clf.best_score_
```

```
1.0
```

```
# Predicted Probability
train_preds = optimized_clf.predict_proba(X_train)[:,1]
test_preds = optimized_clf.predict_proba(X_test)[:,1]
```

```
# Get the predicted classes
train_class_preds = optimized_clf.predict(X_train)
test_class_preds = optimized_clf.predict(X_test)


# Get the accuracy scores
train_accuracy_lr = accuracy_score(train_class_preds,y_train)
test_accuracy_lr = accuracy_score(test_class_preds,y_test)

print("The accuracy on train data is ", train_accuracy_lr)
print("The accuracy on test data is ", test_accuracy_lr)
```

```
The accuracy on train data is  1.0
The accuracy on test data is  1.0
```

```
test_accuracy_lr = accuracy_score(test_class_preds,y_test)
test_precision_score_lr = precision_score(test_class_preds,y_test)
test_recall_score_lr = recall_score(test_class_preds,y_test)
test_f1_score_lr = f1_score(test_class_preds,y_test)
test_roc_score_lr = roc_auc_score(test_class_preds,y_test)

print("The accuracy on test data is ", test_accuracy_lr)
print("The precision on test data is ", test_precision_score_lr)
print("The recall on test data is ", test_recall_score_lr)
print("The f1 on test data is ", test_f1_score_lr)
print("The roc_score on test data is ", test_roc_score_lr)
```

```
The accuracy on test data is  1.0
The precision on test data is  1.0
The recall on test data is  1.0
The f1 on test data is  1.0
The roc_score on test data is  1.0
```

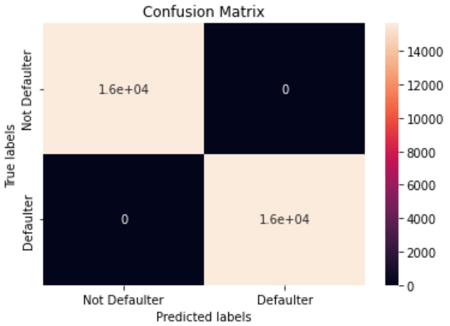
```
# Get the confusion matrix for both train and test

labels = ['Not Defaulter', 'Defaulter']
cm = confusion_matrix(y_train, train_class_preds)
print(cm)

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax) #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)
```

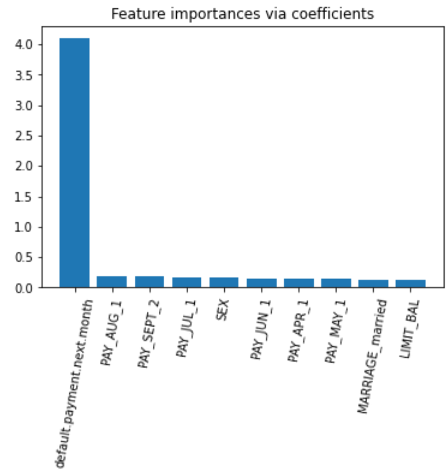
```
[[15653     0]
 [    0 15654]]
[Text(0, 0.5, 'Not Defaulter'), Text(0, 1.5, 'Defaulter')]
```




```
feature_importance = pd.DataFrame({'Features':columns, 'Importance':np.abs(optimized_clf.coef_).ravel() })
```

```
feature_importance = feature_importance.sort_values(by = 'Importance', ascending=False)[:10]
```

```
plt.bar(height=feature_importance['Importance'], x= feature_importance['Features'])
plt.xticks(rotation=80)
plt.title("Feature importances via coefficients")
plt.show()
```

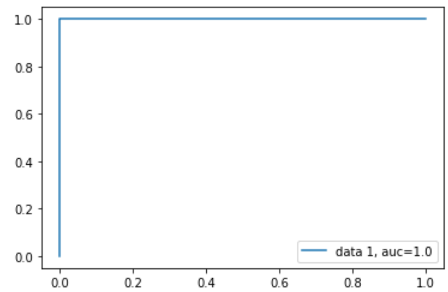



