Importing libraries

In [ ]:
```
# 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,

from sklearn.svm import SVC
```

Loading the Dataset

In [ ]:
```
path = 'Credit_Card.csv'
df = pd.read_csv(path)
```

Viewing the Dataset

In [ ]:
```
df
```

Out[ ]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4
0	1	20000.0	2	2	1	24	2	2	-1	-1
1	2	120000.0	2	2	2	26	-1	2	0	0
2	3	90000.0	2	2	2	34	0	0	0	0
3	4	50000.0	2	2	1	37	0	0	0	0
4	5	50000.0	1	2	1	57	-1	0	-1	0
...	...	...	...	...	...	...	...	...	...	...
29995	29996	220000.0	1	3	1	39	0	0	0	0
29996	29997	150000.0	1	3	2	43	-1	-1	-1	-1
29997	29998	30000.0	1	2	2	37	4	3	2	-1
29998	29999	80000.0	1	3	1	41	1	-1	0	0
29999	30000	50000.0	1	2	1	46	0	0	0	0

30000 rows × 25 columns

Printing the Dataset Information

In [ ]:
```
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
```

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.

Checking for any null values in the dataset

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
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
```

Data Description

Here we check on data description to the descriptive features like mean,max,std and others from our dataset

In [ ]:
```
df.describe()
```

Out[ ]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE
count	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
std	8660.398374	129747.661567	0.489129	0.790349	0.521970	9.217904
min	1.000000	10000.000000	1.000000	0.000000	0.000000	21.000000
25%	7500.750000	50000.000000	1.000000	1.000000	1.000000	28.000000
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.000000
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.000000
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.000000

8 rows × 25 columns

Exploratory Data Analysis

Dependent Variable:

In [ ]:
```
#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)
```

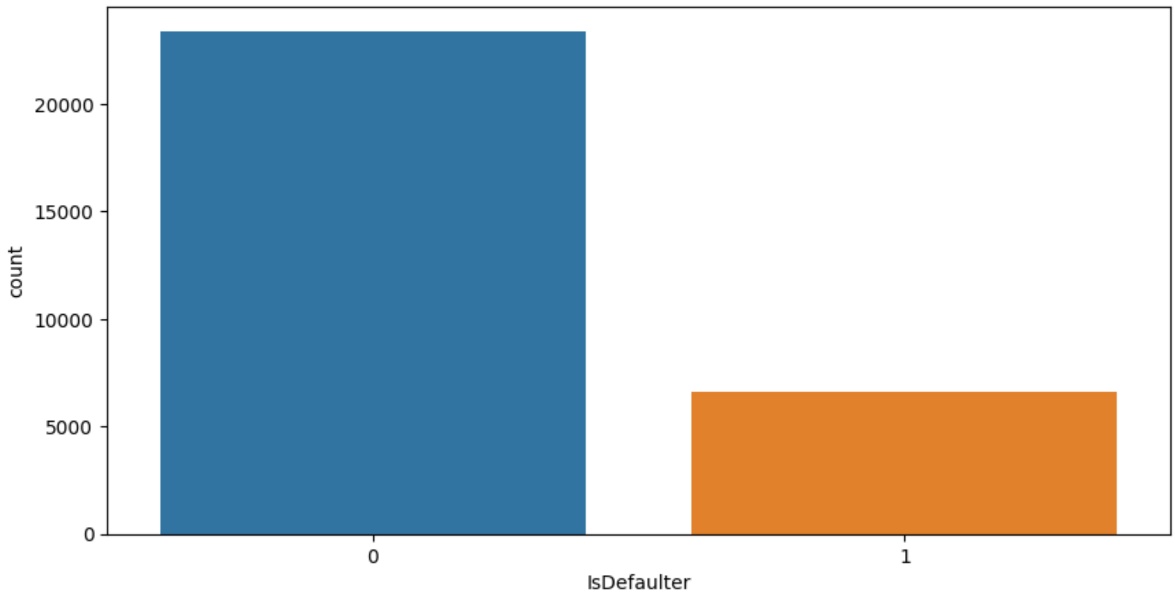
Out[ ]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4
0	1	20000.0	2	2	1	24	2	2	-1	-1
1	2	120000.0	2	2	2	26	-1	2	0	0
2	3	90000.0	2	2	2	34	0	0	0	0
3	4	50000.0	2	2	1	37	0	0	0	0
4	5	50000.0	1	2	1	57	-1	0	-1	0
...	...	...	...	...	...	...	...	...	...	...
29995	29996	220000.0	1	3	1	39	0	0	0	0
29996	29997	150000.0	1	3	2	43	-1	-1	-1	-1
29997	29998	30000.0	1	2	2	37	4	3	2	-1
29998	29999	80000.0	1	3	1	41	1	-1	0	0
29999	30000	50000.0	1	2	1	46	0	0	0	0

30000 rows × 25 columns

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

Out[ ]:
```
<Axes: xlabel='IsDefaulter', ylabel='count'>
```



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

Out[ ]:
```
IsDefaulter
0    23364
1     6636
Name: count, 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.They are demonstrated as below.

SEX

- 1 - Male
- 2 - Female

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

Out[ ]:
```
SEX
2    18112
1    11888
Name: count, dtype: int64
```

Education

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

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

Out[ ]:
```
EDUCATION
2    14030
1    10585
3     4917
4      468
Name: count, 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.

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

Out[ ]:
```
EDUCATION
2    14030
1    10585
3     4917
4      468
Name: count, dtype: int64
```

Marriage

1 = married; 2 = single; 3 = others

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

Out[ ]:
```
MARRIAGE
2    15964
1    13659
3      377
Name: count, dtype: int64
```

We have few values for 0, which are not determined therefore the are added in Others category.

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

Out[ ]:
```
MARRIAGE
2    15964
1    13659
3      377
Name: count, dtype: int64
```

Plotting our categorical features

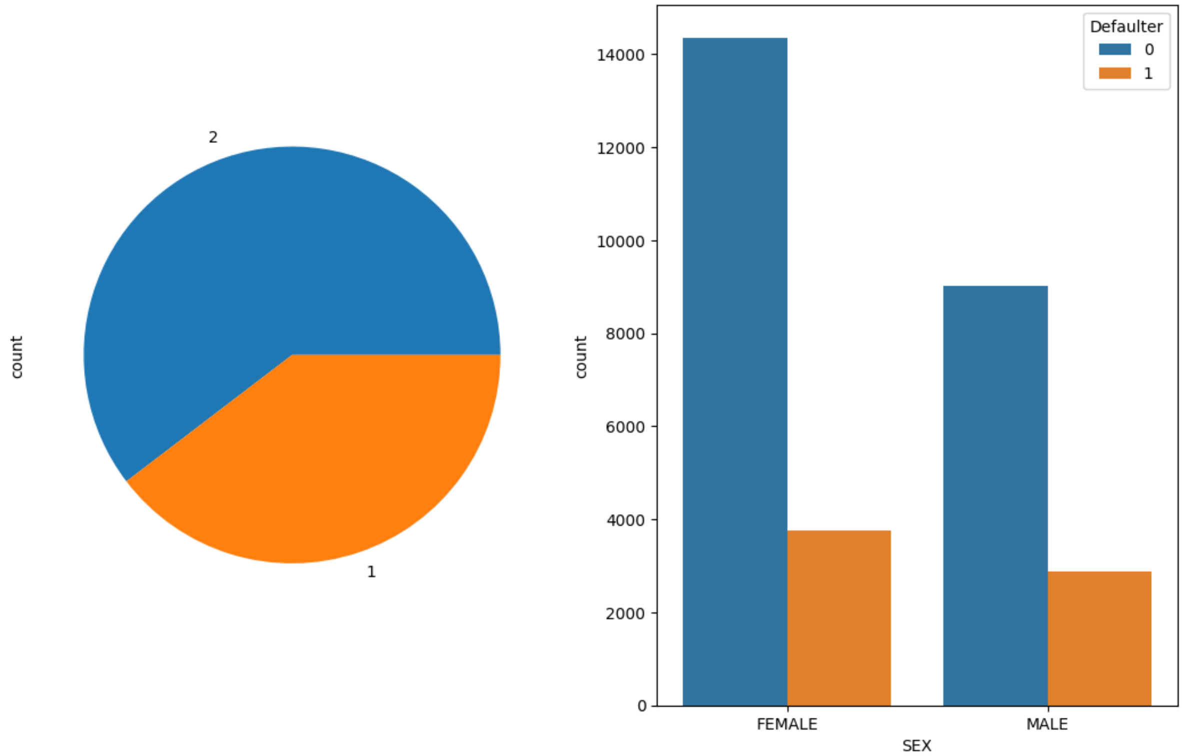
In [ ]:
```
categorical_features = ['SEX', 'EDUCATION', 'MARRIAGE']
```

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

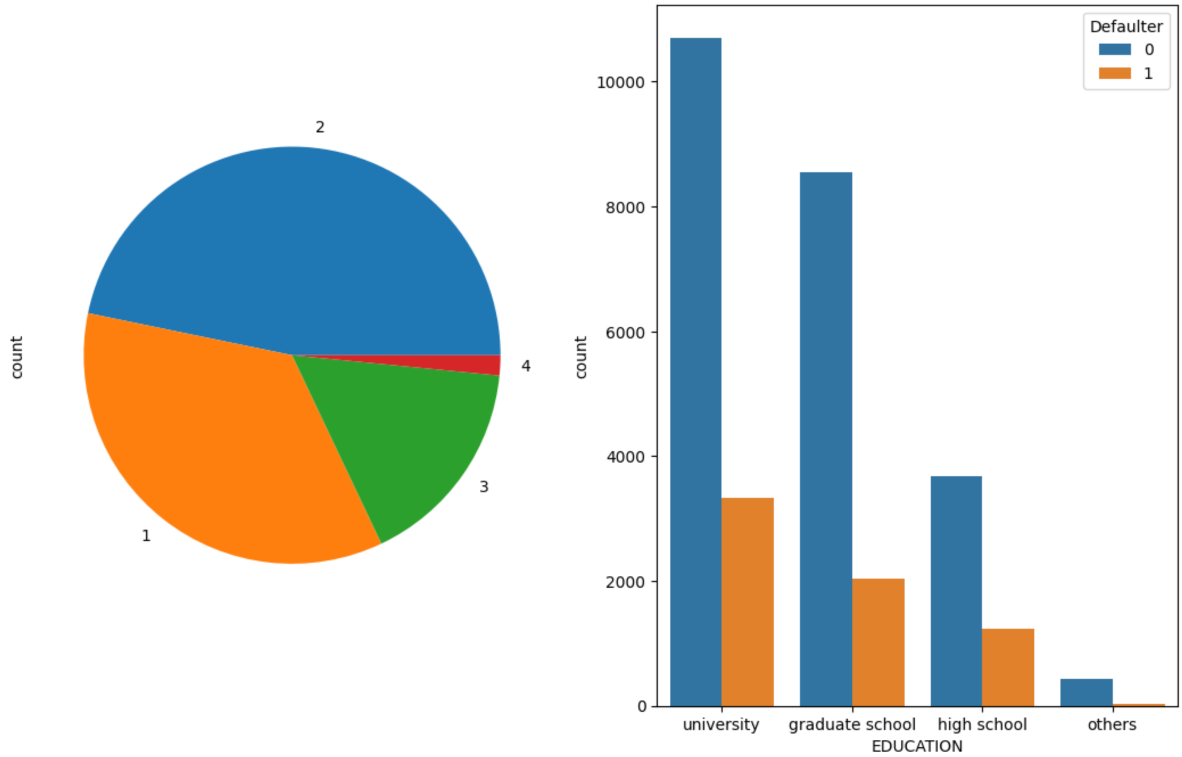
In [ ]:
```
df_cat.replace({'SEX': {1 : 'MALE', 2 : 'FEMALE'}, 'EDUCATION' : {1 : 'grad
```

In [ ]:
```
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)
```

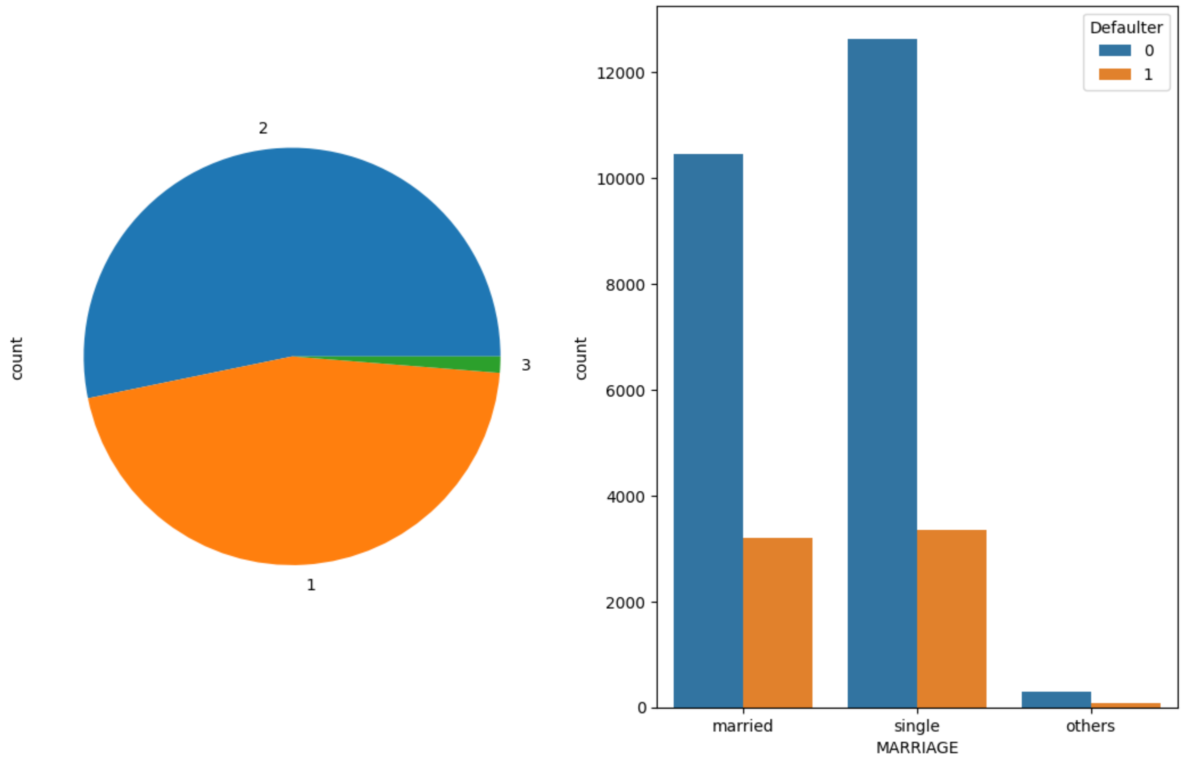
```
<Figure size 1000x500 with 0 Axes>
```




