Project Title : Credit Card Default Prediction

Project Type - Classification In Machine Learning

--->By Krishna from cohort Jerusalem in Almabetter

Contribution - **Individual**

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

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.

Loading the Data and libraries.

```
# Importing the libraries we'll need.
import numpy as np
```



```
%matplotlib inline
import seaborn as sns

import os
import sys
import warnings
warnings.filterwarnings('ignore')


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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.r
```

```
# Upgrading the python version to read the dataset which is in excel file format.
!pip install --upgrade xlrd
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Requirement already satisfied: xlrd in /usr/local/lib/python3.9/dist-packages (2.0.1)
```

```
# loading the data
df = pd.read_csv('/content/drive/MyDrive/csvfile/default of credit card clients.xls - Data
```

```
# checking what the data looks like
df.head()
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL
0	1	20000	2	2	1	24	2	2	-1	-1	...	
1	2	120000	2	2	2	26	-1	2	0	0	...	
2	3	90000	2	2	2	34	0	0	0	0	...	
3	4	50000	2	2	1	37	0	0	0	0	...	
4	5	50000	1	2	1	57	-1	0	-1	0	...	

5 rows × 25 columns

```
# checking the shape of the dataframe
df.shape
```

```
(30000, 25)
```

As we can see that we have around 30000 rows and 25 columns in our dataset.


```
# we can remedy this using set_option function.
pd.set_option('display.max_columns', None)
```

```
# Now, we should be able to see all columns. Checking the last few instances.
df.tail()
```

Understanding our features and the data it contains in detail.

1. ID: ID of each client (unique identifier)
2. LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit)
3. SEX: Gender (1=male, 2=female)
4. EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
5. MARRIAGE: Marital status (1=married, 2=single, 3=others)
6. AGE: Age in years
7. PAY_0: Repayment status in September, 2005 (-2 = Unused,-1=pay duly,0=Revolving Credit, 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)
8. PAY_2: Repayment status in August, 2005 (scale same as above)
9. PAY_3: Repayment status in July, 2005 (scale same as above)
10. PAY_4: Repayment status in June, 2005 (scale same as above)
11. PAY_5: Repayment status in May, 2005 (scale same as above)
12. PAY_6: Repayment status in April, 2005 (scale same as above)
13. BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)
14. BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)

17. BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)
18. BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)
19. PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)
20. PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)
21. PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)
22. PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)
23. PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)
24. PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)
25. default.payment.next.month: Default payment (1=yes, 0=no)

Data Preprocessing.

```
# Let's rename the columns for better understanding.
df.rename(columns={'PAY_0':'REPAY_STATUS_SEPT','PAY_2':'REPAY_STATUS_AUG','PAY_3':
```

```
                      'REPAY_STATUS_JUL','PAY_4':'REPAY_STATUS_JUN','PAY_5':'REPAY_STATUS_MAY

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_

df.rename(columns={'PAY_AMT1':'PRE_PAY_AMT_SEPT','PAY_AMT2':'PRE_PAY_AMT_AUG','PAY_AMT3':'
                      'PAY_AMT4':'PRE_PAY_AMT_JUN','PAY_AMT5':'PRE_PAY_AMT_MAY','PAY_AMT6':'P
```

```
# checking for null values in our dataframe.
df.isna().sum()
```

```
ID                            0
LIMIT_BAL                     0
SEX                           0
EDUCATION                     0
MARRIAGE                      0
AGE                           0
REPAY_STATUS_SEPT             0
REPAY_STATUS_AUG              0
REPAY_STATUS_JUL              0
REPAY_STATUS_JUN              0
REPAY_STATUS_MAY              0
REPAY_STATUS_APR              0
BILL_AMT_SEPT                 0
BILL_AMT_AUG                  0
BILL_AMT_JUL                  0
BILL_AMT_JUN                  0
BILL_AMT_MAY                  0
BILL_AMT_APR                  0
PRE_PAY_AMT_SEPT              0
PRE_PAY_AMT_AUG               0
PRE_PAY_AMT_JUL               0
```



```
PRE_PAY_AMT_APR               0
default payment next month    0
dtype: int64
```

We can clearly see from above that there are no null values in our dataset.

```
# checking some basic info about our dataset.
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  int64
 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   REPAY_STATUS_SEPT           30000 non-null  int64
```

```
 7   REPAY_STATUS_AUG            30000 non-null  int64
 8   REPAY_STATUS_JUL            30000 non-null  int64
 9   REPAY_STATUS_JUN            30000 non-null  int64
 10  REPAY_STATUS_MAY            30000 non-null  int64
 11  REPAY_STATUS_APR            30000 non-null  int64
 12  BILL_AMT_SEPT               30000 non-null  int64
 13  BILL_AMT_AUG                30000 non-null  int64
 14  BILL_AMT_JUL                30000 non-null  int64
 15  BILL_AMT_JUN                30000 non-null  int64
 16  BILL_AMT_MAY                30000 non-null  int64
 17  BILL_AMT_APR                30000 non-null  int64
 18  PRE_PAY_AMT_SEPT            30000 non-null  int64
 19  PRE_PAY_AMT_AUG             30000 non-null  int64
 20  PRE_PAY_AMT_JUL             30000 non-null  int64
 21  PRE_PAY_AMT_JUN             30000 non-null  int64
 22  PRE_PAY_AMT_MAY             30000 non-null  int64
 23  PRE_PAY_AMT_APR             30000 non-null  int64
 24  default payment next month  30000 non-null  int64
dtypes: int64(25)
memory usage: 5.7 MB
```

```
# Checking some desriptive statistics.
df.describe(include='all')
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.00
mean	15000.500000	167484.322667	1.603733	1.853133	1.551867	35.48
std	8660.398374	129747.661567	0.489129	0.790349	0.521970	9.21
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
50%	15000.500000	140000.000000	2.000000	2.000000	2.000000	34.00
75%	22500.250000	240000.000000	2.000000	2.000000	2.000000	41.00
max	30000.000000	1000000.000000	2.000000	6.000000	3.000000	79.00


```
df.head()
```

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	REPAY_STATUS_SEPT	REPAY_STATUS_AUG
0	1	20000	2	2	1	24	2	2

```
# checking for duplicate data in our df.
print(len(df[df.duplicated()]))
df[df.duplicated()]
```

```
0
```

ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	REPAY_STATUS_SEPT	REPAY_STATUS_AUG

So there is no duplicate data in our dataframe.

```
# now let's save this data before operating on it.
credit_card_df = df.copy()
```

EXPLORATORY DATA ANALYSIS

```
# Although the data in our df in all numerical, there are some categorical variables prese
# Exploring our dependent variable.
```

Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

We can see from the above graph and value counts, that we have a unbalanced dataset. The no. of instances for class 0 is significantly higher than class 1

```
df.columns
```

```
Index(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE',
       'REPAY_STATUS_SEPT', 'REPAY_STATUS_AUG', 'REPAY_STATUS_JUL',
       'REPAY_STATUS_JUN', 'REPAY_STATUS_MAY', 'REPAY_STATUS_APR',
       'BILL_AMT_SEPT', 'BILL_AMT_AUG', 'BILL_AMT_JUL', 'BILL_AMT_JUN',
       'BILL_AMT_MAY', 'BILL_AMT_APR', 'PRE_PAY_AMT_SEPT', 'PRE_PAY_AMT_AUG',
       'PRE_PAY_AMT_JUL', 'PRE_PAY_AMT_JUN', 'PRE_PAY_AMT_MAY',
       'PRE_PAY_AMT_APR', 'is_defaulter'],
      dtype='object')
```

```
# Now, we have several other categorical columns like marriage, education, sex.
# Let's check them and see the relationship with our dependent variable.
df['MARRIAGE'].value_counts()
```

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

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

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

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

```
2    14030
1    10585
3     4917
5      280
4      123
6       51
0       14
Name: EDUCATION, dtype: int64
```

In the education variable, as per our data description, 1 refers to graduate school, 2 refers to university etc. however we have no understanding of some numbers present. so we will replace these will others.

Similarly, in our marriage variable, there is a 0 value which has unknown meaning. so we will add that to others.

Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

```
w
# we do this by creating another df by copying a slice of current df.

cat_var_df = df[['SEX','EDUCATION','MARRIAGE','is_defaulter']].copy()

cat_var_df.replace({'SEX': {1 : 'MALE', 2 : 'FEMALE'},
                    'EDUCATION' : {1 : 'graduate school', 2 : 'university', 3 : 'high schoo
                    'MARRIAGE' : {1 : 'married', 2 : 'single', 3 : 'others', 0 : 'others'},
                    inplace = True )
```