```
y_preds_proba_lr = optimized_clf.predict_proba(X_test)[::,1]
```

```
'''
y_pred_proba = y_preds_proba_lr
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()'''
```

```
'\ny_pred_proba = y_preds_proba_lr\nfpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)\nauc = metrics.roc_auc_score(y_test,
y_pred_proba)\nplt.plot(fpr,tpr,label="data 1, auc="+str(auc))\nplt.legend(loc=4)\nplt.show()'
```

```
y_pred_proba = y_preds_proba_lr
fpr, tpr, _ = roc_curve(y_test,  y_pred_proba)
auc = roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



We have implemented logistic regression and we getting f1-sore approx 73%. As we have imbalanced dataset, F1- score is better parameter. Let's go ahead with other models and see if they can yield better result.

Implementing SVC

```
from sklearn.model_selection import GridSearchCV
```

```
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100],
              'kernel': ['rbf']}
```

```
X = df_fr.drop(['IsDefaulter','Payement_Value','Dues'],axis=1)
y = df_fr['IsDefaulter']
```

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42, stratify = y)
```

```
grid_clf = GridSearchCV(SVC(probability=True), param_grid, scoring = 'accuracy', n_jobs = -1, verbose = 3, cv = 3)
grid_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
GridSearchCV(cv=3, estimator=SVC(probability=True), n_jobs=-1,
             param_grid={'C': [0.1, 1, 10, 100], 'kernel': ['rbf']},
             scoring='accuracy', verbose=3)
```

```
optimal_SVC_clf = grid_clf.best_estimator_
```

```
grid_clf.best_params_
```

```
{'C': 100, 'kernel': 'rbf'}
```

```
grid_clf.best_score_
```

```
0.996294775698412
```

```
# Get the predicted classes
train_class_preds = optimal_SVC_clf.predict(X_train)
test_class_preds = optimal_SVC_clf.predict(X_test)
```

```
# Get the accuracy scores
train_accuracy_SVC = accuracy_score(train_class_preds,y_train)
test_accuracy_SVC = accuracy_score(test_class_preds,y_test)

print("The accuracy on train data is ", train_accuracy_lr)
print("The accuracy on test data is ", test_accuracy_lr)
```

```
The accuracy on train data is  1.0
The accuracy on test data is  1.0
```

```
test_accuracy_SVC = accuracy_score(test_class_preds,y_test)
test_precision_score_SVC = precision_score(test_class_preds,y_test)
test_recall_score_SVC = recall_score(test_class_preds,y_test)
test_f1_score_SVC = f1_score(test_class_preds,y_test)
test_roc_score_SVC = roc_auc_score(test_class_preds,y_test)

print("The accuracy on test data is ", test_accuracy_SVC)
print("The precision on test data is ", test_precision_score_SVC)
print("The recall on test data is ", test_recall_score_SVC)
print("The f1 on test data is ", test_f1_score_SVC)
print("The roc_score on test data is ", test_roc_score_SVC)
```

```
The accuracy on test data is  0.9966928214772064
The precision on test data is  0.9989623865110246
The recall on test data is  0.994448030987734
The f1 on test data is  0.996700097055969
The roc_score on test data is  0.9967029107518138
```

We can see from above results that we are getting around 80% train accuracy and 78% for test accuracy which is not bad. But f1- score is 76% approx, so there might be more ground for improvement.

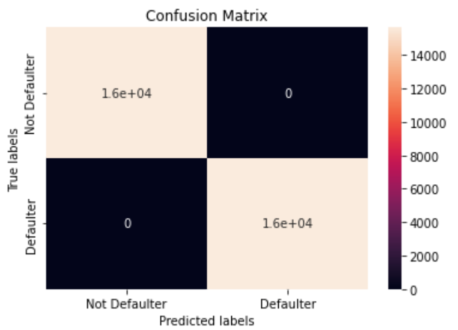
```
 # Get the confusion matrix for both train and test

labels = ['Not Defaulter', 'Defaulter']
cm = confusion_matrix(y_train, train_class_preds)
print(cm)

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax) #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)
```