```
<Figure size 1000x500 with 0 Axes>
```




```
<Figure size 1000x500 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

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

Out[ ]:
```
1000000.0
```

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

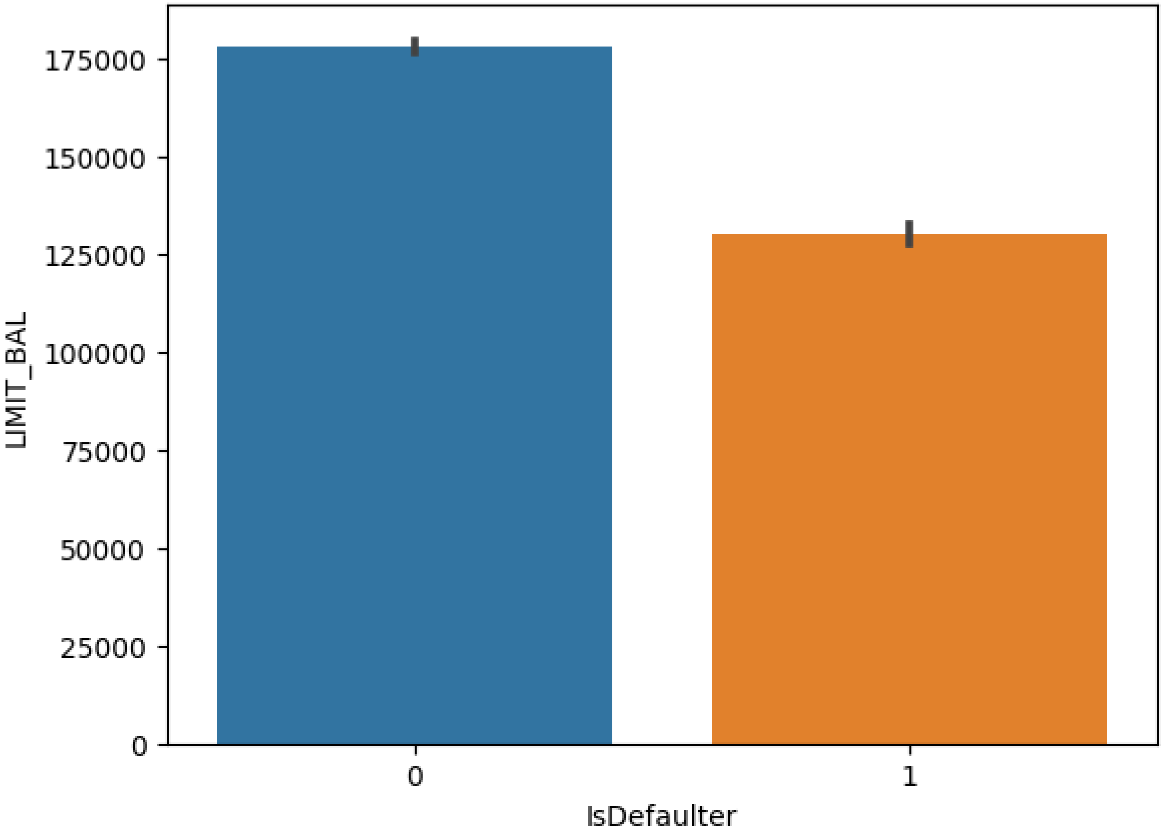
Out[ ]:
```
10000.0
```

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

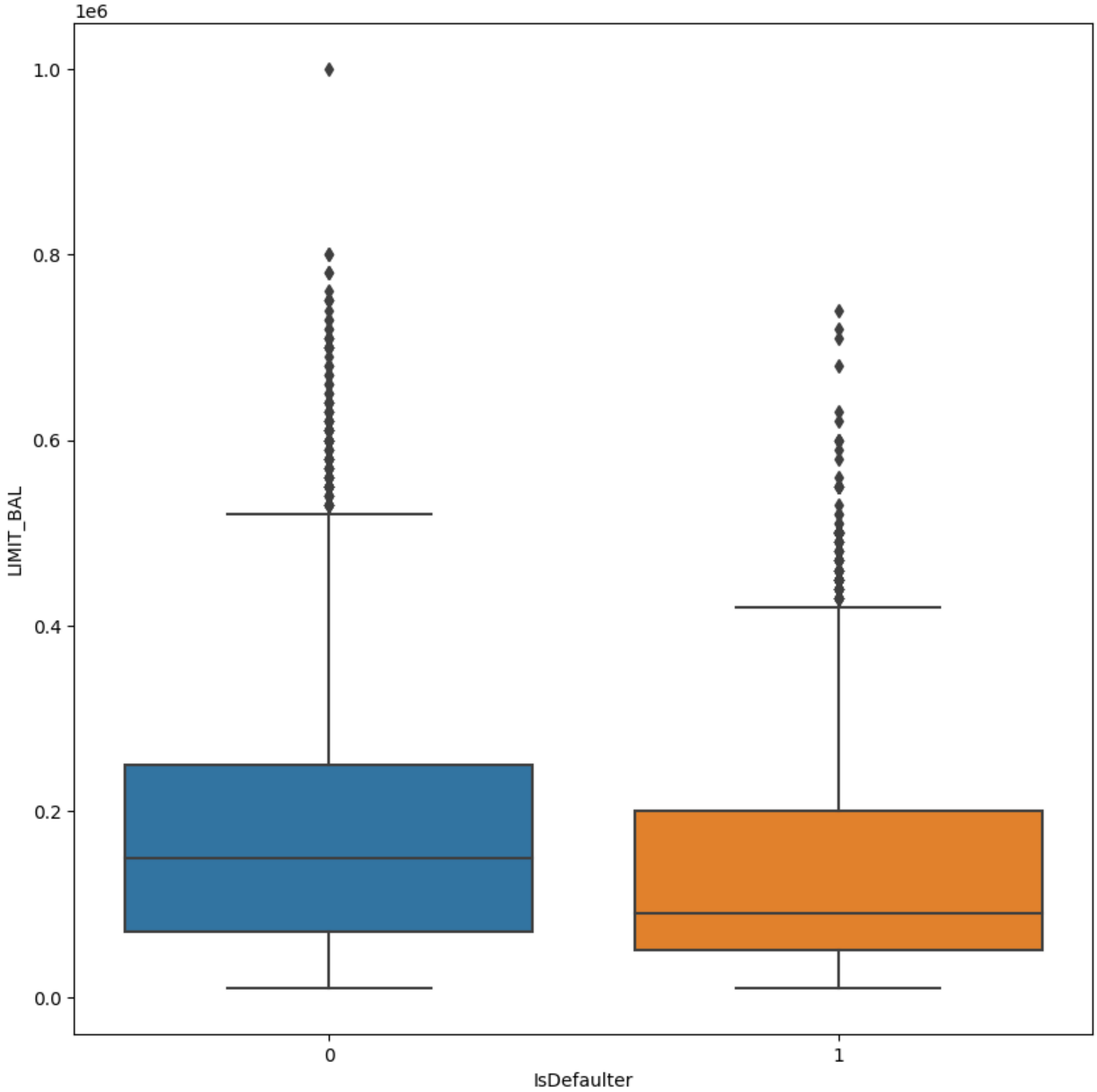
Out[ ]:
```
count      30000.000000
mean      167484.322667
std       129747.661567
min        10000.000000
25%        50000.000000
50%       140000.000000
75%       240000.000000
max      1000000.000000
Name: LIMIT_BAL, dtype: float64
```

In [ ]:
```
sns.barplot(x='IsDefaulter', y='LIMIT_BAL', data=df)
```

Out[ ]:
```
<Axes: xlabel='IsDefaulter', ylabel='LIMIT_BAL'>
```



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



In [ ]:
```
#renaming columns

df.rename(columns={'PAY_0':'PAY_SEPT','PAY_2':'PAY_AUG','PAY_3':'PAY_JUL','
df.rename(columns={'BILL_AMT1':'BILL_AMT_SEPT','BILL_AMT2':'BILL_AMT_AUG','
df.rename(columns={'PAY_AMT1':'PAY_AMT_SEPT','PAY_AMT2':'PAY_AMT_AUG','PAY_
```

In [ ]:
```
df.head()
```

Out[ ]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_SEPT	PAY_AUG	PAY_JUL	PAY_
0	1	20000.0	2	2	1	24	2	2	-1	
1	2	120000.0	2	2	2	26	-1	2	0	
2	3	90000.0	2	2	2	34	0	0	0	
3	4	50000.0	2	2	1	37	0	0	0	
4	5	50000.0	1	2	1	57	-1	0	-1	

5 rows × 26 columns

AGE

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

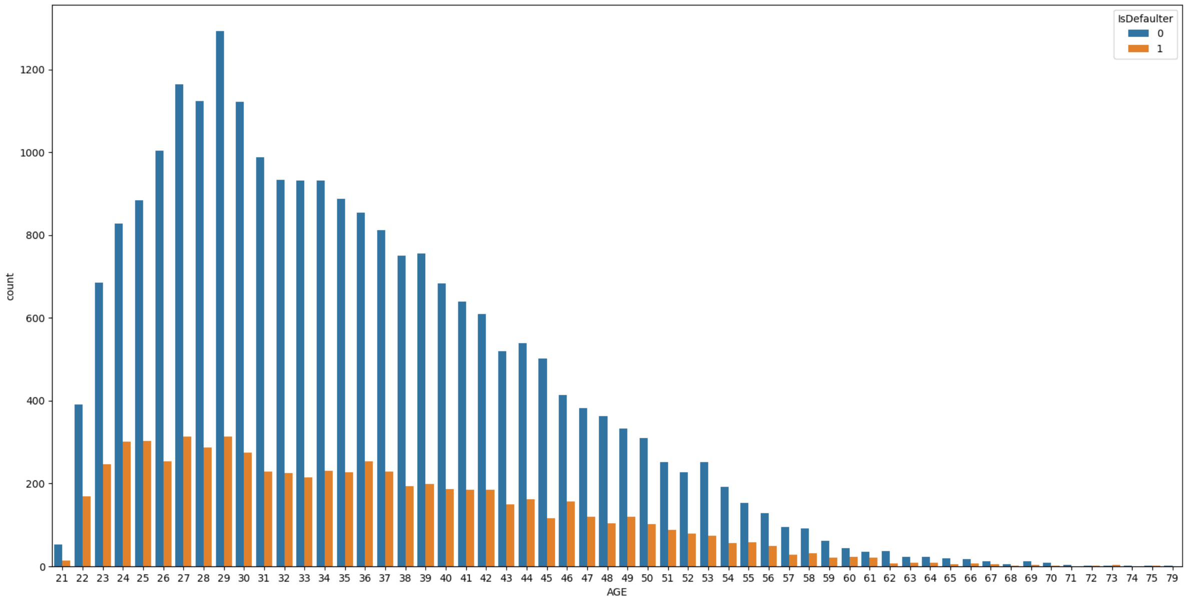
In [ ]:
```
df['AGE'].value_counts()
```

Out[ ]:
```
AGE
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: count, dtype: int64
```

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

In [ ]:
```
# subplots of age
plt.figure(figsize=(20,10))
sns.countplot(x='AGE',hue='IsDefaulter',data=df)
```