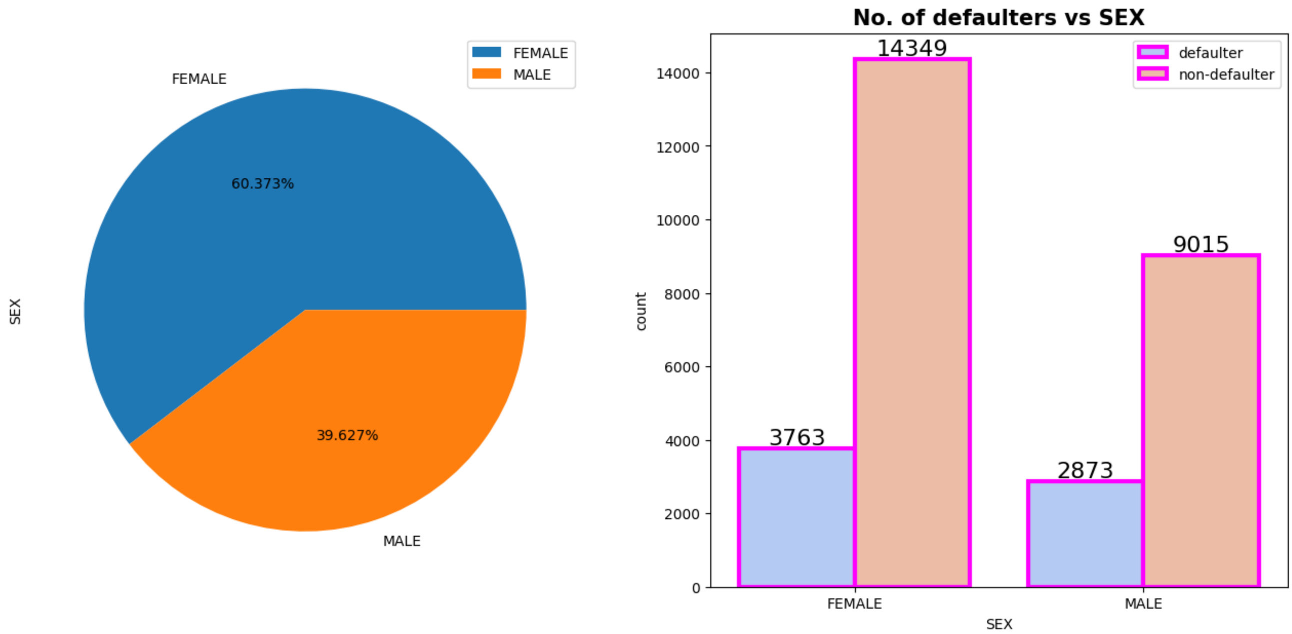
```
cat_var_df.head()
```

SEX EDUCATION MARRIAGE is_defaulter

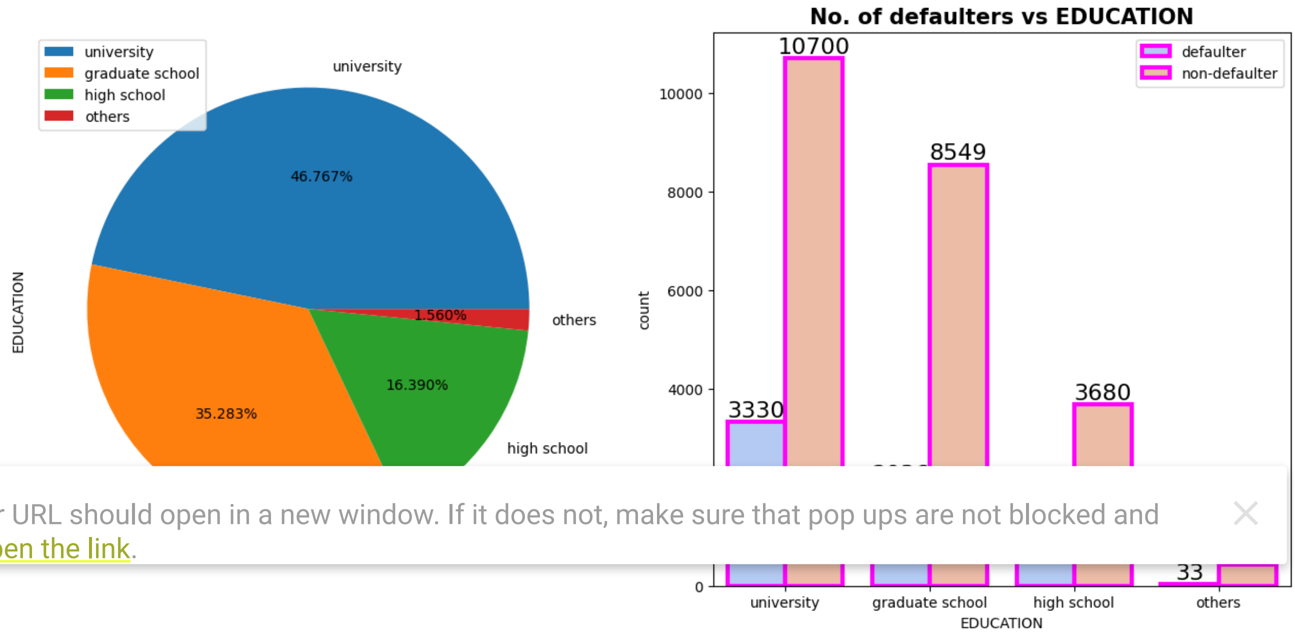
```
# Now Plotting the value counts of these categorical variables.
# Also visualizing the relationship of these variables with our dependent variable using s

for col in cat_var_df.columns[:-1]:
  plt.figure(figsize=(10,5))
  fig, axes = plt.subplots(ncols=2,figsize=(16,7))

  # Plotting the value counts of categorical variables using pie chart.
  cat_var_df[col].value_counts().plot(kind="pie",autopct='%1.3f%%',ax = axes[0],subplots=T

  # Plotting the relationship between above categorical features and our dependent variabl
  ax = sns.countplot(x=col, data=cat_var_df,  palette = 'coolwarm', hue="is_defaulter" ,ed

  # Setting the legend at the best location and setting the title.
  plt.legend(loc='best')
  plt.title(f'No. of defaulters vs {col}',weight ='bold', fontsize= 15)

# Annotating the counts in countplot charts.
  for p in ax.patches:
    height = p.get_height()
    ax.text(p.get_x()+p.get_width()/2, height+100, '{:1.0f}'.format(height),ha = "center",
```

Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

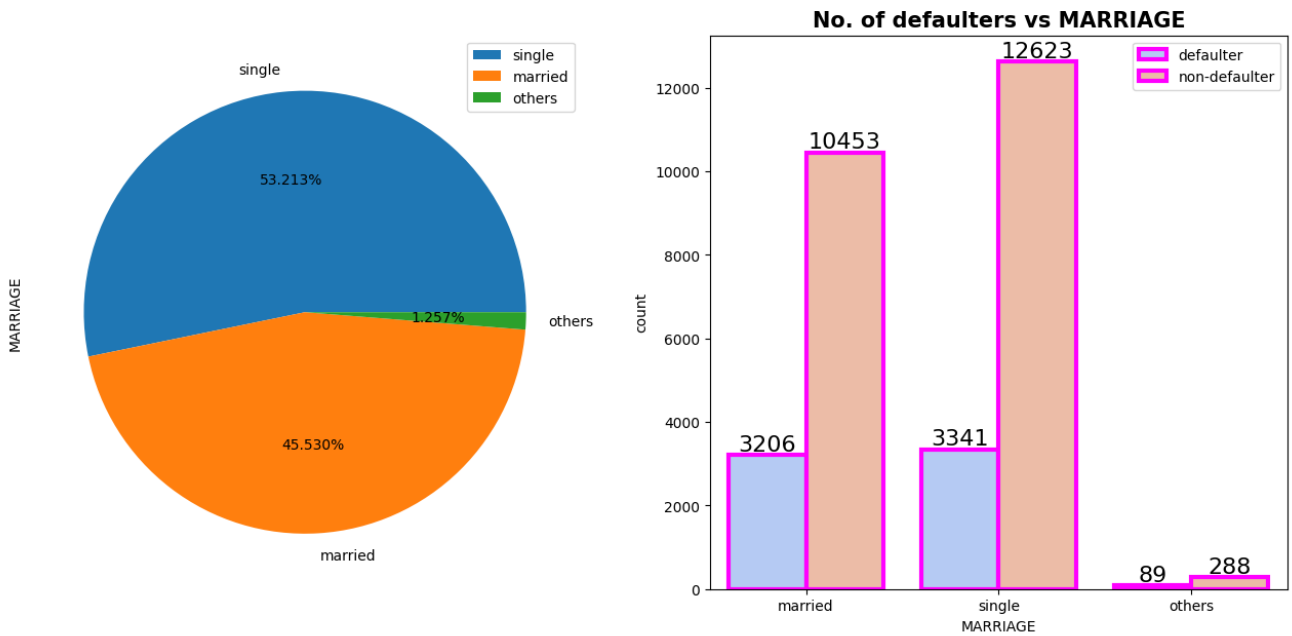
<Figure size 1000x500 with 0 Axes>



<Figure size 1000x500 with 0 Axes>



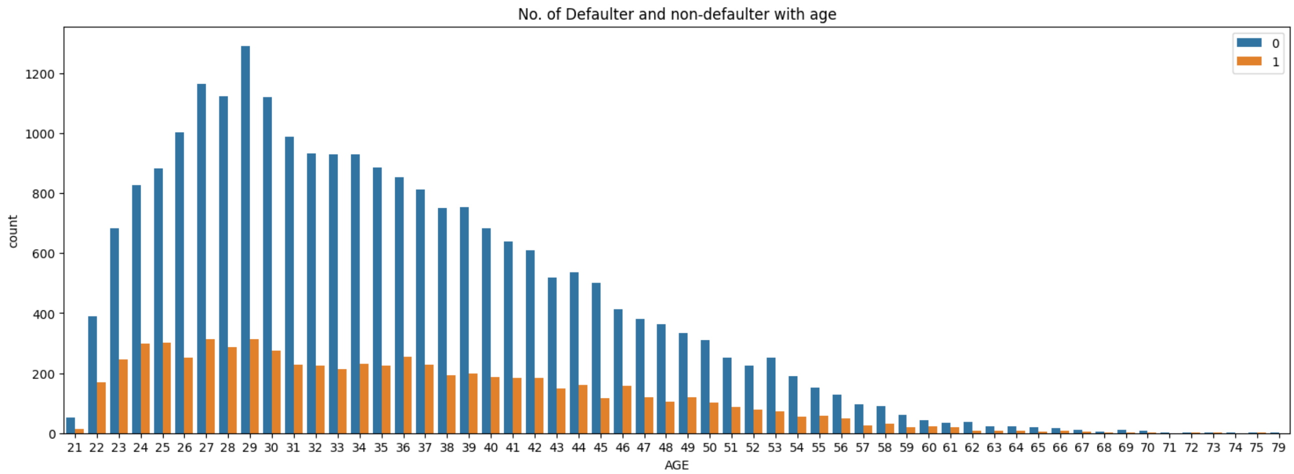
<Figure size 1000x500 with 0 Axes>



From above graphs we can see draw following insights:

- There are more females credit card holders, and therefore there are more female defaulters.
- We can clearly see that single people opt for credit cards more than married people.
- We can clearly see that higher educated people tend to opt for credit cards more than other people.

```
# Checking the relationship between age and our dependent variable.
plt.figure(figsize=(18,6))
ax = sns.countplot(x = 'AGE', hue = 'is_defaulter', data =df, lw=2)
ax.legend(loc='upper right')
plt.title('No. of Defaulter and non-defaulter with age')
plt.show()
```