```
[[15653     0]
 [    0 15654]]
[Text(0, 0.5, 'Not Defaulter'), Text(0, 1.5, 'Defaulter')]
```




```
import torch
```

```
model_save_name = 'SVC_optimized_classifier.pt'
path = F"./{model_save_name}"
torch.save(optimal_SVC_clf, path)
```

```
model_save_name = 'SVC_optimized_classifier.pt'
path = F"./{model_save_name}"
optimal_SVC_clf = torch.load(path)
```

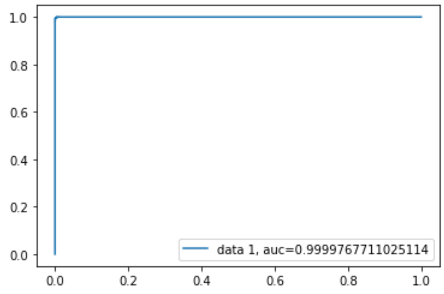
```
optimal_SVC_clf
```

```
SVC(C=100, probability=True)
```

```
# Get the predicted classes
train_class_preds = optimal_SVC_clf.predict(X_train)
test_class_preds = optimal_SVC_clf.predict(X_test)
```

```
y_pred_proba_SVC = optimal_SVC_clf.predict_proba(X_test)[::,1]
```

```
# ROC AUC CURVE
fpr, tpr, _ = roc_curve(y_test,  y_pred_proba_SVC)
auc = roc_auc_score(y_test, y_pred_proba_SVC)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Implementing Decision Tree

Decision Tree is another very popular algorithm for classification problems because it is easy to interpret and understand. An internal node represents a feature, the branch represents a decision rule, and each leaf node represents the outcome. Some advantages of decision trees are that they require less data preprocessing, i.e., no need to normalize features. However, noisy data can be easily overfitted and results in biased results when the data set is imbalanced.

```
param_grid = {'max_depth': [20,30,50,100], 'min_samples_split':[0.1,0.2,0.4]}
```

```
from sklearn.tree import DecisionTreeClassifier
```

```
X = df_fr.drop(['IsDefaulter','Payement_Value','Dues'],axis=1)
y = df_fr['IsDefaulter']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42, stratify = y)
```

```
grid_DTC_clf = GridSearchCV(DecisionTreeClassifier(), param_grid, scoring = 'accuracy', n_jobs = -1, verbose = 3, cv = 3)
grid_DTC_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'max_depth': [20, 30, 50, 100],
                         'min_samples_split': [0.1, 0.2, 0.4]},
             scoring='accuracy', verbose=3)
```

```
grid_DTC_clf.best_score_
```

```
1.0
```

```
optimal_DTC_clf = grid_DTC_clf.best_estimator_
```

```
# Get the predicted classes
train_class_preds = optimal_DTC_clf.predict(X_train)
test_class_preds = optimal_DTC_clf.predict(X_test)
```

```
grid_DTC_clf.best_params_
```

```
{'max_depth': 20, 'min_samples_split': 0.1}
```

```
# Get the accuracy scores
train_accuracy_DTC = accuracy_score(train_class_preds,y_train)
test_accuracy_DTC = accuracy_score(test_class_preds,y_test)
```

```
print("The accuracy on train data is ", train_accuracy_DTC)
print("The accuracy on test data is ", test_accuracy_DTC)
```

```
The accuracy on train data is  1.0
The accuracy on test data is  1.0
```