Out[ ]:
```
<Axes: xlabel='AGE', ylabel='count'>
```

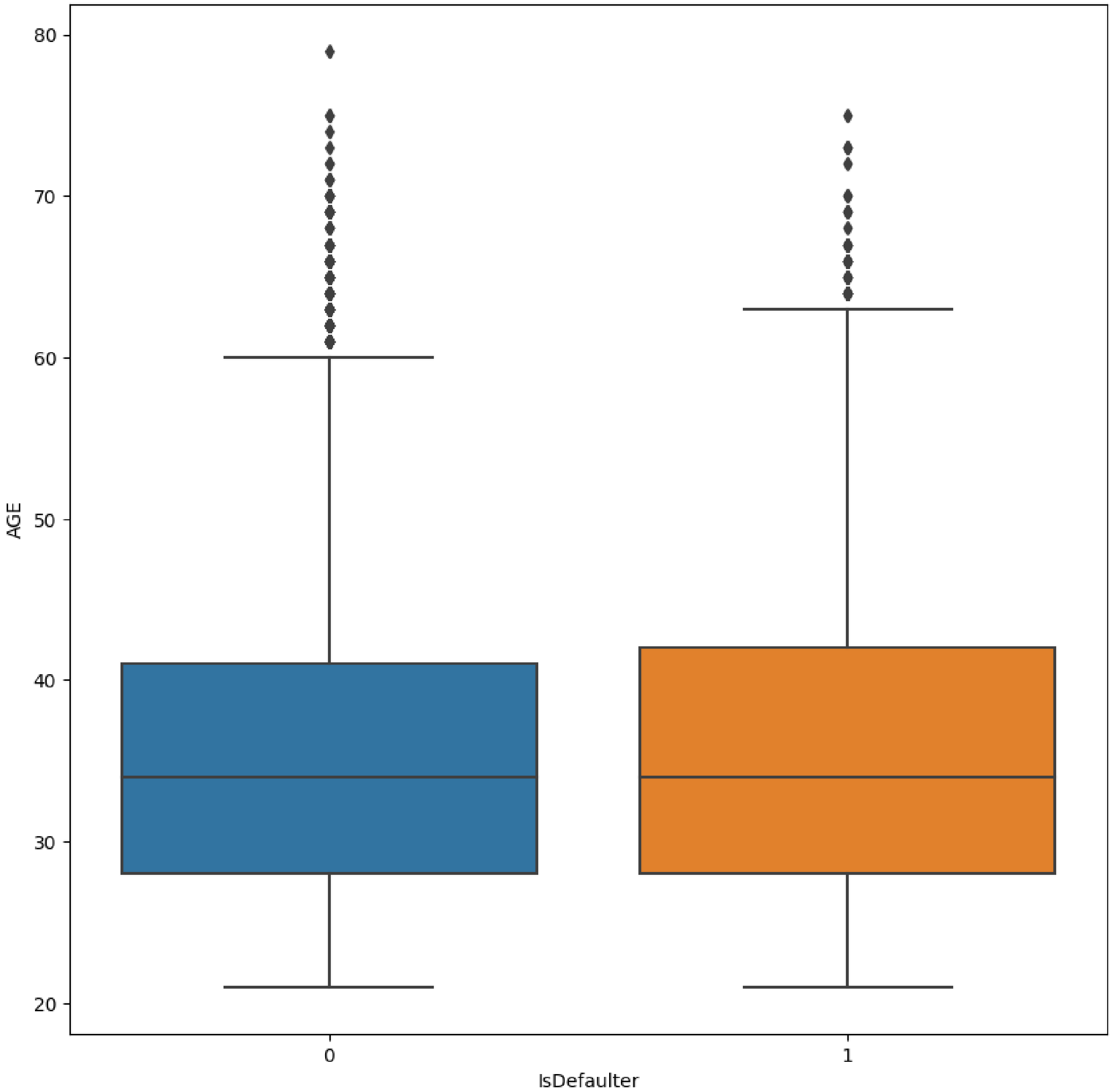


In [ ]:
```
df.groupby('IsDefaulter')['AGE'].mean()
```

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

In [ ]:
```
df = df.astype('int')
```

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

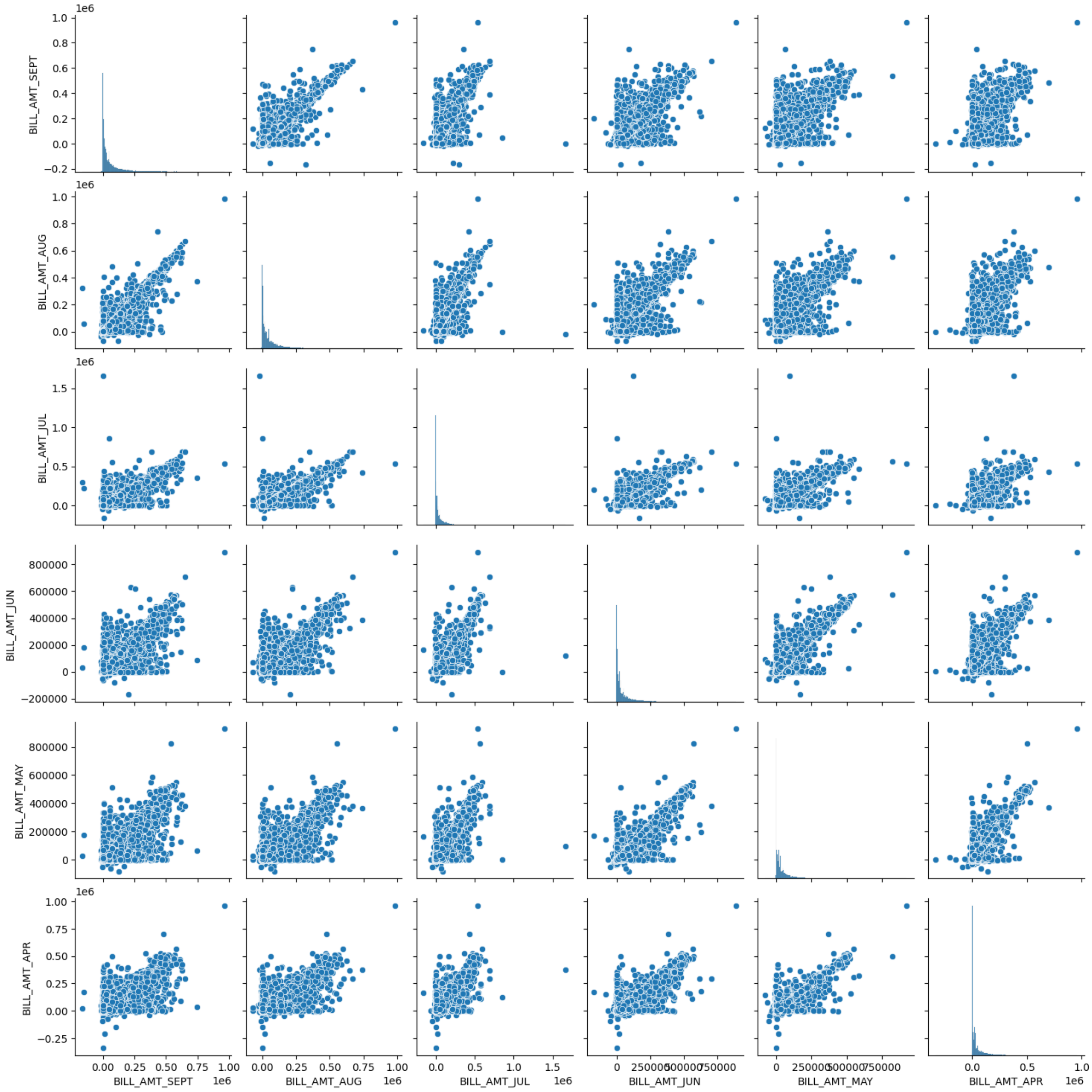


Bill Amount

In [ ]:
```
bill_amnt_df = df[['BILL_AMT_SEPT',     'BILL_AMT_AUG', 'BILL_AMT_JUL', 'BI
```

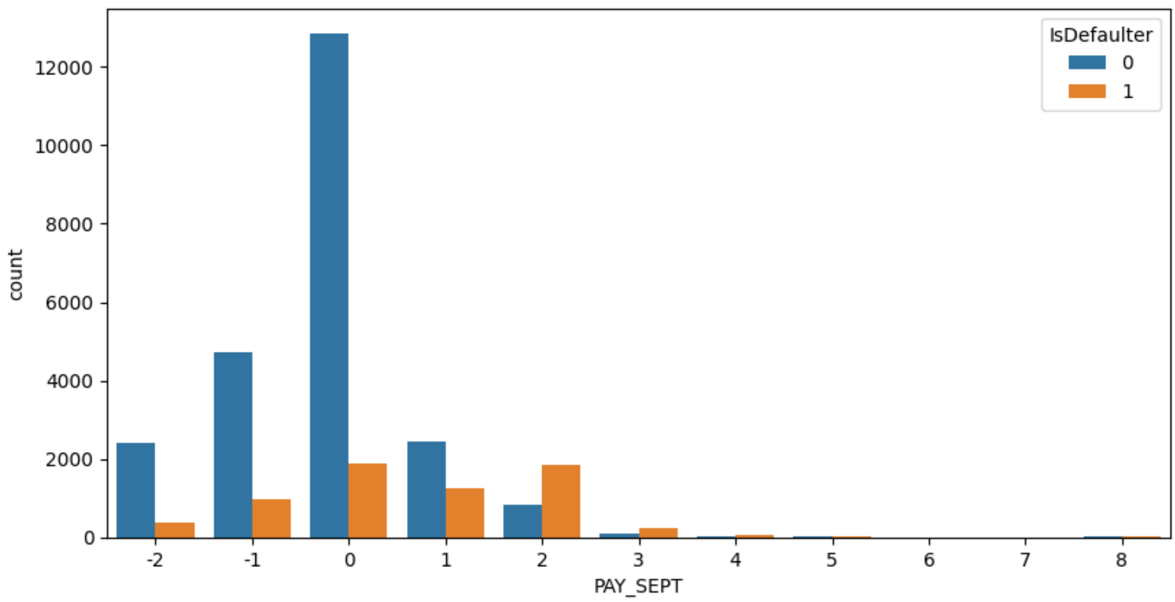
In [ ]:
```
sns.pairplot(data = bill_amnt_df)
```

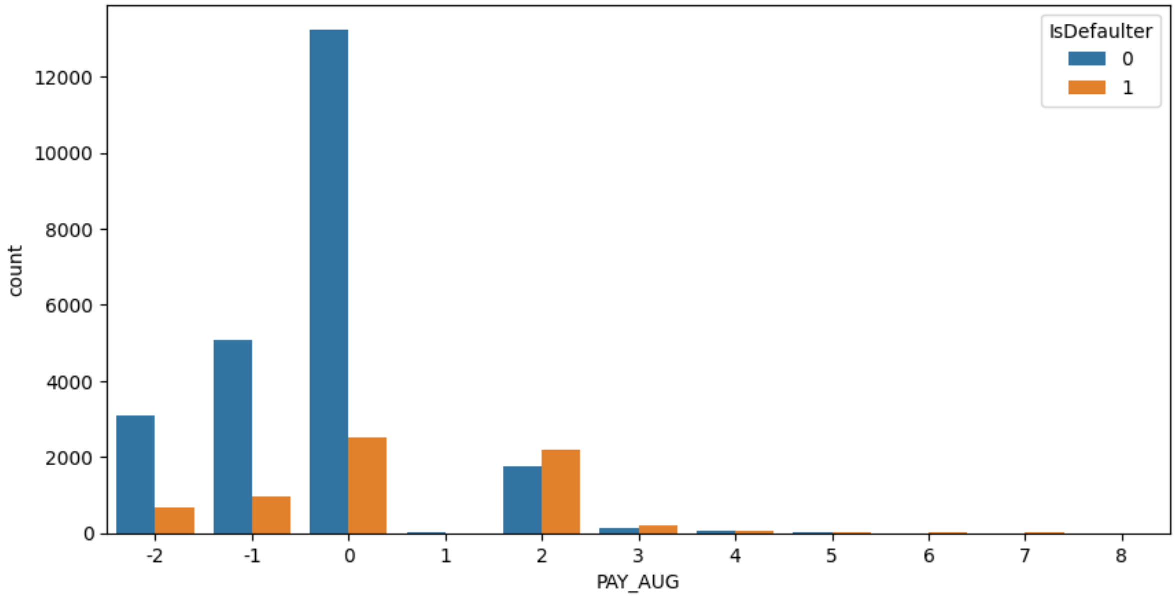
Out[ ]: <seaborn.axisgrid.PairGrid at 0x7f1bbb6a34f0>