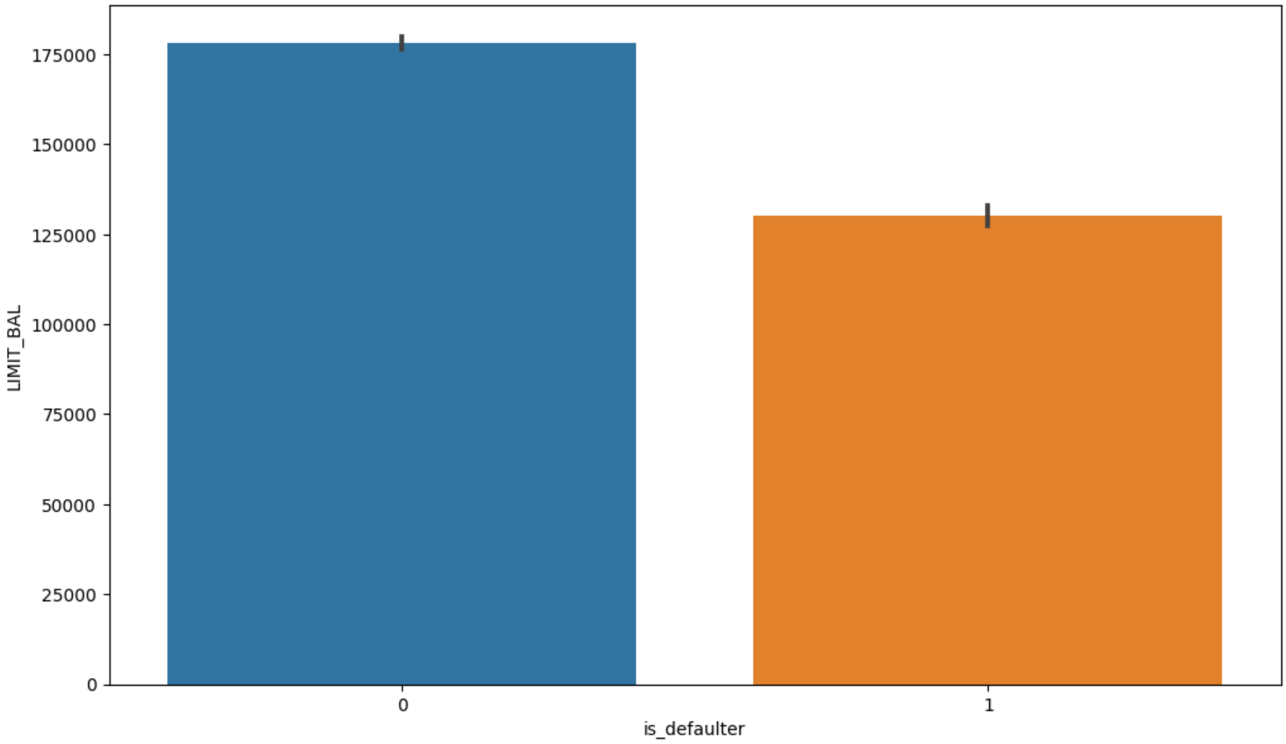

```
# Now lets explore LIMIT_BAL column which contains the credit limit data of our clients.
df['LIMIT_BAL'].describe()
```

```
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
```

```
plt.figure(figsize=(12,7))
sns.barplot(x='is_defaulter', y='LIMIT_BAL', data=df)
```

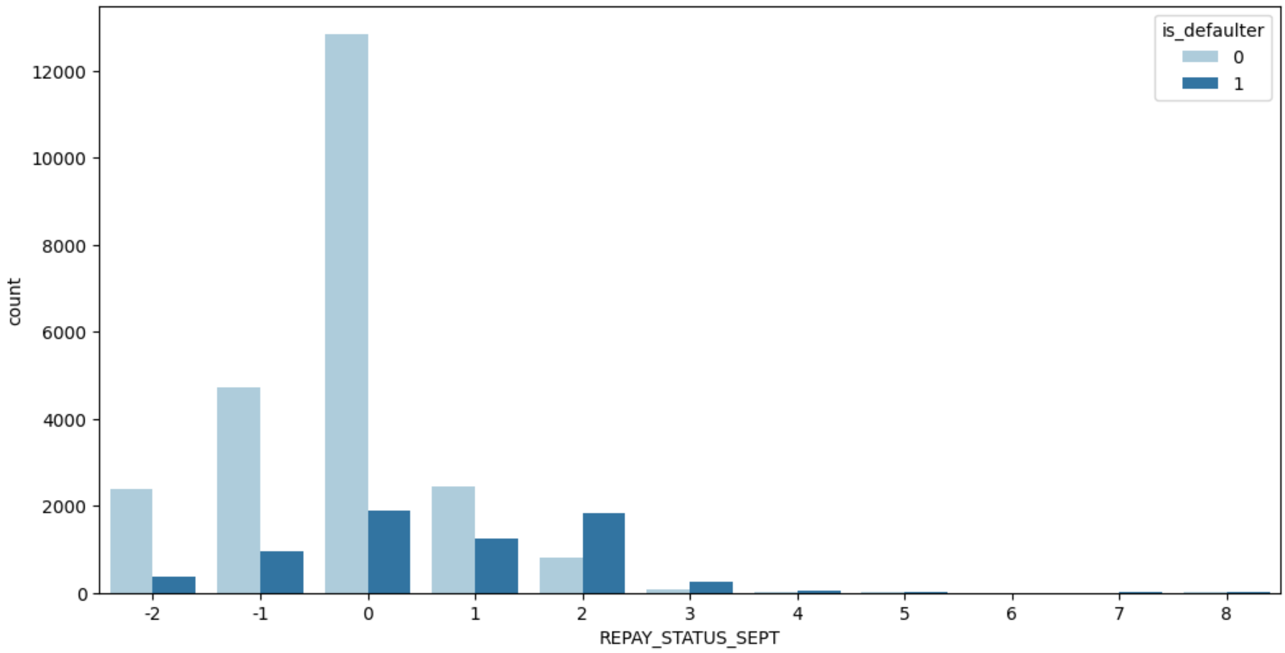
```
<Axes: xlabel='is_defaulter', ylabel='LIMIT_BAL'>
```