Implementing RandomForest

```
from sklearn.ensemble import RandomForestClassifier
```

```
X = df_fr.drop(['IsDefaulter','Payement_Value','Dues'],axis=1)
y = df_fr['IsDefaulter']
```

```
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train,y_train)
```

```
RandomForestClassifier()
```

```
# Get the predicted classes
train_class_preds = rf_clf.predict(X_train)
test_class_preds = rf_clf.predict(X_test)
```

```
# Get the accuracy scores
train_accuracy_rf = accuracy_score(train_class_preds,y_train)
test_accuracy_rf = accuracy_score(test_class_preds,y_test)

print("The accuracy on train data is ", train_accuracy_rf)
print("The accuracy on test data is ", test_accuracy_rf)
```

```
The accuracy on train data is  1.0
The accuracy on test data is  1.0
```

```
test_accuracy_rf = accuracy_score(test_class_preds,y_test)
test_precision_score_rf = precision_score(test_class_preds,y_test)
test_recall_score_rf = recall_score(test_class_preds,y_test)
test_f1_score_rf = f1_score(test_class_preds,y_test)
test_roc_score_rf = roc_auc_score(test_class_preds,y_test)

print("The accuracy on test data is ", test_accuracy_rf)
print("The precision on test data is ", test_precision_score_rf)
print("The recall on test data is ", test_recall_score_rf)
print("The f1 on test data is ", test_f1_score_rf)
print("The roc_score on test data is ", test_roc_score_rf)
```

```
The accuracy on test data is  1.0
The precision on test data is  1.0
The recall on test data is  1.0
The f1 on test data is  1.0
The roc_score on test data is  1.0
```

We can see from above results that we are getting around 99% train accuracy and 83% for test accuracy which depicts that model is overfitting. However our f1-score is around 82%, which is not bad.

```
param_grid = {'n_estimators': [100,150,200], 'max_depth': [10,20,30]}
```

```
grid_rf_clf = GridSearchCV(RandomForestClassifier(), param_grid, scoring = 'accuracy', n_jobs = -1, verbose = 3, cv = 3)
grid_rf_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
GridSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': [10, 20, 30],
                         'n_estimators': [100, 150, 200]},
             scoring='accuracy', verbose=3)
```

```
grid_rf_clf.best_score_
```

```
1.0
```

```
grid_rf_clf.best_params_
```

```
{'max_depth': 10, 'n_estimators': 200}
```

```
optimal_rf_clf = grid_rf_clf.best_estimator_
```

```
# Get the predicted classes
train_class_preds = optimal_rf_clf.predict(X_train)
test_class_preds = optimal_rf_clf.predict(X_test)
```

```
# Get the accuracy scores
train_accuracy_rf = accuracy_score(train_class_preds,y_train)
test_accuracy_rf = accuracy_score(test_class_preds,y_test)
```

```
print("The accuracy on train data is ", train_accuracy_rf)
print("The accuracy on test data is ", test_accuracy_rf)
```

```
The accuracy on train data is  1.0
The accuracy on test data is  1.0
```

```
test_accuracy_rf = accuracy_score(test_class_preds,y_test)
test_precision_score_rf = precision_score(test_class_preds,y_test)
test_recall_score_rf = recall_score(test_class_preds,y_test)
test_f1_score_rf = f1_score(test_class_preds,y_test)
test_roc_score_rf = roc_auc_score(test_class_preds,y_test)

print("The accuracy on test data is ", test_accuracy_rf)
print("The precision on test data is ", test_precision_score_rf)
print("The recall on test data is ", test_recall_score_rf)
print("The f1 on test data is ", test_f1_score_rf)
print("The roc_score on test data is ", test_roc_score_rf)
```