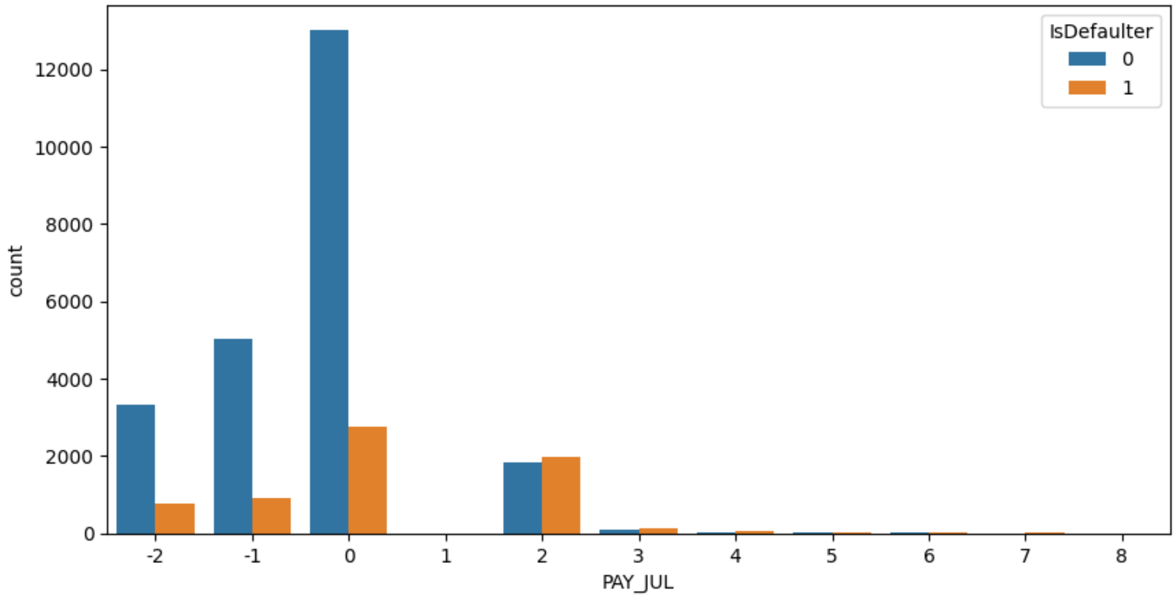


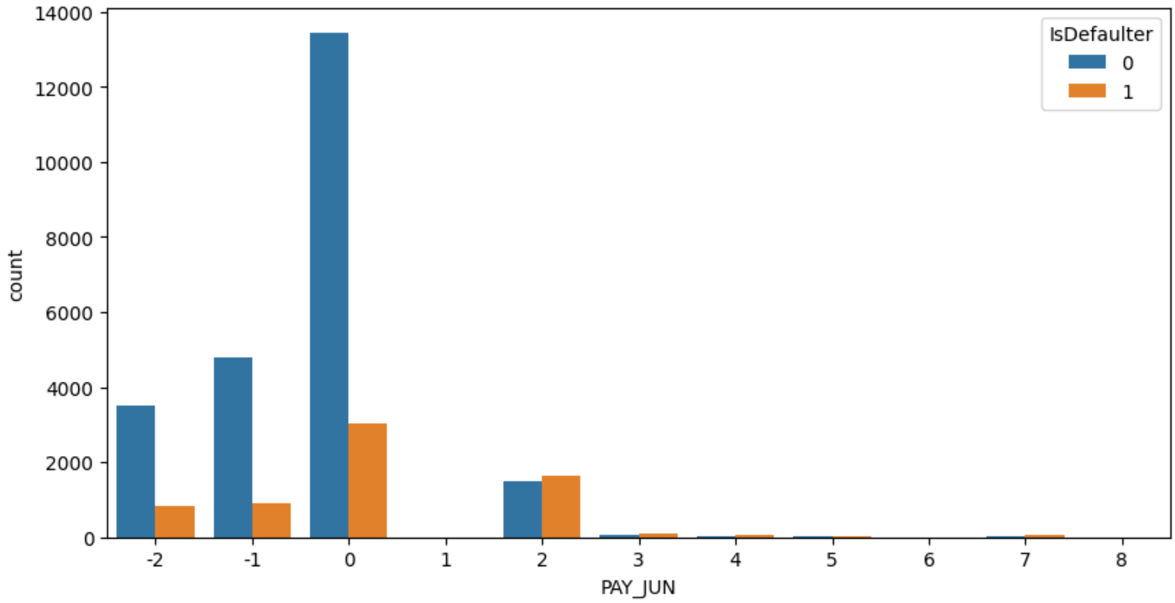
History payment status

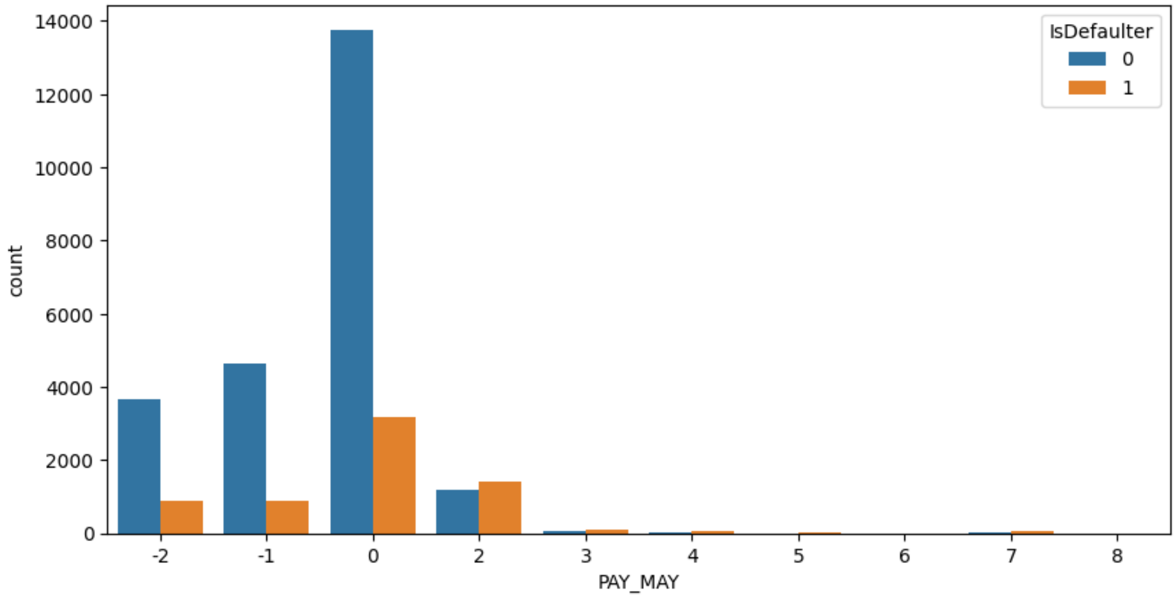
```
In [ ]: pay_col = ['PAY_SEPT',  'PAY_AUG',     'PAY_JUL',     'PAY_JUN',     'PA
        for col in pay_col:
          plt.figure(figsize=(10,5))
          sns.countplot(x = col, hue = 'IsDefaulter', data = df)
```

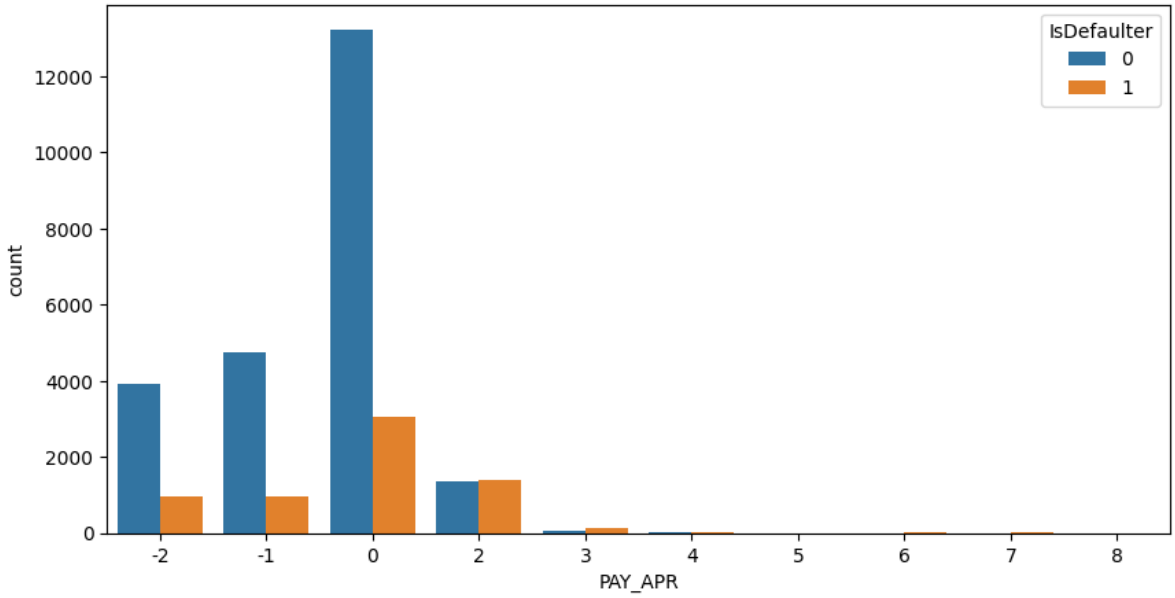



IsDefaulter
0
1
count
12000
10000
8000
6000
4000
2000
0
-2
-1
0
1
2
3
4
5
6
7
8
PAY_AUG


IsDefaulter
0
1
count
12000
10000
8000
6000
4000
2000
0
-2
-1
0
1
2
3
4
5
6
7
8
PAY_JUL


IsDefaulter
0
1
count
14000
12000
10000
8000
6000
4000
2000
0
-2
-1
0
1
2
3
4
5
6
7
8
PAY_JUN



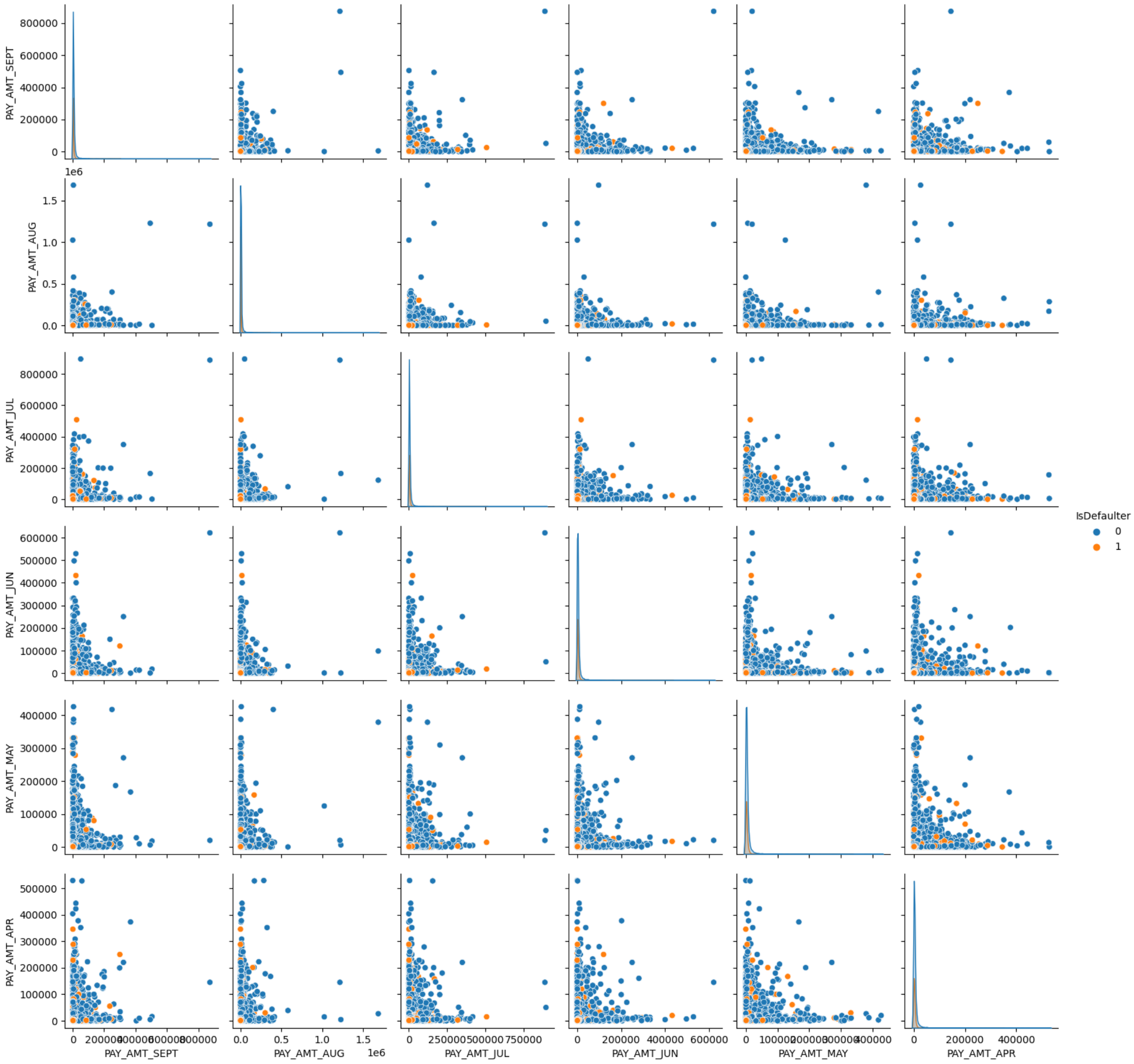


Paid Amount

In [ ]:
```
pay_amnt_df = df[['PAY_AMT_SEPT',        'PAY_AMT_AUG',  'PAY_AMT_JUL',  'PA
```

In [ ]:
```
sns.pairplot(data = pay_amnt_df, hue='IsDefaulter')
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7f2cd5589d20>
```



In [ ]:
```python
# shape of the data
df.shape
```

Out[ ]:
```
(30000, 26)
```

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

In [ ]:
```python
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
```

In [ ]:
```python
x_smote
```

Out[ ]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_SEPT	PAY_AUG	PAY_JUL
0	1	20000	2	2	1	24	2	2	-1
1	2	120000	2	2	2	26	-1	2	0
2	3	90000	2	2	2	34	0	0	0
3	4	50000	2	2	1	37	0	0	0
4	5	50000	1	2	1	57	-1	0	-1
...	...	...	...	...	...	...	...	...	...
46723	10827	80000	2	2	1	29	0	0	-1
46724	3883	20000	1	1	1	31	2	2	2
46725	6998	130000	1	1	2	34	0	0	0
46726	25713	30000	2	2	1	37	2	2	2
46727	16177	383043	2	1	1	36	-1	-1	-1

46728 rows × 25 columns

In [ ]:
```python
columns = list(df.columns)
```

In [ ]:
```python
columns.pop()
```

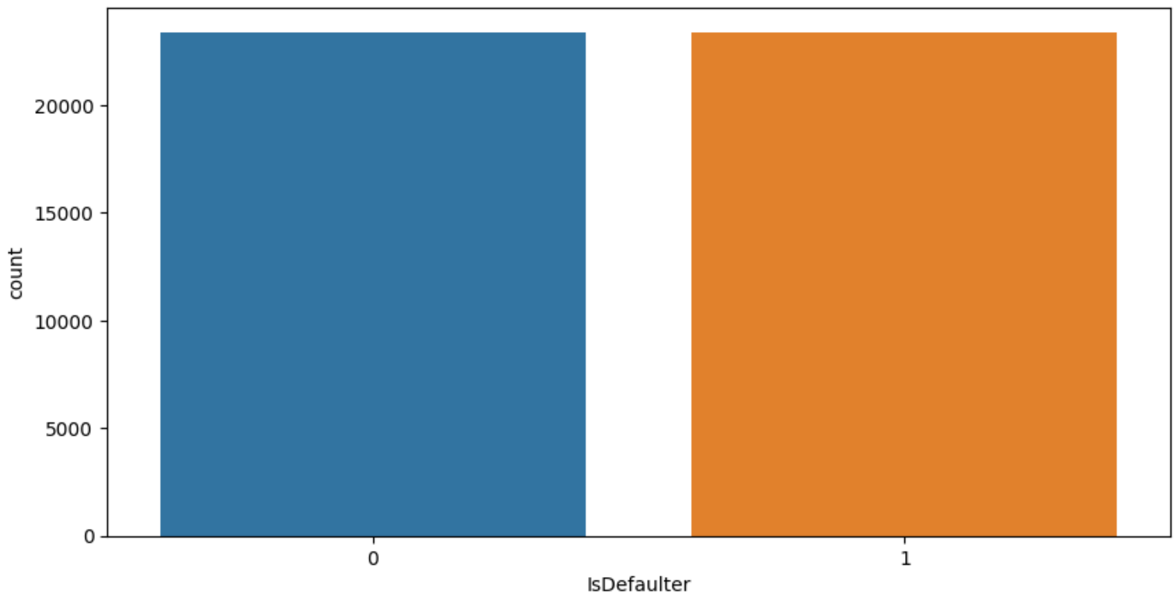
Out[ ]:
```
'IsDefaulter'
```

In [ ]:
```python
balance_df = pd.DataFrame(x_smote, columns=columns)
```

In [ ]:
```python
balance_df['IsDefaulter'] = y_smote
```

In [ ]:
```python
# plot the count after resample
plt.figure(figsize=(10,5))
sns.countplot(x = 'IsDefaulter', data = balance_df)
```