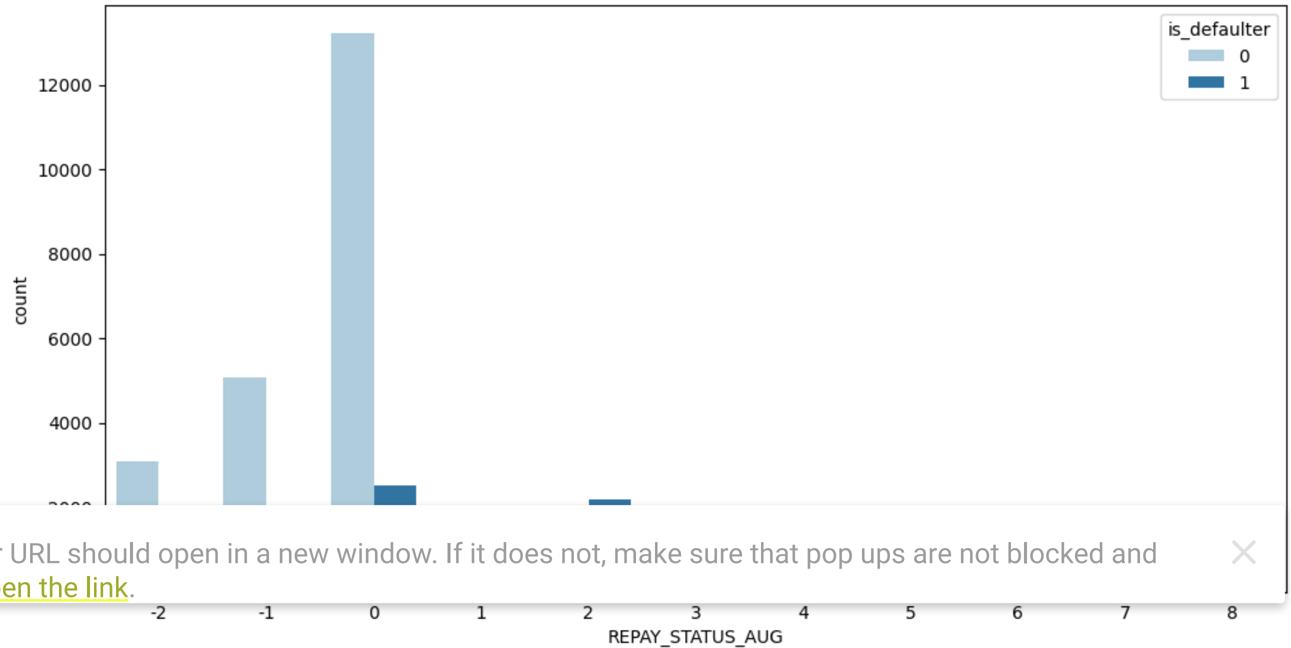


Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

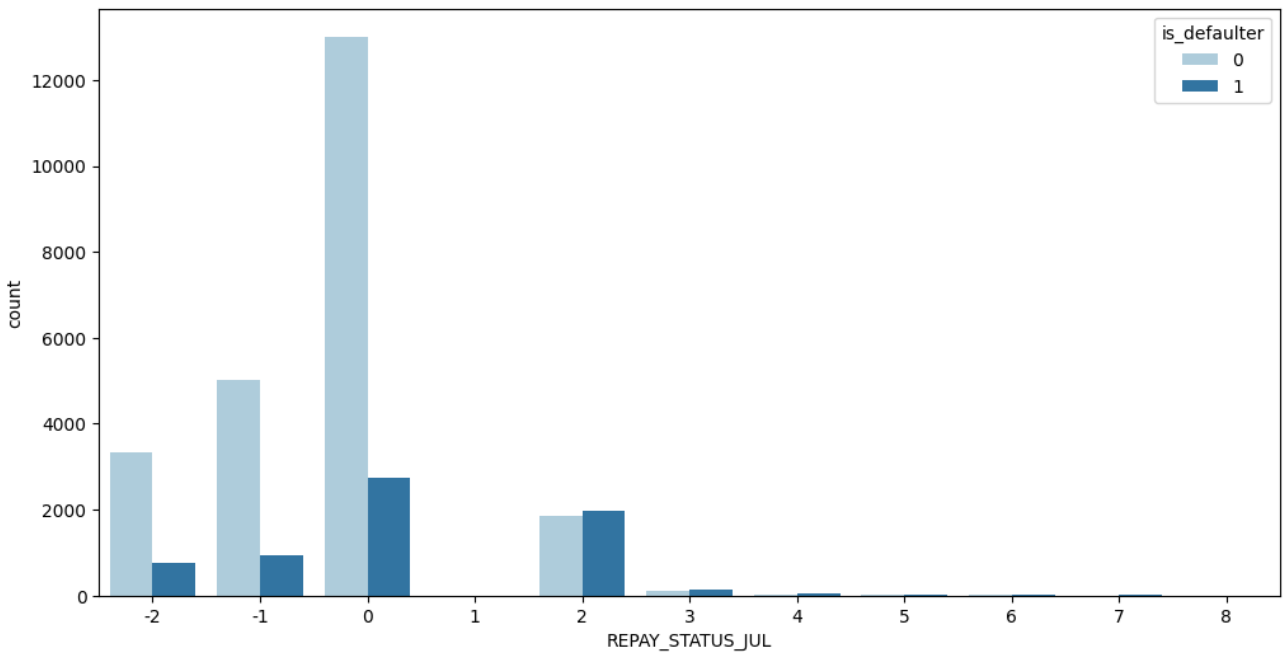
Payment Status History

```
# Looking at the repayment columns for each month.
repayment_feature_list = ['REPAY_STATUS_SEPT',  'REPAY_STATUS_AUG', 'REPAY_STATUS_JUL', 'R

# Plotting graph for each payment feature.
for pay_column in repayment_feature_list:
  plt.figure(figsize=(12,6))
  sns.countplot(x = pay_column, hue = 'is_defaulter', data = df ,palette = 'Paired')
```





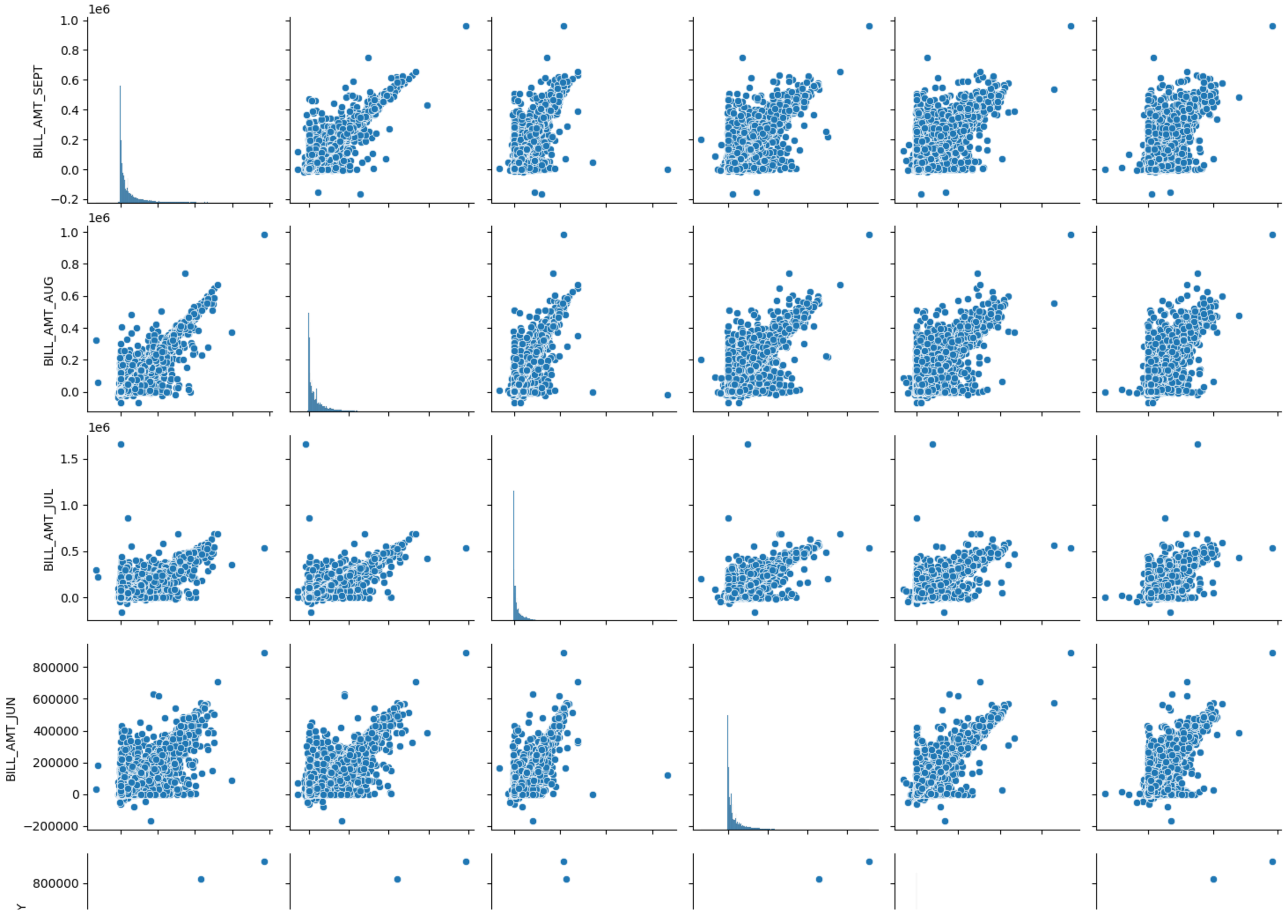
Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.



From above graph it is clear that most often when there is a delay in payment, there is a delay of 2 months. Also we can see that most of our users have revolving credit(value 0) which is defined as credit that is automatically renewed as debts are paid off.

```
# Lets now check the bill amount features.
# Assigning the bill amount features to a single variable

df_bill_amount = df[['BILL_AMT_SEPT', 'BILL_AMT_AUG', 'BILL_AMT_JUL', 'BILL_AMT_JUN', 'BIL
sns.pairplot(data = df_bill_amount)
```

```
<seaborn.axisgrid.PairGrid at 0x7ffa35d13af0>
```



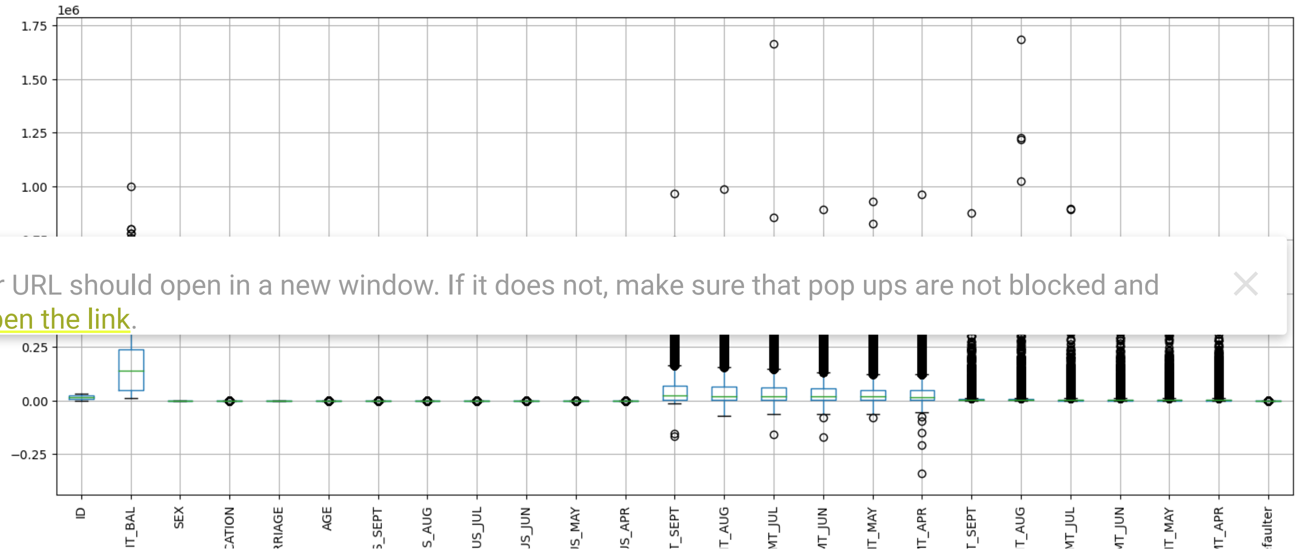
Detecting outliers in our dataframe

```
# Draw box plot to see if there is any outliers in our dataset
plt.figure (figsize= (18,7))
```

Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

```
# rotating xticks to 90 degrees. this is done when we want our x-axis label annotators to
# because there may not be enough space for us to visualize them.
```

```
(array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24, 25]),
 [Text(1, 0, 'ID'),
  Text(2, 0, 'LIMIT_BAL'),
  Text(3, 0, 'SEX'),
  Text(4, 0, 'EDUCATION'),
  Text(5, 0, 'MARRIAGE'),
  Text(6, 0, 'AGE'),
  Text(7, 0, 'REPAY_STATUS_SEPT'),
  Text(8, 0, 'REPAY_STATUS_AUG'),
  Text(9, 0, 'REPAY_STATUS_JUL'),
  Text(10, 0, 'REPAY_STATUS_JUN'),
  Text(11, 0, 'REPAY_STATUS_MAY'),
  Text(12, 0, 'REPAY_STATUS_APR'),
  Text(13, 0, 'BILL_AMT_SEPT'),
  Text(14, 0, 'BILL_AMT_AUG'),
  Text(15, 0, 'BILL_AMT_JUL'),
  Text(16, 0, 'BILL_AMT_JUN'),
  Text(17, 0, 'BILL_AMT_MAY'),
  Text(18, 0, 'BILL_AMT_APR'),
  Text(19, 0, 'PRE_PAY_AMT_SEPT'),
  Text(20, 0, 'PRE_PAY_AMT_AUG'),
  Text(21, 0, 'PRE_PAY_AMT_JUL'),
  Text(22, 0, 'PRE_PAY_AMT_JUN'),
  Text(23, 0, 'PRE_PAY_AMT_MAY'),
  Text(24, 0, 'PRE_PAY_AMT_APR'),
  Text(25, 0, 'is_defaulter')])
```