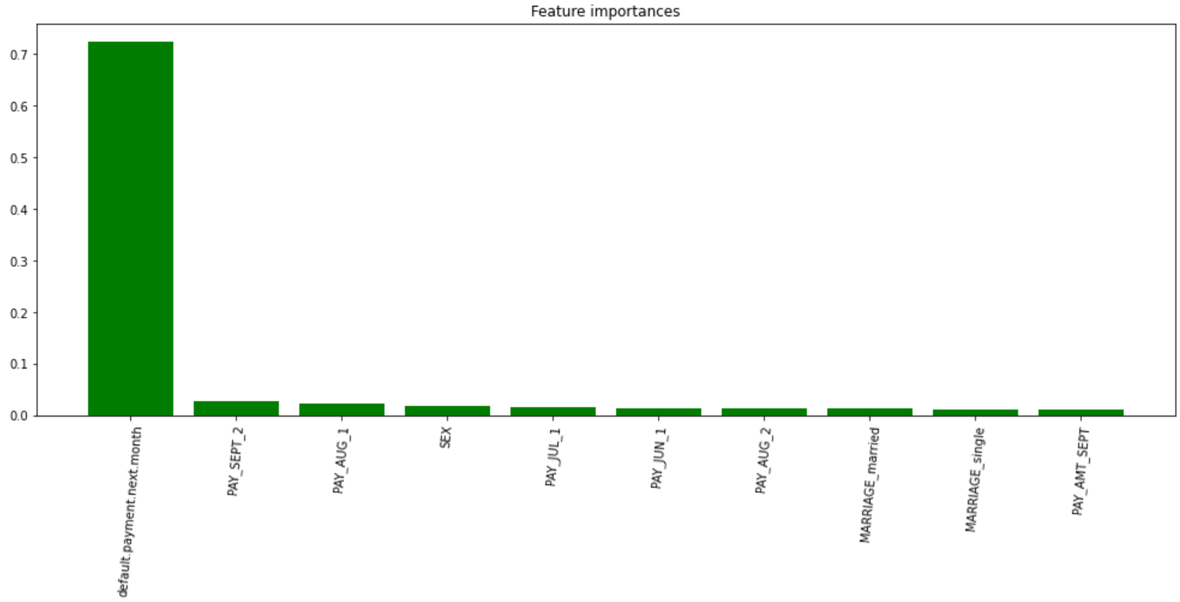
```
The accuracy on test data is  1.0
The precision on test data is  1.0
The recall on test data is  1.0
The f1 on test data is  1.0
The roc_score on test data is  1.0
```

```
len(optimal_rf_clf.feature_importances_)
```

```
81
```

```
# Feature Importance
feature_importances_rf = pd.DataFrame(optimal_rf_clf.feature_importances_,
                                   index = columns,
                                    columns=['importance_rf']).sort_values('importance_rf',
                                                                    ascending=False)[:10]
```

```
plt.subplots(figsize=(17,6))
plt.title("Feature importances")
plt.bar(feature_importances_rf.index, feature_importances_rf['importance_rf'],
        color="g",  align="center")
plt.xticks(feature_importances_rf.index, rotation = 85)
#plt.xlim([-1, X.shape[1]])
plt.show()
```




```
model_save_name = 'rf_optimized_classifier.pt'
path = F"./{model_save_name}"
torch.save(optimal_rf_clf, path)
```

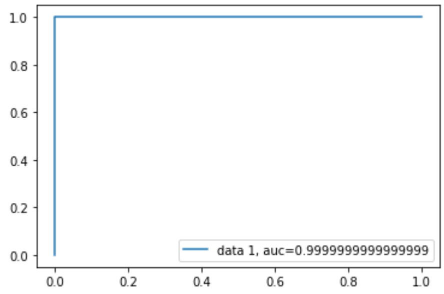
```
model_save_name = 'rf_optimized_classifier.pt'
path = F"./{model_save_name}"
optimal_rf_clf = torch.load(path)
```

```
# Get the predicted classes
train_class_preds = optimal_rf_clf.predict(X_train)
test_class_preds = optimal_rf_clf.predict(X_test)
```

```
y_preds_proba_rf = optimal_rf_clf.predict_proba(X_test)[::,1]
```

```
import sklearn.metrics as metrics
```

```
y_pred_proba = y_preds_proba_rf
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```



Implementing XGBoost

```
#import lightgbm and xgboost
import lightgbm as lgb
import xgboost as xgb
```

Applying XGBoost

```
#The data is stored in a DMatrix object
#label is used to define our outcome variable
dtrain=xgb.DMatrix(X_train,label=y_train)
dtest=xgb.DMatrix(X_test)
```

```
#setting parameters for xgboost
parameters={'max_depth':7, 'eta':1, 'silent':1,'objective':'binary:logistic','eval_metric':'auc','learning_rate':.05}
```

```
#training our model
num_round=50
from datetime import datetime
start = datetime.now()
xg=xgb.train(parameters,dtrain,num_round)
stop = datetime.now()
```

```
[16:37:36] WARNING: ../src/learner.cc:627:
Parameters: { "silent" } might not be used.