Out[ ]:
```
<Axes: xlabel='IsDefaulter', ylabel='count'>
```



In [ ]:
```python
balance_df[balance_df['IsDefaulter']==1]
```

Out[ ]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_SEPT	PAY_AUG	PAY_JUL
0	1	20000	2	2	1	24	2	2	-1
1	2	120000	2	2	2	26	-1	2	0
13	14	70000	1	2	2	30	1	2	2
16	17	20000	1	1	2	24	0	0	2
21	22	120000	2	2	1	39	-1	-1	-1
...	...	...	...	...	...	...	...	...	...
46723	19742	56575	2	2	2	26	1	0	0
46724	7160	252470	1	2	1	53	0	0	0
46725	23249	20000	2	2	2	44	0	0	0
46726	29479	10000	1	2	2	33	0	0	0
46727	26037	30000	1	2	1	37	1	1	1

23364 rows × 26 columns

Feature Engineering

In [ ]:
```
df_fr = balance_df.copy()
```

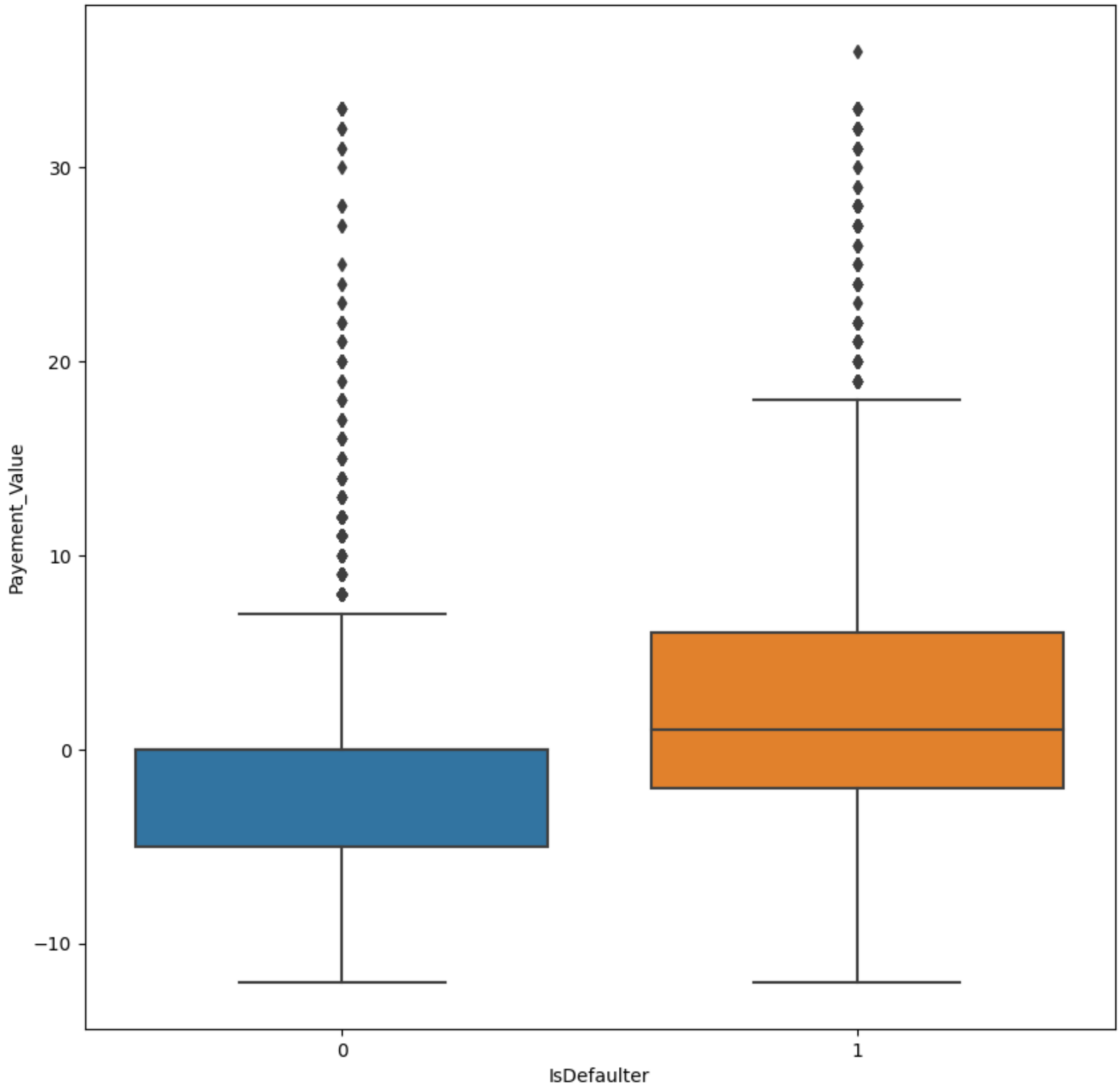
In [ ]:
```
df_fr['Payement_Value'] = df_fr['PAY_SEPT'] + df_fr['PAY_AUG'] + df_fr['PAY
```

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

Out[ ]:
```
IsDefaulter
0   -1.980140
1    1.704503
Name: Payement_Value, dtype: float64
```

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

Out[ ]:
```
<Axes: xlabel='IsDefaulter', ylabel='Payement_Value'>
```



In [ ]:
```
df_fr['Dues'] = (df_fr['BILL_AMT_APR']+df_fr['BILL_AMT_MAY']+df_fr['BILL_AM
```

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

Out[ ]:
```
IsDefaulter
0    187742.051532
1    195826.211822
Name: Dues, dtype: float64
```

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

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

In [ ]:
```

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'])
```

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

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

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

In [ ]:
```
df_fr.replace({'SEX': {1 : 'MALE', 2 : 'FEMALE'}, 'EDUCATION' : {1 : 'gradu
```

In [ ]:
```
df_fr.head()
```

Out[ ]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_SEPT	PAY_AUG	PAY_JUL	P/
0	1	20000	FEMALE	university	married	24	2	2	-1	
1	2	120000	FEMALE	university	single	26	-1	2	0	
2	3	90000	FEMALE	university	single	34	0	0	0	
3	4	50000	FEMALE	university	married	37	0	0	0	
4	5	50000	MALE	university	married	57	-1	0	-1	

5 rows × 28 columns

One Hot Encoding

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

In [ ]:
```
df_fr.head()
```

Out[ ]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_SEPT	PAY_AUG	PAY_JUL	P/
0	1	20000	FEMALE	university	married	24	2	2	-1	
1	2	120000	FEMALE	university	single	26	-1	2	0	
2	3	90000	FEMALE	university	single	34	0	0	0	
3	4	50000	FEMALE	university	married	37	0	0	0	
4	5	50000	MALE	university	married	57	-1	0	-1	

5 rows × 28 columns

In [ ]:
```
df_fr.drop(['EDUCATION_others','MARRIAGE_others'],axis = 1, inplace = True)
df_fr = pd.get_dummies(df_fr, columns = ['PAY_SEPT',    'PAY_AUG',     'PA
df_fr.head()
```

Out[ ]:

	ID	LIMIT_BAL	SEX	AGE	BILL_AMT_SEPT	BILL_AMT_AUG	BILL_AMT_JUL	BILL_AMT
0	1	20000	FEMALE	24	3913	3102	689	
1	2	120000	FEMALE	26	2682	1725	2682	
2	3	90000	FEMALE	34	29239	14027	13559	
3	4	50000	FEMALE	37	46990	48233	49291	
4	5	50000	MALE	57	8617	5670	35835	

5 rows × 85 columns

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

Out[ ]:

	ID	LIMIT_BAL	SEX	AGE	BILL_AMT_SEPT	BILL_AMT_AUG	BILL_AMT_JUL	BILL_AMT_JUI
0	1	20000	0	24	3913	3102	689	
1	2	120000	0	26	2682	1725	2682	327
2	3	90000	0	34	29239	14027	13559	1433
3	4	50000	0	37	46990	48233	49291	2831
4	5	50000	1	57	8617	5670	35835	2094

5 rows × 85 columns

```
In [ ]: 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()
```

Out[ ]:

	Unnamed: 0	LIMIT_BAL	SEX	AGE	BILL_AMT_SEPT	BILL_AMT_AUG	BILL_AMT_JUL	BILL_/
0	0	20000	0	24	3913	3102	689	
1	1	120000	0	26	2682	1725	2682	
2	2	90000	0	34	29239	14027	13559	
3	3	50000	0	37	46990	48233	49291	
4	4	50000	1	57	8617	5670	35835	

5 rows × 85 columns

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

Implementing Logistic Regression

```
In [ ]: df_log_reg = df_fr.copy()
```

```
In [ ]: df_log_reg.head()
```

Out[ ]:

	LIMIT_BAL	SEX	AGE	BILL_AMT_SEPT	BILL_AMT_AUG	BILL_AMT_JUL	BILL_AMT_JUN	E
0	20000	0	24	3913	3102	689	0	
1	120000	0	26	2682	1725	2682	3272	
2	90000	0	34	29239	14027	13559	14331	
3	50000	0	37	46990	48233	49291	28314	
4	50000	1	57	8617	5670	35835	20940	

5 rows × 84 columns

```
In [ ]: 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)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, r
```

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

In [ ]:
```
grid_lr_clf = GridSearchCV(LogisticRegression(), param_grid, scoring = 'acc
grid_lr_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 14 candidates, totalling 42 fits
[CV 3/3] END .................C=0.001, penalty=l1;, score=nan total time=
0.1s
[CV 1/3] END .................C=0.001, penalty=l1;, score=nan total time=
0.1s
[CV 2/3] END .................C=0.001, penalty=l1;, score=nan total time=
0.1s
[CV 1/3] END ..................C=0.01, penalty=l1;, score=nan total time=
0.0s
[CV 2/3] END ..................C=0.01, penalty=l1;, score=nan total time=
0.0s
[CV 3/3] END ..................C=0.01, penalty=l1;, score=nan total time=
0.0s
[CV 2/3] END ...............C=0.001, penalty=l2;, score=1.000 total time=
5.2s
[CV 3/3] END ...............C=0.001, penalty=l2;, score=1.000 total time=
5.1s
[CV 1/3] END ................C=0.01, penalty=l2;, score=1.000 total time=
5.0s
[CV 1/3] END ...............C=0.001, penalty=l2;, score=1.000 total time=
5.4s
[CV 1/3] END ...................C=0.1, penalty=l1;, score=nan total time=
0.1s
[CV 2/3] END ...................C=0.1, penalty=l1;, score=nan total time=
0.1s
[CV 3/3] END ...................C=0.1, penalty=l1;, score=nan total time=
0.1s
[CV 3/3] END ................C=0.01, penalty=l2;, score=1.000 total time=
0.4s
[CV 2/3] END ................C=0.01, penalty=l2;, score=1.000 total time=
0.5s
[CV 1/3] END .....................C=1, penalty=l1;, score=nan total time=
0.0s
[CV 2/3] END .....................C=1, penalty=l1;, score=nan total time=
0.0s
[CV 3/3] END .....................C=1, penalty=l1;, score=nan total time=
0.0s
[CV 1/3] END .................C=0.1, penalty=l2;, score=1.000 total time=
0.4s
[CV 3/3] END .................C=0.1, penalty=l2;, score=1.000 total time=
0.4s
[CV 2/3] END .................C=0.1, penalty=l2;, score=1.000 total time=
0.7s
[CV 1/3] END ....................C=10, penalty=l1;, score=nan total time=
0.1s
[CV 2/3] END ....................C=10, penalty=l1;, score=nan total time=
0.0s
[CV 3/3] END ....................C=10, penalty=l1;, score=nan total time=
0.0s
[CV 1/3] END ...................C=1, penalty=l2;, score=1.000 total time=
0.6s
[CV 2/3] END ...................C=1, penalty=l2;, score=1.000 total time=
0.7s
[CV 3/3] END ...................C=1, penalty=l2;, score=1.000 total time=
0.7s
[CV 1/3] END ...................C=100, penalty=l1;, score=nan total time=
0.1s
[CV 2/3] END ...................C=100, penalty=l1;, score=nan total time=
0.0s
[CV 3/3] END ...................C=100, penalty=l1;, score=nan total time=
0.0s
[CV 1/3] END ..................C=10, penalty=l2;, score=1.000 total time=
0.6s
[CV 3/3] END ..................C=10, penalty=l2;, score=1.000 total time=
```

```
0.6s
[CV 2/3] END .................C=10, penalty=l2;, score=1.000 total time=
0.9s
[CV 1/3] END ................C=100, penalty=l2;, score=1.000 total time=
0.5s
[CV 2/3] END ................C=100, penalty=l2;, score=1.000 total time=
0.5s
[CV 3/3] END .................C=1000, penalty=l1;, score=nan total time=
0.1s
[CV 2/3] END .................C=1000, penalty=l1;, score=nan total time=
0.2s
[CV 3/3] END ................C=100, penalty=l2;, score=1.000 total time=
0.7s
[CV 1/3] END ...............C=1000, penalty=l2;, score=1.000 total time=
0.4s
[CV 1/3] END .................C=1000, penalty=l1;, score=nan total time=
0.8s
[CV 2/3] END ...............C=1000, penalty=l2;, score=1.000 total time=
0.4s
[CV 3/3] END ...............C=1000, penalty=l2;, score=1.000 total time=
0.3s
```

Out[ ]:

```
▸ GridSearchCV
▸ estimator: LogisticRegression
  ▸ LogisticRegression
```

In [ ]:

```
optimized_clf = grid_lr_clf.best_estimator_
```

In [ ]:

```
grid_lr_clf.best_params_
```

Out[ ]:

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

In [ ]:

```
grid_lr_clf.best_score_
```

Out[ ]:

```
1.0
```

In [ ]:

```
# 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
```

In [ ]:

```
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
```

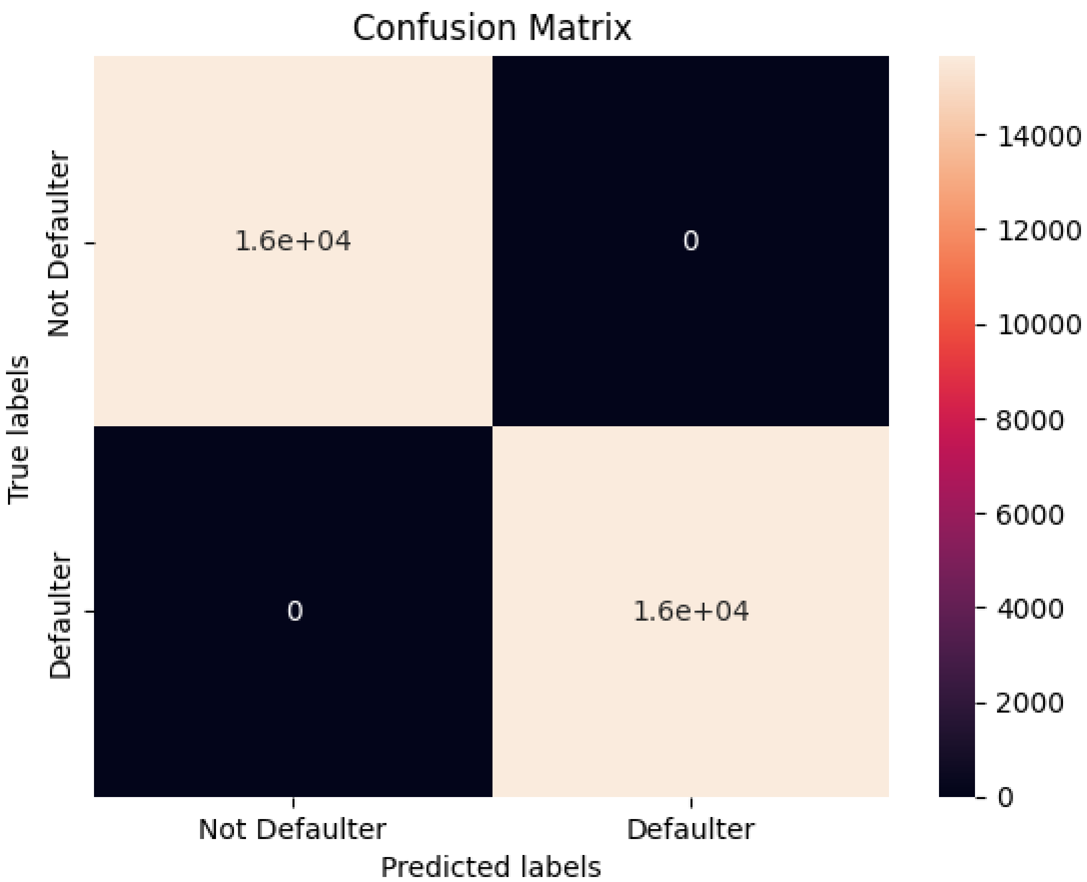
In [ ]:

```
# 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]]
```

Out[ ]:

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



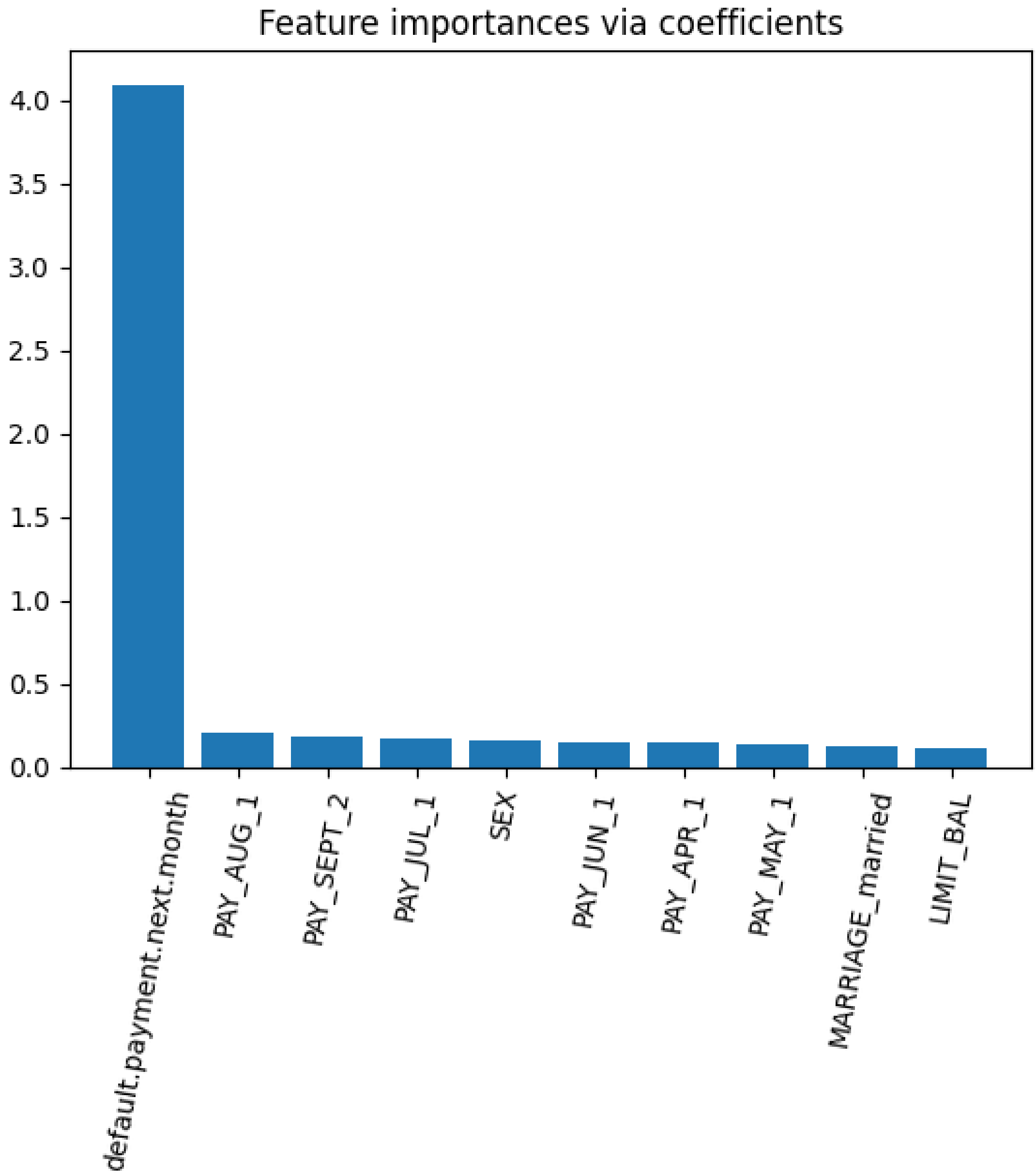
In [ ]:

```
feature_importance = pd.DataFrame({'Features':columns, 'Importance':np.abs(
```

In [ ]:

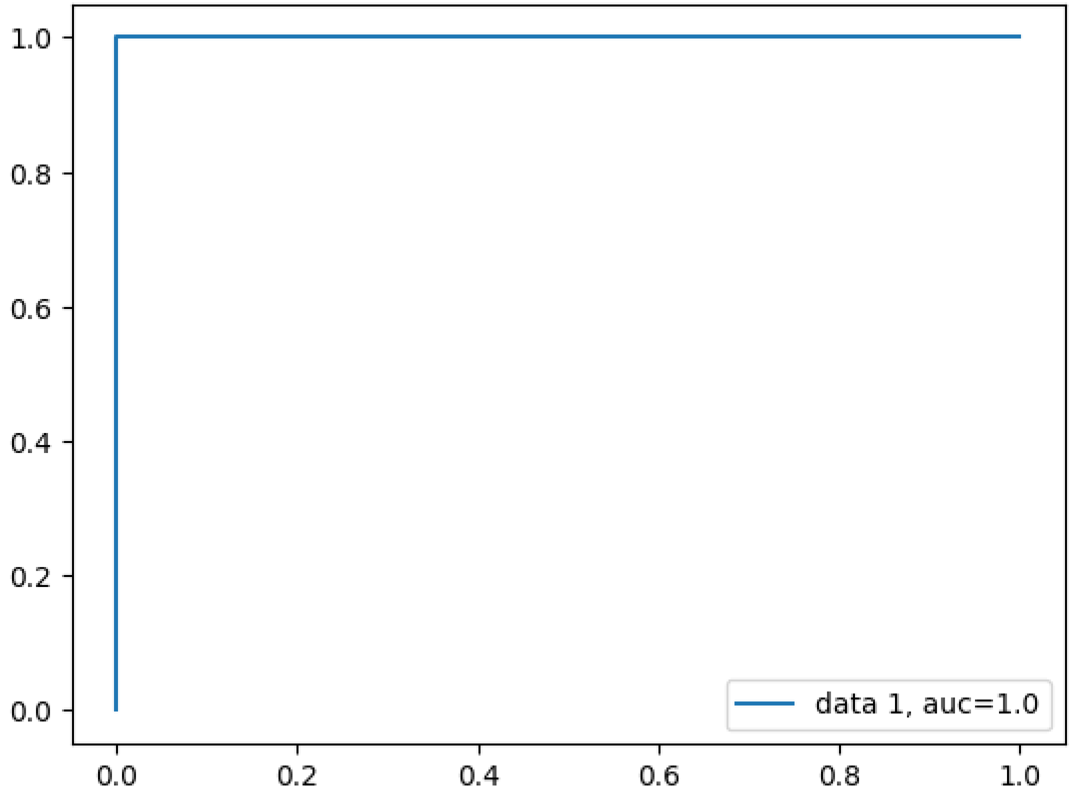
```
feature_importance = feature_importance.sort_values(by = 'Importance', asce
```

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



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

In [ ]:
```
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.

Implementing SVC

In [ ]:
```
from sklearn.model_selection import GridSearchCV
```

In [ ]:
```
# 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)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, r
```

In [ ]:
```
grid_clf = GridSearchCV(SVC(probability=True), param_grid, scoring = 'accur
grid_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
[CV 1/3] END ...................C=1, kernel=rbf;, score=0.993 total time=
1.3min
[CV 2/3] END ...................C=1, kernel=rbf;, score=0.995 total time=
1.4min
[CV 3/3] END .................C=0.1, kernel=rbf;, score=0.988 total time=
3.0min
[CV 2/3] END .................C=0.1, kernel=rbf;, score=0.987 total time=
3.2min
[CV 1/3] END .................C=0.1, kernel=rbf;, score=0.985 total time=
3.4min
[CV 1/3] END ..................C=10, kernel=rbf;, score=0.995 total time=
1.0min
[CV 3/3] END ...................C=1, kernel=rbf;, score=0.995 total time=
1.4min
[CV 2/3] END ..................C=10, kernel=rbf;, score=0.997 total time=
1.1min
[CV 3/3] END ..................C=10, kernel=rbf;, score=0.997 total time=
1.1min
[CV 1/3] END .................C=100, kernel=rbf;, score=0.995 total time=
48.0s
[CV 2/3] END .................C=100, kernel=rbf;, score=0.997 total time=
50.6s
[CV 3/3] END .................C=100, kernel=rbf;, score=0.997 total time=
47.7s
```

Out[ ]:

▸ **GridSearchCV**

▸ **estimator: SVC**

▸ SVC

In [ ]:

```
optimal_SVC_clf = grid_clf.best_estimator_
```

In [ ]:

```
grid_clf.best_params_
```

Out[ ]:

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

In [ ]:

```
grid_clf.best_score_
```

Out[ ]:

```
0.9964544823494704
```

In [ ]:

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

In [ ]:

```
# 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
```

In [ ]:

```
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  0.9968873613903119
The precision on test data is  0.9990920881971466
The recall on test data is  0.9947055785123967
The f1 on test data is  0.9968940080238126
The roc_score on test data is  0.9968968820007601
```

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.

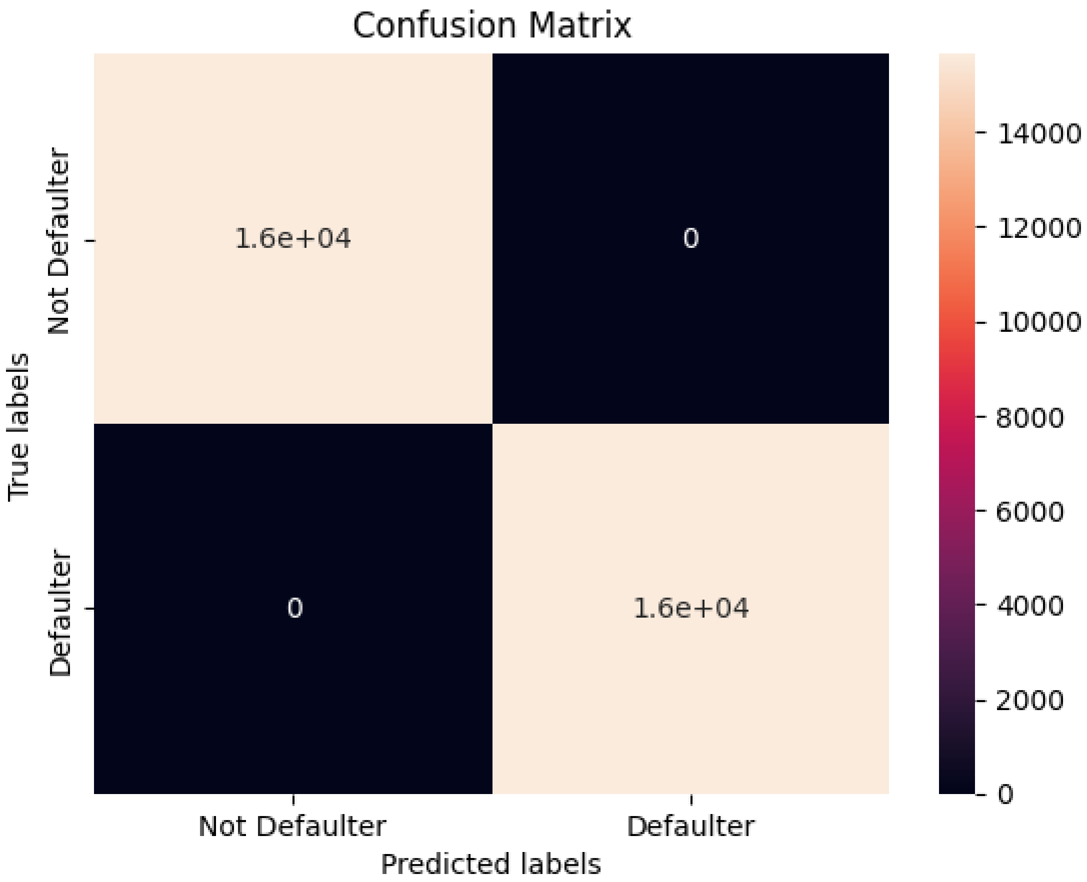
In [ ]:

```
 # 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]]
```

Out[ ]:

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



In [ ]:
```
import torch
```

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

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

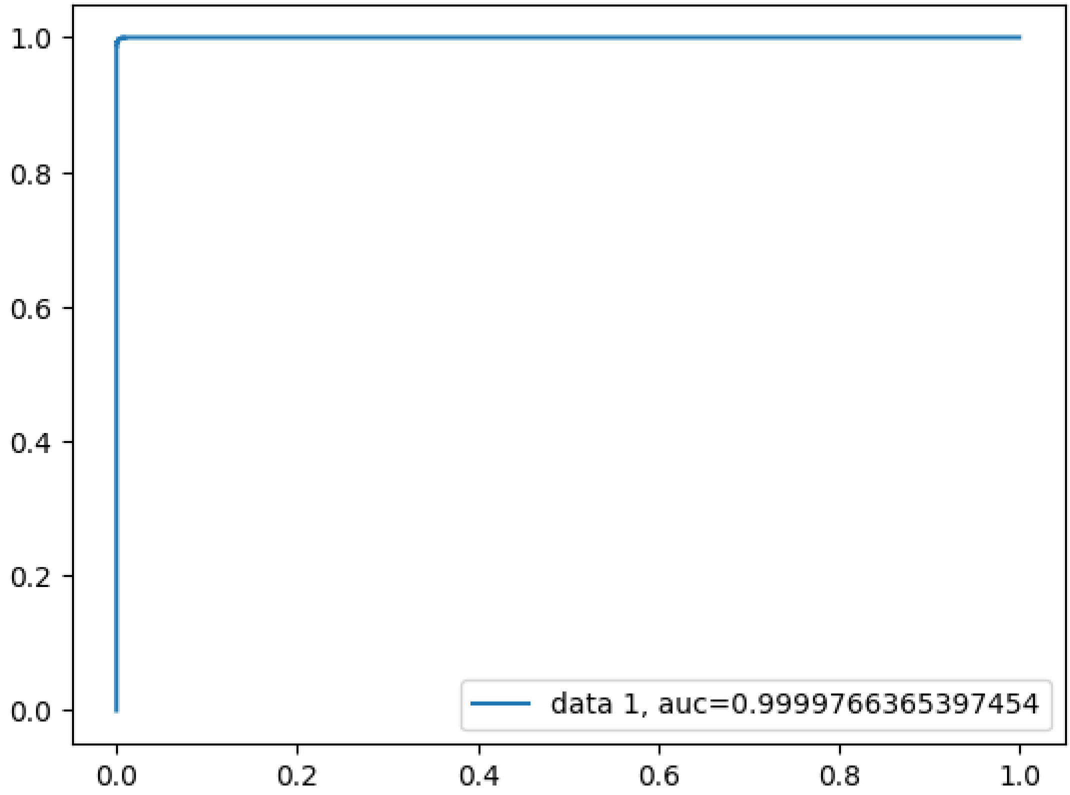
In [ ]:
```
optimal_SVC_clf
```

Out[ ]:
```
▾ SVC
SVC(C=100, probability=True)
```

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

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

In [ ]:
```
# 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.

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

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
```

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

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, r
```

In [ ]:
```
grid_DTC_clf = GridSearchCV(DecisionTreeClassifier(), param_grid, scoring =
grid_DTC_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
[CV 1/3] END max_depth=20, min_samples_split=0.1;, score=1.000 total time=
0.2s
[CV 2/3] END max_depth=20, min_samples_split=0.1;, score=1.000 total time=
0.2s
[CV 3/3] END max_depth=20, min_samples_split=0.1;, score=1.000 total time=
0.2s
[CV 2/3] END max_depth=20, min_samples_split=0.2;, score=1.000 total time=
0.1s
[CV 1/3] END max_depth=20, min_samples_split=0.2;, score=1.000 total time=
0.1s
[CV 3/3] END max_depth=20, min_samples_split=0.2;, score=1.000 total time=
0.1s
[CV 2/3] END max_depth=20, min_samples_split=0.4;, score=1.000 total time=
0.1s
[CV 1/3] END max_depth=20, min_samples_split=0.4;, score=1.000 total time=
0.1s
[CV 3/3] END max_depth=20, min_samples_split=0.4;, score=1.000 total time=
0.1s
[CV 1/3] END max_depth=30, min_samples_split=0.1;, score=1.000 total time=
0.1s
[CV 2/3] END max_depth=30, min_samples_split=0.1;, score=1.000 total time=
0.1s
[CV 1/3] END max_depth=30, min_samples_split=0.2;, score=1.000 total time=
0.1s
[CV 3/3] END max_depth=30, min_samples_split=0.1;, score=1.000 total time=
0.2s
[CV 2/3] END max_depth=30, min_samples_split=0.2;, score=1.000 total time=
0.1s
[CV 1/3] END max_depth=30, min_samples_split=0.4;, score=1.000 total time=
0.1s
[CV 3/3] END max_depth=30, min_samples_split=0.2;, score=1.000 total time=
0.1s
[CV 3/3] END max_depth=30, min_samples_split=0.4;, score=1.000 total time=
0.1s
[CV 2/3] END max_depth=30, min_samples_split=0.4;, score=1.000 total time=
0.1s
[CV 1/3] END max_depth=50, min_samples_split=0.1;, score=1.000 total time=
0.1s
[CV 2/3] END max_depth=50, min_samples_split=0.1;, score=1.000 total time=
0.1s
[CV 3/3] END max_depth=50, min_samples_split=0.1;, score=1.000 total time=
0.1s
[CV 1/3] END max_depth=50, min_samples_split=0.2;, score=1.000 total time=
0.1s
[CV 2/3] END max_depth=50, min_samples_split=0.2;, score=1.000 total time=
0.1s
[CV 3/3] END max_depth=50, min_samples_split=0.2;, score=1.000 total time=
0.1s
[CV 1/3] END max_depth=50, min_samples_split=0.4;, score=1.000 total time=
0.1s
[CV 3/3] END max_depth=50, min_samples_split=0.4;, score=1.000 total time=
0.1s
[CV 2/3] END max_depth=50, min_samples_split=0.4;, score=1.000 total time=
0.1s
[CV 1/3] END max_depth=100, min_samples_split=0.1;, score=1.000 total time=
0.1s
[CV 2/3] END max_depth=100, min_samples_split=0.1;, score=1.000 total time=
0.1s
[CV 3/3] END max_depth=100, min_samples_split=0.1;, score=1.000 total time=
0.1s
[CV 2/3] END max_depth=100, min_samples_split=0.2;, score=1.000 total time=
0.1s
[CV 1/3] END max_depth=100, min_samples_split=0.2;, score=1.000 total time=
```

```
0.2s
[CV 3/3] END max_depth=100, min_samples_split=0.2;, score=1.000 total time=
0.1s
[CV 2/3] END max_depth=100, min_samples_split=0.4;, score=1.000 total time=
0.1s
[CV 1/3] END max_depth=100, min_samples_split=0.4;, score=1.000 total time=
0.1s[CV 3/3] END max_depth=100, min_samples_split=0.4;, score=1.000 total t
ime=   0.1s
```

Out[ ]:

```
▸ GridSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [ ]:

```
grid_DTC_clf.best_score_
```

Out[ ]:

```
1.0
```

In [ ]:

```
optimal_DTC_clf = grid_DTC_clf.best_estimator_
```

In [ ]:

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

In [ ]:

```
grid_DTC_clf.best_params_
```

Out[ ]:

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

In [ ]:

```
# 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

In [ ]:

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

In [ ]:

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

In [ ]:

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

Out[ ]:

```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [ ]:

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

In [ ]:
```
# 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  0.9999351533622982
```

In [ ]:
```
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  0.9999351533622982
The precision on test data is  1.0
The recall on test data is  0.9998703151342239
The f1 on test data is  0.9999351533622982
The roc_score on test data is  0.9999351575671119
```

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.

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

In [ ]:
```
grid_rf_clf = GridSearchCV(RandomForestClassifier(), param_grid, scoring =
grid_rf_clf.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 9 candidates, totalling 27 fits
[CV 1/3] END ....max_depth=10, n_estimators=100;, score=1.000 total time=
3.8s
[CV 3/3] END ....max_depth=10, n_estimators=100;, score=1.000 total time=
4.5s
[CV 2/3] END ....max_depth=10, n_estimators=100;, score=1.000 total time=
5.0s
[CV 2/3] END ....max_depth=10, n_estimators=150;, score=1.000 total time=
7.0s
[CV 3/3] END ....max_depth=10, n_estimators=150;, score=1.000 total time=
6.8s
[CV 1/3] END ....max_depth=10, n_estimators=200;, score=1.000 total time=
10.0s
[CV 1/3] END ....max_depth=10, n_estimators=150;, score=1.000 total time=
7.3s
[CV 2/3] END ....max_depth=10, n_estimators=200;, score=1.000 total time=
9.7s
[CV 3/3] END ....max_depth=10, n_estimators=200;, score=1.000 total time=
9.8s
[CV 1/3] END ....max_depth=20, n_estimators=100;, score=1.000 total time=
6.9s
[CV 2/3] END ....max_depth=20, n_estimators=100;, score=1.000 total time=
6.1s
[CV 3/3] END ....max_depth=20, n_estimators=100;, score=1.000 total time=
5.1s
[CV 3/3] END ....max_depth=20, n_estimators=150;, score=1.000 total time=
8.4s
[CV 1/3] END ....max_depth=20, n_estimators=150;, score=1.000 total time=
10.0s
[CV 2/3] END ....max_depth=20, n_estimators=150;, score=1.000 total time=
9.5s
[CV 1/3] END ....max_depth=20, n_estimators=200;, score=1.000 total time=
9.7s
[CV 1/3] END ....max_depth=30, n_estimators=100;, score=1.000 total time=
5.6s
[CV 2/3] END ....max_depth=30, n_estimators=100;, score=1.000 total time=
5.8s
[CV 2/3] END ....max_depth=20, n_estimators=200;, score=1.000 total time=
10.3s
[CV 3/3] END ....max_depth=30, n_estimators=100;, score=1.000 total time=
6.3s
[CV 3/3] END ....max_depth=20, n_estimators=200;, score=1.000 total time=
13.6s
[CV 2/3] END ....max_depth=30, n_estimators=150;, score=1.000 total time=
7.7s
[CV 1/3] END ....max_depth=30, n_estimators=150;, score=1.000 total time=
10.3s
[CV 3/3] END ....max_depth=30, n_estimators=150;, score=1.000 total time=
9.3s
[CV 1/3] END ....max_depth=30, n_estimators=200;, score=1.000 total time=
11.8s
[CV 2/3] END ....max_depth=30, n_estimators=200;, score=1.000 total time=
9.3s
[CV 3/3] END ....max_depth=30, n_estimators=200;, score=1.000 total time=
8.2s
```

Out[ ]:

▸ **GridSearchCV**

▸ **estimator: RandomForestClassifier**

▸ RandomForestClassifier

In [ ]:

```
grid_rf_clf.best_score_
```

Out[ ]:
```
1.0
```

In [ ]:
```
grid_rf_clf.best_params_
```

Out[ ]:
```
{'max_depth': 30, 'n_estimators': 100}
```

In [ ]:
```
optimal_rf_clf = grid_rf_clf.best_estimator_
```

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

In [ ]:
```
# 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
```

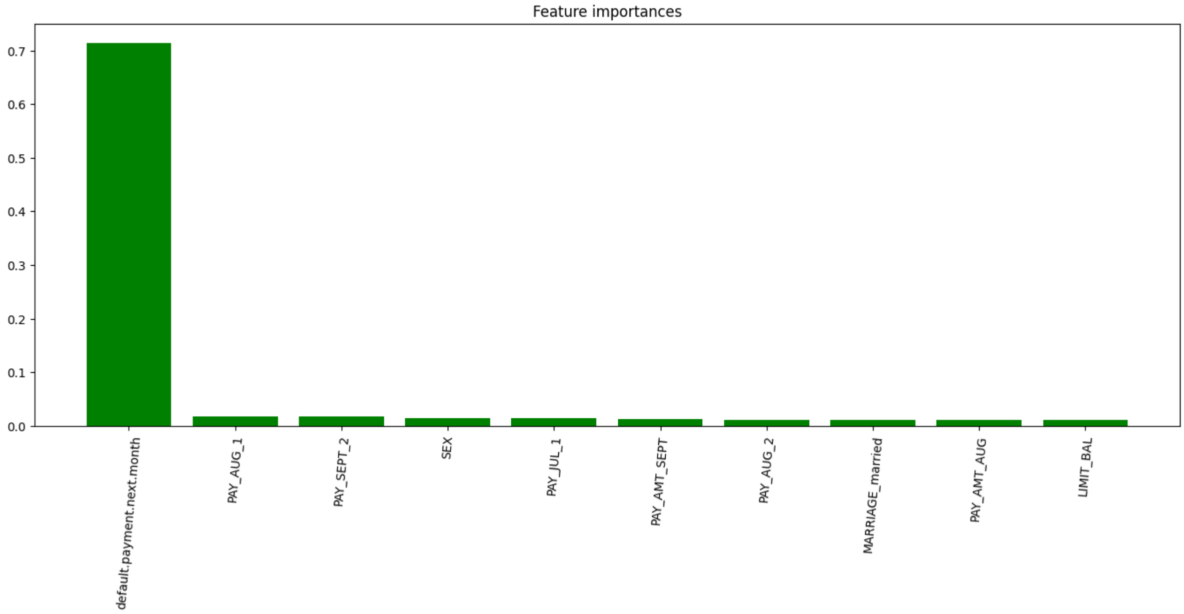
In [ ]:
```
test_accuracy_rf = accuracy_score(test_class_preds,y_test)
test_precision_score_rf = precision_score(test_class_preds,y_test)
test_recall_score_rf = recall_score(test_class_preds,y_test)
test_f1_score_rf = f1_score(test_class_preds,y_test)
test_roc_score_rf = roc_auc_score(test_class_preds,y_test)