From the above boxplot, we can see that there are quite a few outliers present in our features. And most of these outliers are present in features containing Pre-payment and Bill amount data.

```
# creating a list columns in which outliers are present.
outlier_columns = ['LIMIT_BAL', 'BILL_AMT_SEPT','BILL_AMT_AUG', 'BILL_AMT_JUL', 'BILL_AMT_
                 'BILL_AMT_APR', 'PRE_PAY_AMT_SEPT', 'PRE_PAY_AMT_AUG', 'PRE_PAY_AMT_JUL',
                 'PRE_PAY_AMT_APR']
# using IQR method for dropping outliers from above columns
Q1 = df[outlier_columns].quantile(0.25)
Q3 = df[outlier_columns].quantile(0.75)

IQR = Q3 - Q1                   # interquartile range
```

```
# using interquartile range to find and remove outliers from our dataframe.
df = df[~((df[outlier_columns] < (Q1 - 1.5 * IQR)) |(df[outlier_columns] > (Q3 + 1.5 * IQR
```

```
# checking the new shape of the data.
df.shape
```

```
(19731, 25)
```

```
# Dropping some of the unnecessary columns.
df.drop(['ID'], axis=1,inplace =True)
```

```
df.shape
```

```
(19731, 24)
```

Feature Engineering

```
# Now checking for correlation among our dependent variables (Multicollinearity) using VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
```

Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

```
# performing VIF analysis
calc_vif(df[[i for i in df.describe().columns if i not in ['is_defaulter']]])
```

	variables	VIF
0	LIMIT_BAL	3.206036
1	SEX	9.186694
2	EDUCATION	7.376490
3	MARRIAGE	6.309368
4	AGE	10.795935
5	REPAY_STATUS_SEPT	1.862369
6	REPAY_STATUS_AUG	3.506609
7	REPAY_STATUS_JUL	4.636375
8	REPAY_STATUS_JUN	5.607305
9	REPAY_STATUS_MAY	6.362055
10	REPAY_STATUS_APR	4.304172
11	BILL_AMT_SEPT	25.508539
12	BILL_AMT_AUG	48.010351

As we can see from above, that some of our features have high multicollinearity in them particularly the bill amount columns. so we need to do some feature engineering on them.

15	BILL_AMT_MAY	61.187104

```
# Lets add up all bill amount features together in one.
df['TOTAL_BILL_PAY'] = df['BILL_AMT_SEPT'] + df['BILL_AMT_AUG'] + df['BILL_AMT_JUL'] + df[
```

17	PRE_PAY_AMT_SEPT	3.798648

Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

',

	variables	VIF
0	LIMIT_BAL	3.188650
1	SEX	9.170535
2	EDUCATION	7.355176
3	MARRIAGE	6.302731
4	AGE	10.790621
5	REPAY_STATUS_SEPT	1.861136

Label and One Hot encoding

8	REPAY_STATUS_JUN	5.588313

```
df['SEX']
```

```
0        2
1        2
2        2
3        2
5        1
        ..
29991    1
29992    1
29994    1
29996    1
29999    1
Name: SEX, Length: 19731, dtype: int64
```

17 TOTAL_BILL_PAY 3.713747

Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

```
# One hot encoding.
df = pd.get_dummies(df,columns=['EDUCATION','MARRIAGE'])
```

```
df.head()
```

	LIMIT_BAL	SEX	AGE	REPAY_STATUS_SEPT	REPAY_STATUS_AUG	REPAY_STATUS_JUL	REPAY_
0	20000	0	24	2	2	-1	
1	120000	0	26	-1	2	0	
2	90000	0	34	0	0	0	
3	50000	0	37	0	0	0	
5	50000	1	37	0	0	0	

5 rows × 34 columns

```
# final data shape
df.shape
```

```
    (19731, 34)
```

```
# Creating dependent variable and independent variable
independent_variables = df.drop(['is_defaulter'],axis=1)
dependent_variable = df['is_defaulter']
```

```
# scaling the data using zscore.
from scipy.stats import zscore
x = round(independent_variables.apply(zscore),3)
y = dependent_variable
```

```
# train test split
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state =
```

APPLYING SMOTE (Synthetic Minority Oversampling Technique)

Since we have an imbalanced dataset, we are going to need to apply some technique to remedy this. So we will try oversampling technique called SMOTE.

```
# applying oversampling to overcome class imbalance
from imblearn.over_sampling import SMOTE
smote= SMOTE()
```



```
print('Original dataset shape', Counter(y_train))
print('Resample dataset shape', Counter(y_train_smote))
Counter(y_train_smote)
```

```
    Original dataset shape Counter({0: 11694, 1: 4090})
    Resample dataset shape Counter({1: 14626, 0: 14626})
    Counter({1: 14626, 0: 14626})
```

MODEL IMPLEMENTATION

```
# importing all the evaluation metrics that we will need for comparison.
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
from sklearn.metrics import roc_auc_score, confusion_matrix, roc_curve, auc, classificatio
```

1. LOGISTIC REGRESSION

```
# Importing Logistics Regression and GridSearchCV

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


# initiate the model.
logistic_model = LogisticRegression(class_weight='balanced')

# define the parameter grid.
param_grid = {'penalty':['l1','l2'], 'C' : [0.0001,0.001,0.003,0.004,0.005, 0.01, 0.1, 0.2

# implementing the model.
logistic_model= GridSearchCV(logistic_model, param_grid, scoring = 'accuracy', n_jobs = -1
logistic_model.fit(x_train_smote, y_train_smote)
```

```
Fitting 3 folds for each of 40 candidates, totalling 120 fits
```

```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

```
# getting the best estimator
logistic_model.best_estimator_
```

```
LogisticRegression
LogisticRegression(C=0.005, class_weight='balanced')
```



```
{'C': 0.005, 'penalty': 'l2'}
```

```
# getting the predicted probability of target variable.
y_train_preds_logistic = logistic_model.predict_proba(x_train_smote)[:,1]
y_test_preds_logistic = logistic_model.predict_proba(x_test)[:,1]


# getting the predicted class
y_train_class_preds_logistic = logistic_model.predict(x_train_smote)
y_test_class_preds_logistic = logistic_model.predict(x_test)


# checking the accuracy on training and unseen test data.
logistic_train_accuracy= accuracy_score(y_train_smote, y_train_class_preds_logistic)
logistic_test_accuracy= accuracy_score(y_test, y_test_class_preds_logistic)

print("The accuracy on train data is ", logistic_train_accuracy)
print("The accuracy on test data is ", logistic_test_accuracy)
```

```
The accuracy on train data is  0.6819020921646383
The accuracy on test data is  0.6820369901190778
```

```
# writing a function for evaluating various metrics
def evaluation_metrics(actual, predicted):

  """ This function is used to find the accuracy score , precision score , recall score ,
      Confusion Matrix , Classification  report """
  metrics_list = []
  accuracy = accuracy_score(actual,predicted)
  precision = precision_score(actual, predicted)
  recall = recall_score(actual, predicted)
  model_f1_score = f1_score(actual, predicted)
  auc_roc_score = roc_auc_score(actual , predicted)
  model_confusion_matrix = confusion_matrix(actual , predicted)

  metrics_list = [accuracy,precision,recall,model_f1_score,auc_roc_score, model_confusion_
  return metrics_list
```

```
evaluation_metrics(y_test, y_test_class_preds_logistic)
```

```
[0.6820369901190778,
 0.421259842519685,
 0.632512315270936,
 0.5057109098070106,
 0.6658468806914024,
 array([[2050,  882],
        [ 373,  642]])]
```

```
# Let's store these metrics in a dataframe. that way we can easily compare metrics of diff
n_
metric_values = evaluation_metrics(y_test, y_test_class_preds_logistic)

# zipping together above lists to form a dictionary
metric_dict = dict(zip(metric_name_list,metric_values))

# creating a dataframe out of this.
evaluation_metric_df = pd.DataFrame.from_dict(metric_dict, orient='index').reset_index()
evaluation_metric_df.columns = ['Evaluation Metric','Logistic Regression']
```

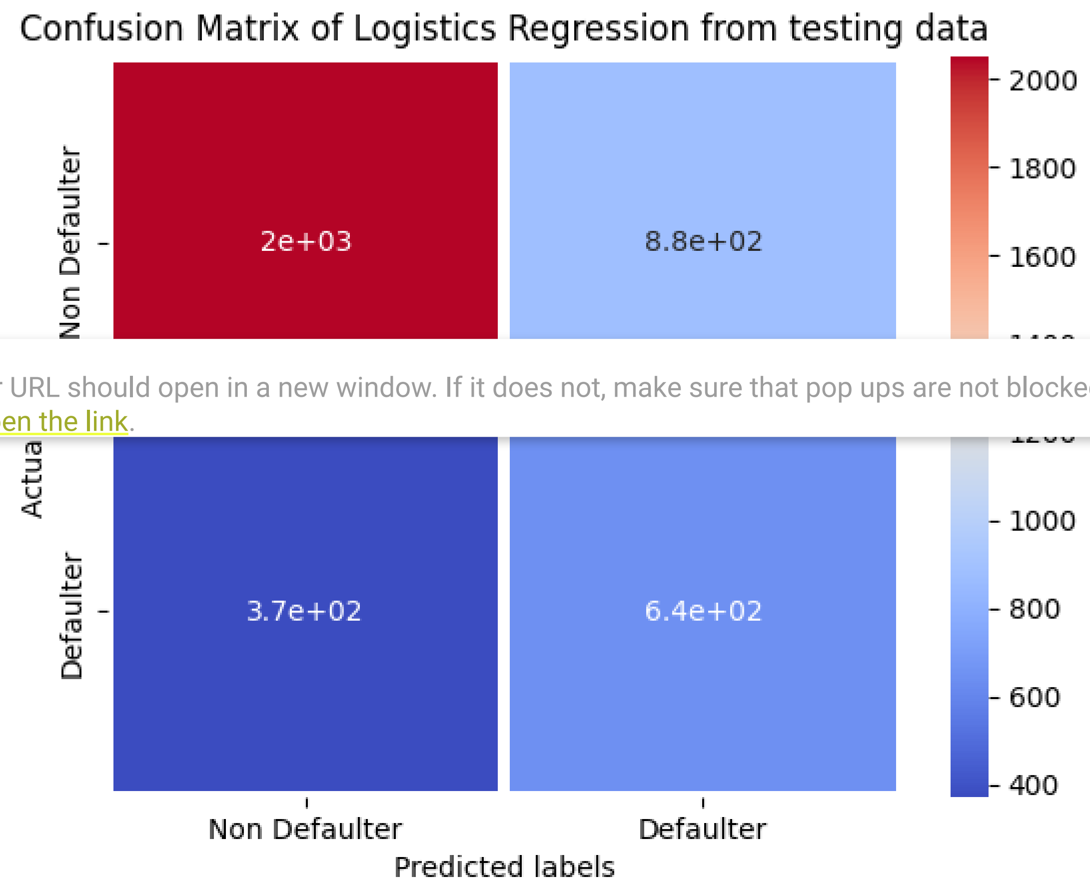