  This could be a false alarm, with some parameters getting used by language bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actually being used
  but getting flagged wrongly here. Please open an issue if you find any such cases.
```

```
#Execution time of the model
execution_time_xgb = stop-start
execution_time_xgb
```

```
datetime.timedelta(seconds=2, microseconds=326039)
```

```
#now predicting our model on train set
train_class_preds_probs=xg.predict(dtrain)
#now predicting our model on test set
test_class_preds_probs =xg.predict(dtest)
```

```
len(train_class_preds_probs)
```

```
31307
```

```
train_class_preds = []
test_class_preds = []
for i in range(0,len(train_class_preds_probs)):
  if train_class_preds_probs[i] >= 0.5:
    train_class_preds.append(1)
  else:
    train_class_preds.append(0)

for i in range(0,len(test_class_preds_probs)):
  if test_class_preds_probs[i] >= 0.5:
    test_class_preds.append(1)
  else:
    test_class_preds.append(0)
```

```
test_class_preds_probs[:20]
```

```
array([0.04043363, 0.04043363, 0.04043363, 0.04043363, 0.04043363,
       0.04043363, 0.04043363, 0.04043363, 0.95956635, 0.04043363,
       0.95956635, 0.04043363, 0.95956635, 0.04043363, 0.95956635,
       0.04043363, 0.95956635, 0.04043363, 0.95956635, 0.04043363],
      dtype=float32)
```

```
test_class_preds[:20]
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]
```

```
len(y_train)
```

```
31307
```

```
len(train_class_preds)
```

```
31307
```

```
# Get the accuracy scores
train_accuracy_xgb = accuracy_score(train_class_preds,y_train)
test_accuracy_xgb = accuracy_score(test_class_preds,y_test)

print("The accuracy on train data is ", train_accuracy_xgb)
print("The accuracy on test data is ", test_accuracy_xgb)
```

```
The accuracy on train data is  1.0
The accuracy on test data is  1.0
```

```
test_accuracy_xgb = accuracy_score(test_class_preds,y_test)
test_precision_xgb = precision_score(test_class_preds,y_test)
test_recall_score_xgb = recall_score(test_class_preds,y_test)
test_f1_score_xgb = f1_score(test_class_preds,y_test)
test_roc_score_xgb = roc_auc_score(test_class_preds,y_test)

print("The accuracy on test data is ", test_accuracy_xgb)
print("The precision on test data is ", test_precision_xgb)
print("The recall on test data is ", test_recall_score_xgb)
print("The f1 on test data is ", test_f1_score_xgb)
print("The roc_score on train data is ", test_roc_score_xgb)
```

```
The accuracy on test data is  1.0
The precision on test data is  1.0
The recall on test data is  1.0
The f1 on test data is  1.0
The roc_score on train data is  1.0
```

Hyperparameter Tuning

```
from xgboost import  XGBClassifier
```

```
X = df_fr.drop(['IsDefaulter','Payement_Value','Dues'],axis=1)
y = df_fr['IsDefaulter']
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42, stratify = y)
```

```
param_test1 = {
 'max_depth':range(3,10,2),
 'min_child_weight':range(1,6,2)
}
gsearch1 = GridSearchCV(estimator = XGBClassifier( learning_rate =0.1, n_estimators=140, max_depth=5,
 min_child_weight=1, gamma=0, subsample=0.8, colsample_bytree=0.8,
 objective= 'binary:logistic', nthread=4, scale_pos_weight=1, seed=27),
 param_grid = param_test1, scoring='accuracy',n_jobs=-1, cv=3, verbose = 2)
gsearch1.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
GridSearchCV(cv=3,
             estimator=XGBClassifier(base_score=None, booster=None,
```