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
```

In [ ]:
```
len(optimal_rf_clf.feature_importances_)
```

Out[ ]:
```
81
```

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

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()
```



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

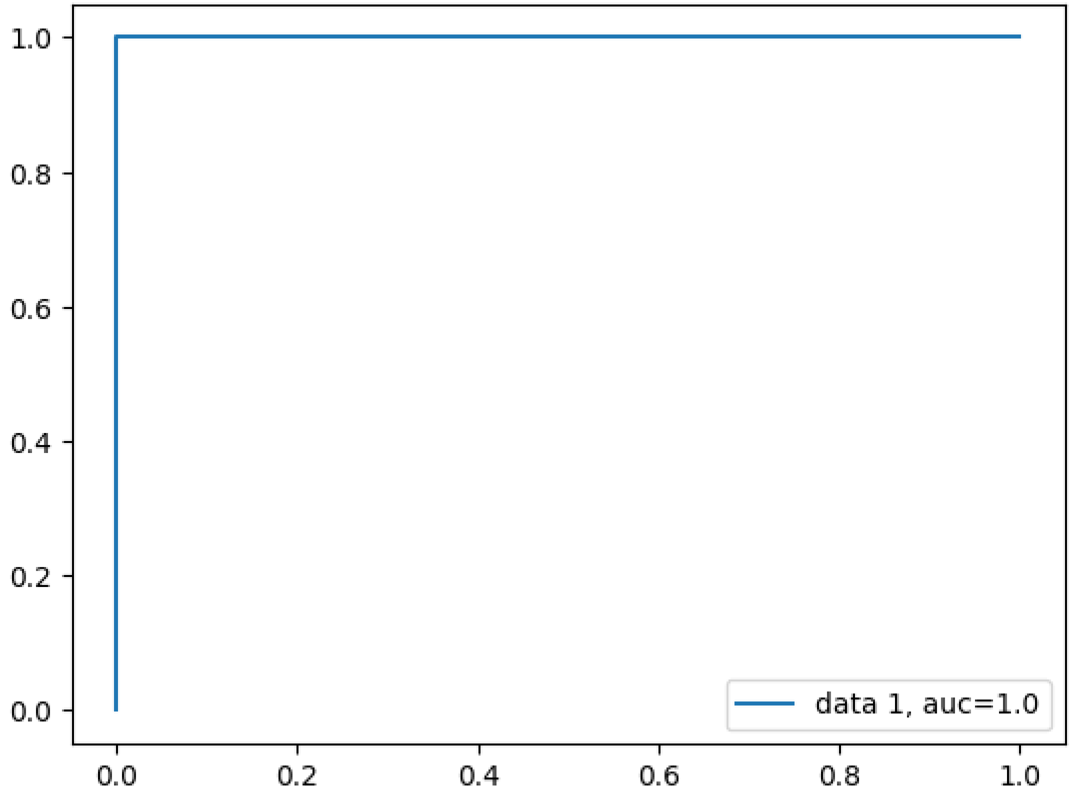
In [ ]:
```
model_save_name = 'rf_optimized_classifier.pt'
path = F"./{model_save_name}"
optimal_rf_clf = torch.load(path)
```

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

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

In [ ]:
```
import sklearn.metrics as metrics
```

In [ ]:
```
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

In [ ]:
```
#import lightgbm and xgboost
import lightgbm as lgb
import xgboost as xgb
```

Applying XGBoost

In [ ]:
```
#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)
```

In [ ]:
```
#setting parameters for xgboost
parameters={'max_depth':7, 'eta':1, 'silent':1,'objective':'binary:logistic
```

In [ ]:
```
#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 languag
e bindings but
  then being mistakenly passed down to XGBoost core, or some parameter actu
ally being used
  but getting flagged wrongly here. Please open an issue if you find any su
ch cases.
```

In [ ]:

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

Out[ ]:

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

In [ ]:

```
#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)
```

In [ ]:

```
len(train_class_preds_probs)
```

Out[ ]:

```
31307
```

In [ ]:

```
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)
```

In [ ]:

```
test_class_preds_probs[:20]
```

Out[ ]:

```
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)
```

In [ ]:

```
test_class_preds[:20]
```

Out[ ]:

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

In [ ]:

```
len(y_train)
```

Out[ ]:

```
31307
```

In [ ]:

```
len(train_class_preds)
```

Out[ ]:

```
31307
```

In [ ]:
```
# 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
```

In [ ]:
```
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

In [ ]:
```
from xgboost import  XGBClassifier
```

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

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, r
```

In [ ]:
```
param_test1 = {
 'max_depth':range(3,10,2),
 'min_child_weight':range(1,6,2)
}
gsearch1 = GridSearchCV(estimator = XGBClassifier( learning_rate =0.1, n_es
 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
```

Out[ ]:
```
GridSearchCV(cv=3,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=Non
e,
                                     colsample_bynode=None,
                                     colsample_bytree=0.8,
                                     early_stopping_rounds=None,
                                     enable_categorical=False, eval_metric=
None,
                                     gamma=0, gpu_id=None, grow_policy=Non
e,
                                     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=Non
e,
                                     n_estimators=140, n_jobs=None, nthread
=4,
                                     num_parallel_tree=None, predictor=Non
e,
                                     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)
```

In [ ]:
```
gsearch1.best_score_
```

Out[ ]:
```
1.0
```

In [ ]:
```
optimal_xgb = gsearch1.best_estimator_
```

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

In [ ]:
```
# 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
```

In [ ]:
```
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)
```

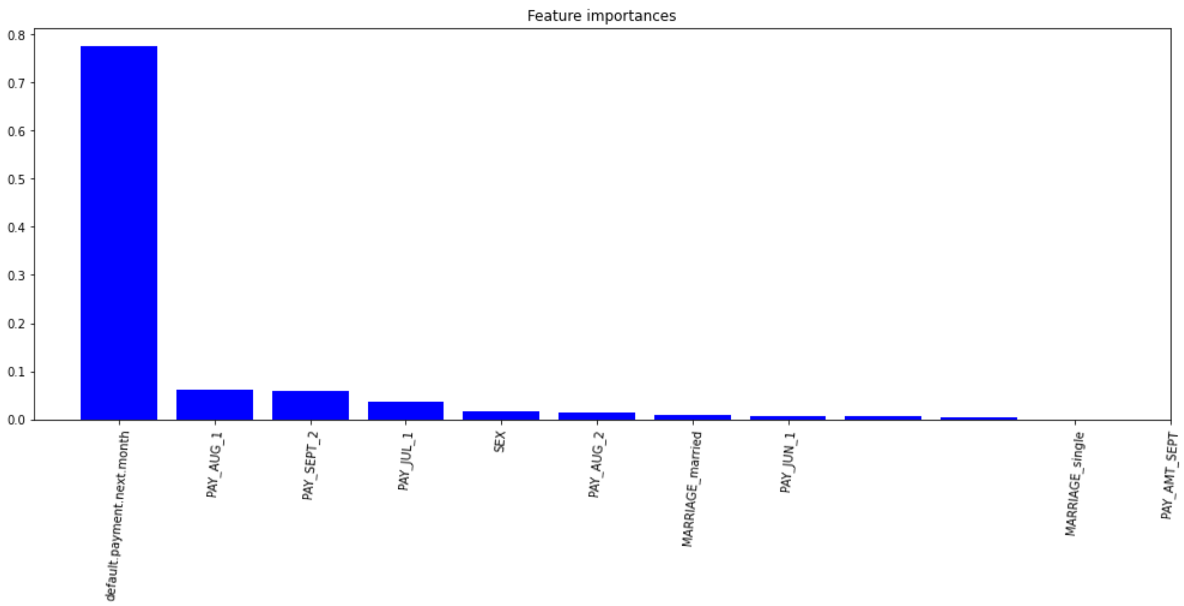
```
The accuracy on test data is  1.0
The precision on test data is  1.0
The recall on test data is  1.0
The f1 on test data is  1.0
The roc_score on train data is  1.0
```

In [ ]:
```
pd.DataFrame(optimal_xgb.feature_importances_,
                                   index = columns,
                                    columns=['importance_xgb']).sort_values
                                                                       asc
```

Out[ ]:

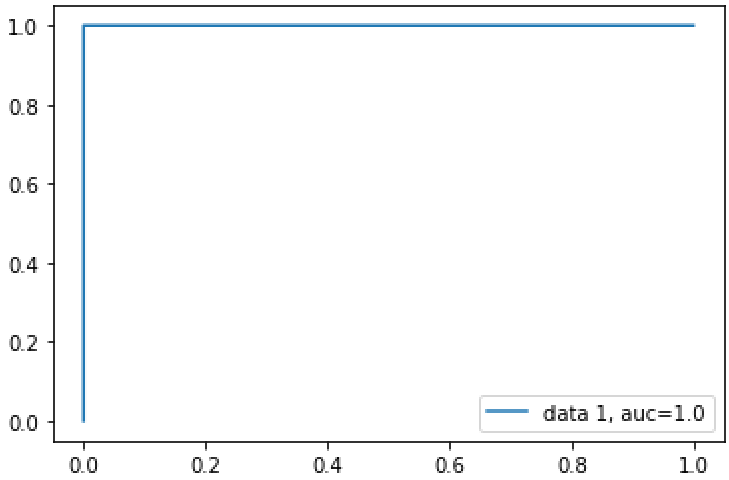
	importance_xgb
default.payment.next.month	0.774541
PAY_AUG_1	0.061343
PAY_SEPT_2	0.058241
PAY_JUL_1	0.036967
SEX	0.016271
PAY_AUG_2	0.014785
MARRIAGE_married	0.008888
PAY_JUN_1	0.006657
PAY_JUL_-1	0.005823
PAY_SEPT_1	0.005516

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

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



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

In [ ]:
```
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()
```



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

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

Evaluating the models

In [ ]:
```
recall_score
```

Out[ ]:
```
<function sklearn.metrics._classification.recall_score(y_true, y_pred, *, l
abels=None, pos_label=1, average='binary', sample_weight=None, zero_divisio
n='warn')>
```

In [ ]:
```
classifiers = ['Logistic Regression', 'SVC', 'Random Forest CLf', 'Xgboost
train_accuracy = [train_accuracy_lr, train_accuracy_SVC, train_accuracy_rf,
test_accuracy = [test_accuracy_lr, test_accuracy_SVC, test_accuracy_rf, tes
precision_score = [test_precision_score_lr, test_precision_score_SVC, test_
recall_score = [test_recall_score_lr, test_recall_score_SVC, test_recall_sc
f1_score = [test_f1_score_lr, test_f1_score_SVC, test_f1_score_rf, test_f1_
```

In [ ]:
```
pd.DataFrame({'Classifier':classifiers, 'Train Accuracy': train_accuracy, '
```

Out[ ]:

	Classifier	Train Accuracy	Test Accuracy	Precision Score	Recall Score	F1 Score
0	Logistic Regression	1.0	1.000000	1.000000	1.000000	1.0000
1	SVC	1.0	0.996693	0.998962	0.994448	0.9967
2	Random Forest CLf	1.0	1.000000	1.000000	1.000000	1.0000
3	Xgboost Clf	1.0	1.000000	1.000000	1.000000	1.0000

Plotting ROC AUC for all the models

In [ ]:

```
classifiers_proba = [(optimized_clf, y_preds_proba_lr),
               (optimal_rf_clf, y_preds_proba_rf),
               (optimal_xgb, y_preds_proba_xgb),
               (optimal_SVC_clf,y_pred_proba_SVC)]

# Define a result table as a DataFrame
result_table = pd.DataFrame(columns=['classifiers', 'fpr','tpr','auc'])

# Train the models and record the results
for pair in classifiers_proba:

    fpr, tpr, _ = roc_curve(y_test,  pair[1])
    auc = roc_auc_score(y_test, pair[1])

    result_table = result_table.append({'classifiers':pair[0].__class__.__n
                                        'fpr':fpr,
                                        'tpr':tpr,
                                        'auc':auc}, ignore_index=True)

# Set name of the classifiers as index labels
result_table.set_index('classifiers', inplace=True)
```

In [ ]:

```
result_table
```

Out[ ]:

classifiers	fpr	tpr	auc
LogisticRegression	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.00012970168612191958, 0.15642023346303...	1.000000
RandomForestClassifier	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0009079118028534371, 0.001297016861219...	1.000000
XGBClassifier	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.0031128404669260703, 0.004150453955901...	1.000000
SVC	[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, ...	[0.0, 0.00012970168612191958, 0.21089494163424...	0.999977

In [ ]:

```
fig = plt.figure(figsize=(8,6))

for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
             label="{}, AUC={:.3f}".format(i, result_table.loc[i]['auc']))

plt.plot([0,1], [0,1], color='orange', linestyle='--')
```

```
plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("Flase Positive Rate", fontsize=15)

plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=15)

plt.title('ROC Curve Analysis', fontweight='bold', fontsize=15)
plt.legend(prop={'size':13}, loc='lower right')

plt.show()
```