```
evaluation_metric_df
```

	Evaluation Metric	Logistic Regression
0	accuracy	0.682037

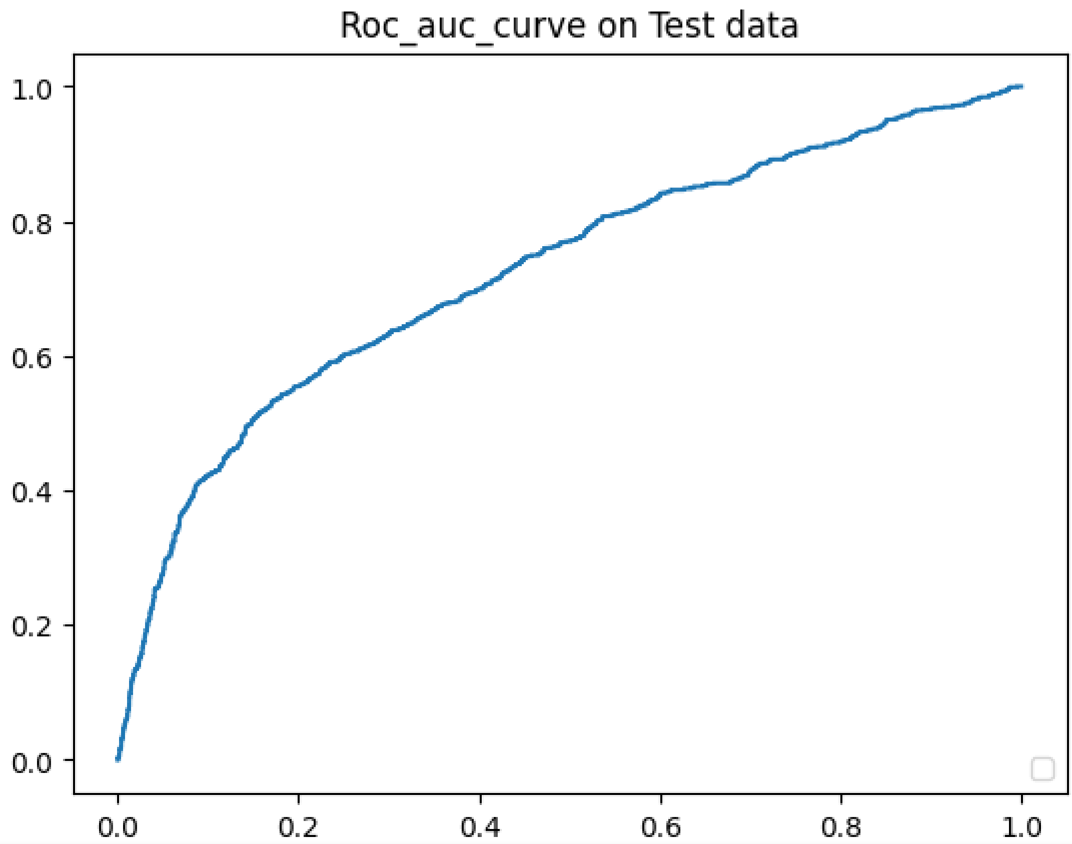
```
# Plotting the confusion matrix from test data

labels = ['Non Defaulter', 'Defaulter']
cm = confusion_matrix(y_test,y_test_class_preds_logistic)
ax= plt.subplot()
sns.heatmap(cm, annot=True, cmap='coolwarm', ax = ax, lw = 3) #annot=True to annotate cell

# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('Actual labels')
ax.set_title('Confusion Matrix of Logistics Regression from testing data')
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)

# also printing confusion matrix values
print(cm)
```

```
[[2050  882]
 [ 373  642]]
```



```
# Plotting Roc_auc_curve for test data
y_test_pred_logistic = logistic_model.predict_proba(x_test)[:,1]
fpr, tpr, _ = roc_curve(y_test,y_test_pred_logistic)
plt.plot(fpr,tpr)
plt.title("Roc_auc_curve on Test data")
plt.legend(loc=4)
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that a
```



Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

```
# printing the classification report.
print('classification_report is \n {}'.format(classification_report(y_test, y_test_class_p
```

```
classification_report is
               precision    recall  f1-score   support

           0       0.85      0.70      0.77      2932
           1       0.42      0.63      0.51      1015

    accuracy                           0.68      3947
   macro avg       0.63      0.67      0.64      3947
weighted avg       0.74      0.68      0.70      3947
```

```
evaluation_metric_df
```

	Evaluation Metric	Logistic Regression
0	accuracy	0.682037
1	precision	0.42126
2	recall	0.632512
3	f1_score	0.505711

Conclusion:

- We have implemented logistic regression and we are getting accuracy_score is approx 68%
- Precision score is around 41% and f1_score is around 50%
- roc_auc approx is 67% and recall_score is approx 64%

2. Random Forest Classifier

```
# Importing Random forest
from sklearn.ensemble import RandomForestClassifier
```

```
model_rf= RandomForestClassifier()

grid_values = {'n_estimators':[50,80,90,100], 'max_depth':[9,11,14]}             # initia
grid_rf = GridSearchCV(model_rf, param_grid = grid_values, scoring = 'accuracy', cv=3)
```

Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
# getting the best estimator
grid_rf.best_estimator_
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=14)
```

```
# getting the best parameter
grid_rf.best_params_
```

```
{'max_depth': 14, 'n_estimators': 100}
```

```
# Getting the predicted classes
```

```
# Getting the predicted classes
y_train_class_preds_rf = grid_rf.predict(x_train_smote)
y_test_class_preds_rf = grid_rf.predict(x_test)


# Getting the evaluation metrics using our function and adding it to evaluation dataframe
evaluation_metric_df['Random Forest']=evaluation_metrics(y_test,y_test_class_preds_rf)
evaluation_metric_df
```

	Evaluation Metric	Logistic Regression	Random Forest
0	accuracy	0.682037	0.876869
1	precision	0.42126	0.771282
2	recall	0.632512	0.740887
3	f1_score	0.505711	0.755779
4	roc_auc_score	0.665847	0.832415
5	confusion_matrix	[[2050, 882], [373, 642]]	[[2709, 223], [263, 752]]

```
# Plotting the confusion matrix from test data

labels = ['Non Defaulter', 'Defaulter']
cm = confusion_matrix(y_test,y_test_class_preds_rf)
ax= plt.subplot()
sns.heatmap(cm, annot=True, cmap='coolwarm', ax = ax, lw = 3) #annot=True to annotate cell

# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('Actual labels')
```

Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

```
ax.yaxis.set_ticklabels(labels)

# also printing confusion matrix values
print(cm)
```

```
[[2709  223]
 [ 263  752]]
```


```
print('classification_report is \n {}'.format(classification_report(y_test, y_test_class_p
```

```
classification_report is
               precision    recall  f1-score   support

           0       0.91      0.92      0.92      2932
           1       0.77      0.74      0.76      1015

    accuracy                           0.88      3947
   macro avg       0.84      0.83      0.84      3947
weighted avg       0.88      0.88      0.88      3947
```

```
# Printing Roc_auc_curve from test data

y_test_preds_proba_rf = grid_rf.predict_proba(x_test)[::,1]
fpr, tpr, _ = roc_curve(y_test,  y_test_preds_proba_rf)
auc = roc_auc_score(y_test,  y_test_preds_proba_rf)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.title("Roc_auc_curve on testing data")
plt.legend(loc=4)
```

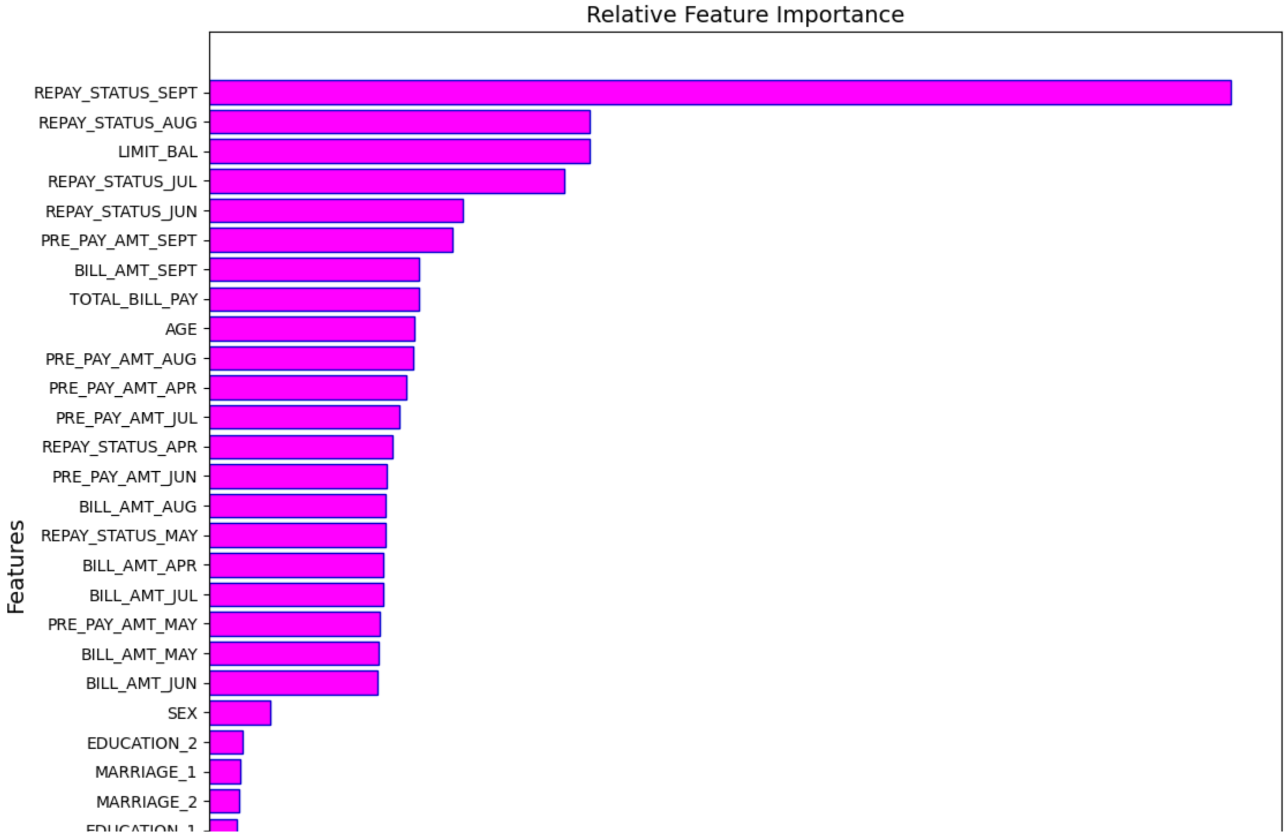
Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