```
                                       callbacks=None, colsample_bylevel=None,
                                       colsample_bynode=None,
                                       colsample_bytree=0.8,
                                       early_stopping_rounds=None,
                                       enable_categorical=False, eval_metric=None,
                                       gamma=0, gpu_id=None, grow_policy=None,
                                       importance_type=None,
                                       interaction_constraints=None,
                                       learning_rate=0.1, max_bin=None,
                                       max_cat_to_onehot=None,
                                       max_delta_step=None, max_depth=5,
                                       max_leaves=None, min_child_weight=1,
                                       missing=nan, monotone_constraints=None,
                                       n_estimators=140, n_jobs=None, nthread=4,
                                       num_parallel_tree=None, predictor=None,
                                       random_state=None, reg_alpha=None, ...),
                 n_jobs=-1,
                 param_grid={'max_depth': range(3, 10, 2),
                             'min_child_weight': range(1, 6, 2)},
                 scoring='accuracy', verbose=2)
```

```
gsearch1.best_score_
```

```
1.0
```

```
optimal_xgb = gsearch1.best_estimator_
```

```
# Get the predicted classes
train_class_preds = optimal_xgb.predict(X_train)
test_class_preds = optimal_xgb.predict(X_test)
```

```
# Get the accuracy scores
train_accuracy_xgb_tuned = accuracy_score(train_class_preds,y_train)
test_accuracy_xgb_tuned = accuracy_score(test_class_preds,y_test)

print("The accuracy on train data is ", train_accuracy_xgb_tuned)
print("The accuracy on test data is ", test_accuracy_xgb_tuned)
```

```
The accuracy on train data is  1.0
The accuracy on test data is  1.0
```

```
test_accuracy_xgb_tuned = accuracy_score(test_class_preds,y_test)
test_precision_xgb_tuned = precision_score(test_class_preds,y_test)
test_recall_score_xgb_tuned = recall_score(test_class_preds,y_test)
test_f1_score_xgb_tuned = f1_score(test_class_preds,y_test)
test_roc_score_xgb_tuned = roc_auc_score(test_class_preds,y_test)

print("The accuracy on test data is ", test_accuracy_xgb_tuned)
print("The precision on test data is ", test_precision_xgb_tuned)
print("The recall on test data is ", test_recall_score_xgb_tuned)
print("The f1 on test data is ", test_f1_score_xgb_tuned)
print("The roc_score on train data is ", test_roc_score_xgb_tuned)
```

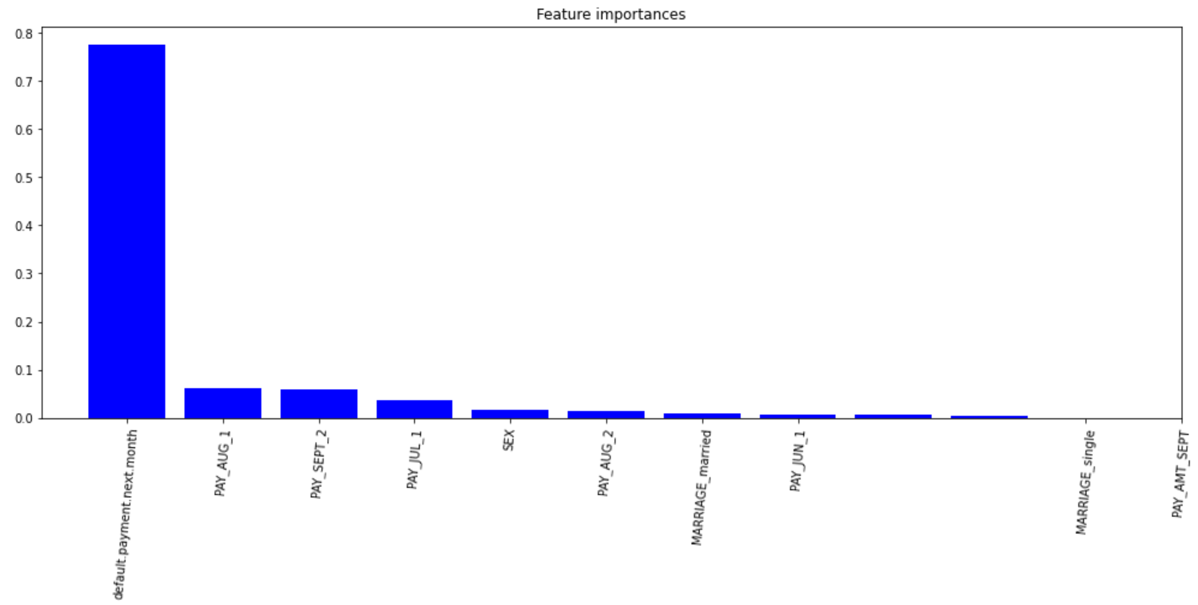
```
The accuracy on test data is  1.0
The precision on test data is  1.0
The recall on test data is  1.0
The f1 on test data is  1.0
The roc_score on train data is  1.0
```