Roc_auc_curve on testing data

Random Forest model has inbuilt support for showing the feature importances - i.e. which feature is more important in coming up with the predicted results. This helps us interpret and understand the model better.

```
# getting columns names from training data
features = x_train_smote.columns

# getting the feature importances
importances = grid_rf.best_estimator_.feature_importances_
indices = np.argsort(importances)
```

```
# plotting the feature importances using a horizontal bar graph.
plt.figure (figsize= (12,12))
plt.title('Relative Feature Importance', fontsize=14)
plt.barh(range(len(indices)), importances[indices], color='magenta', edgecolor='mediumblue
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.ylabel('Features', fontsize=14)
plt.show()
```



3. K-Nearest Neighbour Classifier

```
# Import K Nearest Neighbour Classifier
from sklearn.neighbors import KNeighborsClassifier

# initializing the model
```



```
# knn the parameter to be tuned is n_neighbors
param_grid = {'n_neighbors':[4,5,6,7,8,10,12,14]}

# Fitting the model

knn_cv= GridSearchCV(knn,param_grid, scoring = 'accuracy',cv=3)
knn_cv.fit(x_train_smote,y_train_smote)
```

```
GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier
```

```
# find best score
knn_cv.best_score_
```

```
0.7788533212022085
```

```
# best parameters
```

```
knn_cv.best_params_
```

```
{'n_neighbors': 4}
```

```
knn_cv.best_estimator_
```

```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=4)
```

```
# Get the predicted classes
y_train_class_preds_knn = knn_cv.predict(x_train_smote)
y_test_class_preds_knn = knn_cv.predict(x_test)
```

```
# getting the evaluation metrics and adding it to metric dataframe.
evaluation_metric_df['KNeighborsClassifier'] = evaluation_metrics(y_test,y_test_class_pred
evaluation_metric_df
```

	Evaluation Metric	Logistic Regression	Random Forest	KNeighborsClassifier
0	accuracy	0.682037	0.876869	0.864454
1	precision	0.42126	0.771282	0.688679
2	recall	0.632512	0.740887	0.863054
3	f1_score	0.505711	0.755779	0.766069
4	roc_auc_score	0.665847	0.832415	0.863996

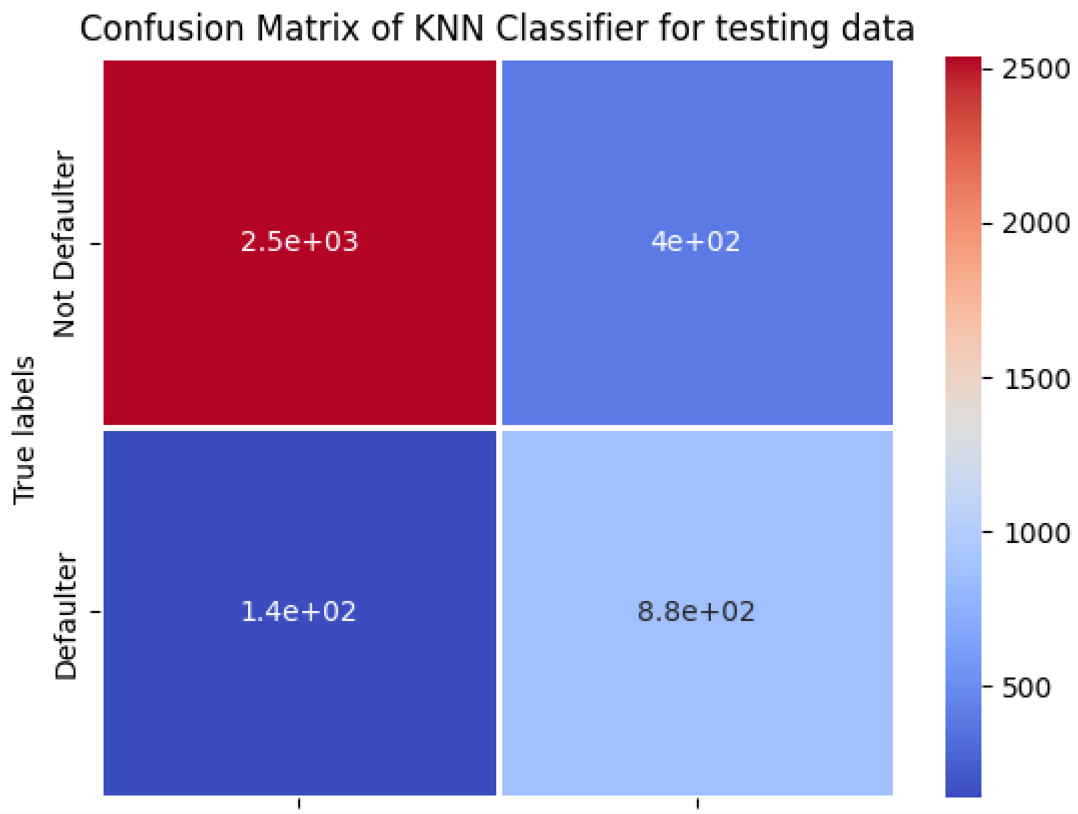

```
# Printing the classification report
print('classification_report is \n {}'.format(classification_report(y_test, y_test_class_p
```

```
classification_report is
               precision    recall  f1-score   support

           0       0.95      0.86      0.90      2932
           1       0.69      0.86      0.77      1015

    accuracy                           0.86      3947
   macro avg       0.82      0.86      0.84      3947
weighted avg       0.88      0.86      0.87      3947
```

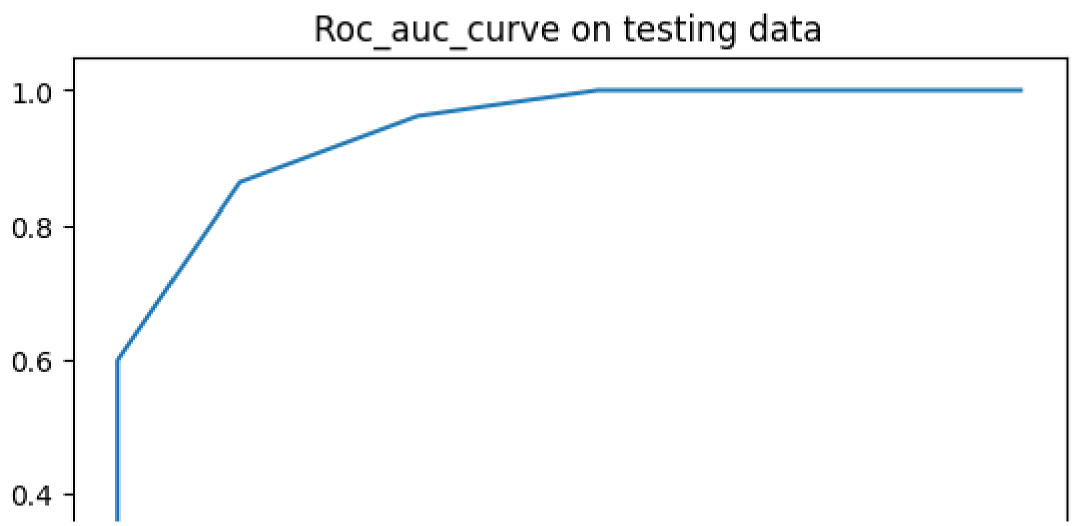
```
# Plotting the confusion matrix for testing data
labels = ['Not Defaulter', 'Defaulter']
cm = confusion_matrix(y_test,y_test_class_preds_knn)
print(cm)

ax= plt.subplot()
sns.heatmap(cm, annot=True, linewidths=1, cmap='coolwarm',ax = ax)
```

```
# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix of KNN Classifier for testing data')
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)
```

```
[[2536  396]
 [ 139  876]]
[Text(0, 0.5, 'Not Defaulter'), Text(0, 1.5, 'Defaulter')]
```




```
# Printing Roc_auc_curve from test data

y_test_preds_proba_knn = knn_cv.predict_proba(x_test)[::,1]
fpr, tpr, _ = roc_curve(y_test,  y_test_preds_proba_knn)
auc = roc_auc_score(y_test,  y_test_preds_proba_rf)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.title("Roc_auc_curve on testing data")
plt.legend(loc=4)
plt.show()
```



4. Support Vector Classifier

```
# Importing support vector machine algorithm from sklearn
from sklearn import svm

# initiate a svm Classifier
svm_model = svm.SVC(kernel = 'poly',gamma='scale', probability=True)

# fit the model using the training sets
svm_model.fit(x_train_smote, y_train_smote)
```

```
SVC
SVC(kernel='poly', probability=True)
```

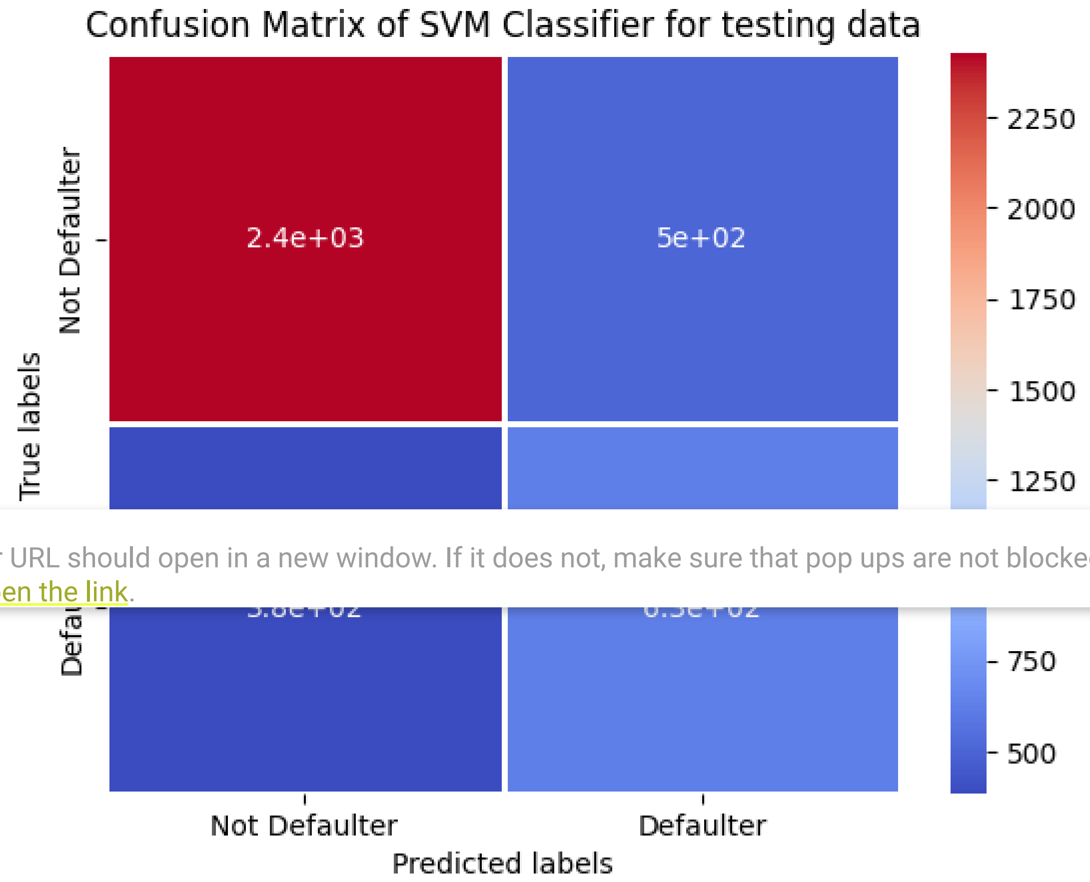
Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

```
y_train_class_preds_svm = svm_model.predict(x_train_smote)
y_test_class_preds_svm = svm_model.predict(x_test)


evaluation_metric_df['Support Vector classifier'] = evaluation_metrics(y_test,y_test_class_
evaluation_metric_df
```

	Evaluation Metric	Logistic Regression	Random Forest	KNeighborsClassifier	Support Vector classifier
0	accuracy	0.682037	0.876869	0.864454	0.774512
1	precision	0.42126	0.771282	0.688679	0.555066
2	recall	0.632512	0.740887	0.863054	0.62069
3	f1_score	0.505711	0.755779	0.766069	0.586047
4	roc_auc_score	0.665847	0.832415	0.863996	0.724226
5	confusion_matrix	[[2050, 882],	[[2709, 223],	[[2536, 396], [139, 876]]	[[2427, 505],

```
# Plotting the confusion matrix for testing data
labels = ['Not Defaulter', 'Defaulter']
cm = confusion_matrix(y_test,y_test_class_preds_svm)
print(cm)

ax= plt.subplot()
sns.heatmap(cm, annot=True, linewidths=1, cmap='coolwarm',ax = ax)

# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix of SVM Classifier for testing data')
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)
```

```
[[2427  505]
 [ 385  630]]
[Text(0, 0.5, 'Not Defaulter'), Text(0, 1.5, 'Defaulter')]
```



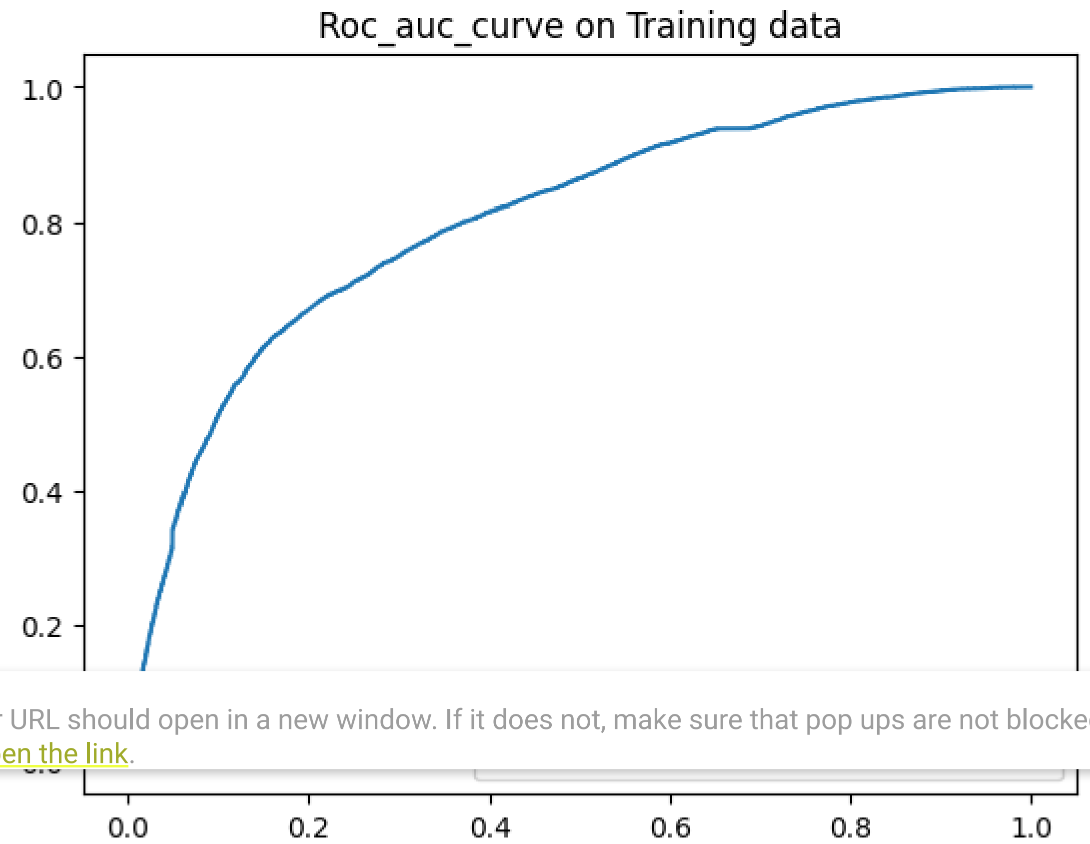
Your URL should open in a new window. If it does not, make sure that pop ups are not blocked and reopen the link.

```
# Printing the classification report.
print('classification_report is \n {}'.format(classification_report(y_test, y_test_class_p
```

```
classification_report is
               precision    recall  f1-score   support

           0       0.95      0.86      0.90      2932
           1       0.69      0.86      0.77      1015

    accuracy                           0.86      3947
   macro avg       0.82      0.86      0.84      3947
```

```
weighted avg       0.88      0.86      0.87      3947
```

```
# Roc_auc_curve on taining data

y_train_preds_proba_svm = svm_model.predict_proba(x_train_smote)[::,1]
fpr, tpr, _ = roc_curve(y_train_smote,  y_train_preds_proba_svm )
auc = roc_auc_score(y_train_smote,  y_train_preds_proba_svm )
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.title("Roc_auc_curve on Training data")
plt.legend(loc=4)
plt.show()
```




```
# finally, we can compare our models on variour evaluation metric values.
evaluation_metric_df
```

	Evaluation Metric	Logistic Regression	Random Forest	KNeighborsClassifier	Support Vector classifier
0	accuracy	0.682037	0.876869	0.864454	0.774512
1	precision	0.42126	0.771282	0.688679	0.555066
2	recall	0.632512	0.740887	0.863054	0.62069
3	f1_score	0.505711	0.755779	0.766069	0.586047
4	roc_auc_score	0.665847	0.832415	0.863996	0.724226
5	confusion_matrix	[[2050, 882],	[[2709, 223],	[[2536, 396], [139, 876]]	[[2427, 505],

Conclusions Drawn:

After conducting this thorough exercise, we found that :

- Most of the credit card users are Female and have higher number of defaults.
- Most of the credit card users are highly educated.
- Single users have more no. of credit cards.
- The number of credit card users goes down with increase in age as old people have less consumption and may not be able to use credit cards and their purchases are usually made by younger family members.
- Using a Logistic Regression classifier, we can predict an approximate accuracy of 67.7% and ROC_AUC score of 0.663
- Using Random Forest Classifier, we can predict an accuracy of around 87.6% and ROC_AUC score of 0.837
- Using K-Neighbor Classifier, we can predict an accuracy of 86% and ROC_AUC score of 0.865
- Using Support Vector Machine Classifier, we can predict an accuracy of 76.7% and ROC_AUC score of around 0.72
- Random Forest Classifier and K Neighbors classifier perform the best among all models.

Our best models are Random Forest and K-Neighbor Classifier as they have the best Precision, Recall, ROC_AUC and F1 score values. This being an imbalanced dataset, **Recall will be most important metric as we don't want to classify a defaulter as a non defaulter so that makes K Neighbor Classifier model more suitable for the task**.