```
pd.DataFrame(optimal_xgb.feature_importances_,
                                   index = columns,
                                    columns=['importance_xgb']).sort_values('importance_xgb',
                                                                    ascending=False)[:10]
```

	importance_xgb
default.payment.next.month	0.774541
PAY_AUG_1	0.061343
PAY_SEPT_2	0.058241
PAY_JUL_1	0.036967
SEX	0.016271
PAY_AUG_2	0.014785
MARRIAGE_married	0.008888
PAY_JUN_1	0.006657
PAY_JUL_-1	0.005823
PAY_SEPT_1	0.005516

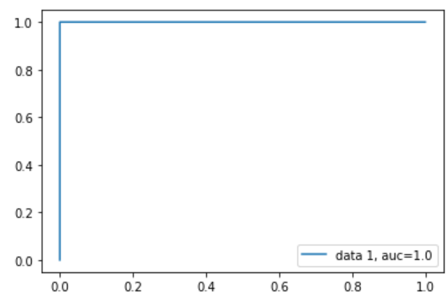
```
# Feature Importance
feature_importances_xgb = pd.DataFrame(optimal_xgb.feature_importances_,
                                   index = columns,
                                    columns=['importance_xgb']).sort_values('importance_xgb',
                                                                    ascending=False)[:10]
```

```
plt.subplots(figsize=(17,6))
plt.title("Feature importances")
plt.bar(feature_importances_xgb.index, feature_importances_xgb['importance_xgb'],
        color="b",  align="center")
plt.xticks(feature_importances_rf.index, rotation = 85)
#plt.xlim([-1, X.shape[1]])
plt.show()
```




```
y_preds_proba_xgb = optimal_xgb.predict_proba(X_test)[::,1]
```

```
y_pred_proba = y_preds_proba_xgb
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```




```
model_save_name = 'xgb_optimized_classifier.pt'
path = F"./{model_save_name}"
torch.save(optimal_xgb, path)
```

```
model_save_name = 'xgb_optimized_classifier.pt'
path = F"./{model_save_name}"
optimal_xgb = torch.load(path)
```

Evaluating the models

```
recall_score
```

```
<function sklearn.metrics._classification.recall_score(y_true, y_pred, *, labels=None, pos_label=1, average='binary', sample_weight=None,
zero_division='warn')>
```

```
classifiers = ['Logistic Regression', 'SVC', 'Random Forest CLf', 'Xgboost Clf']
train_accuracy = [train_accuracy_lr, train_accuracy_SVC, train_accuracy_rf, train_accuracy_xgb_tuned]
test_accuracy = [test_accuracy_lr, test_accuracy_SVC, test_accuracy_rf, test_accuracy_xgb_tuned]
precision_score = [test_precision_score_lr, test_precision_score_SVC, test_precision_score_rf, test_precision_xgb_tuned]
recall_score = [test_recall_score_lr, test_recall_score_SVC, test_recall_score_rf, test_recall_score_xgb_tuned]
f1_score = [test_f1_score_lr, test_f1_score_SVC, test_f1_score_rf, test_f1_score_xgb_tuned]
```

```
pd.DataFrame({'Classifier':classifiers, 'Train Accuracy': train_accuracy, 'Test Accuracy': test_accur                                    s
```