INTRODUCTION TO DATA SCIENCE

“Credit Card Default Prediction”

Project By

Namit Kapoor - 21UCS135

Course Instructors

Dr. Subrat Dash

Dr. Lal Upendra Pratap Singh

Dr. Aloke Dutta

Department of Computer Science Engineering

The LNM Institute of Information Technology

LNMIIT
The LNM Institute of Information Technology

TABLE OF CONTENTS

1. Problem Statement

In recent years, credit card issuers in Taiwan have faced the cash and credit card debt crisis, and delinquency is expected to peak in the third quarter of 2006 (Chou, 2006). In order to increase market share, card-issuing banks in Taiwan over-issued cash and credit cards to unqualified applicants. At the same time, most cardholders, irrespective of their repayment ability, overused credit cards for consumption and accumulated heavy credit and cash–card debts. The crisis caused a blow to consumer finance confidence and is a big challenge for banks and cardholders. The project is aimed at predicting the default of customers in Taiwan

1.1 Objective

The primary objective of this data science project is to develop a predictive model that can accurately forecast the likelihood of credit card default among customers in Taiwan. By leveraging historical data on customer behavior, financial transactions, and credit profiles, the aim is to create a robust predictive tool that assists card-issuing banks in identifying high-risk customers. This predictive model will contribute to risk mitigation efforts, enabling timely intervention and targeted strategies to reduce default rates

1.2 Tools and Frameworks Used

- **Libraries used in EDA & Machine Learning:**

1. Pandas
2. Numpy
3. Matplotib
4. Seaborn
5. Sklearn
6. Scipy

- **Graphs used for representation:**

1. Bar plot
2. Box Plot
3. Grouped bar plot
4. Heatmap

- **ML Models used for training & testing:**

1. Logistic Regression.
2. KNN.
3. Random Forest.
4. Support Vector Classifier

2. Description of the Dataset

- The dataset provides insights into credit card holders during a financial crisis, aiming to predict customer defaults. Exploratory Data Analysis (EDA) uncovers key trends across demographics. Females show slightly higher default rates, while higher credit limits correlate with reduced defaults. Education, marital status, and age exhibit distinct connections with default behaviors. These findings inform predictive models, offering vital insights into factors influencing credit defaults among diverse customer groups.

- This dataset contains information on default payments, demographic factors, credit data, history of payment, and bill statements of credit card clients in Taiwan from April 2005 to September 2005.

  There are 25 features in the dataset:
  - **ID**: ID of each client
  - **LIMIT_BAL**: Amount of given credit in NT dollars (includes individual and family/supplementary credit)
  - **SEX**: Gender (1=male, 2=female)
  - **EDUCATION**: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)
  - **MARRIAGE**: Marital status (1=married, 2=single, 3=others)
  - **AGE**: Age in years
  - **PAY_0**: Repayment status in September, 2005 (-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)
  - **PAY_2**: Repayment status in August, 2005 (scale same as above)
  - **PAY_3**: Repayment status in July, 2005 (scale same as above)
  - **PAY_4**: Repayment status in June, 2005 (scale same as above)
  - **PAY_5**: Repayment status in May, 2005 (scale same as above)
  - **PAY_6**: Repayment status in April, 2005 (scale same as above)
  - **BILL_AMT1**: Amount of bill statement in September, 2005 (NT dollar)
  - **BILL_AMT2**: Amount of bill statement in August, 2005 (NT dollar)

- **BILL_AMT3**: Amount of bill statement in July, 2005 (NT dollar)
- **BILL_AMT4**: Amount of bill statement in June, 2005 (NT dollar)
- **BILL_AMT5**: Amount of bill statement in May, 2005 (NT dollar)
- **BILL_AMT6**: Amount of bill statement in April, 2005 (NT dollar)
- **PAY_AMT1**: Amount of previous payment in September, 2005 (NT dollar)
- **PAY_AMT2**: Amount of previous payment in August, 2005 (NT dollar)
- **PAY_AMT3**: Amount of previous payment in July, 2005 (NT dollar)
- **PAY_AMT4**: Amount of previous payment in June, 2005 (NT dollar)
- **PAY_AMT5**: Amount of previous payment in May, 2005 (NT dollar)
- **PAY_AMT6**: Amount of previous payment in April, 2005 (NT dollar)
- **Default**: Default payment (1=yes, 0=no)

Source of our dataset:

https://archive.ics.uci.edu/dataset/350/default+of+credit+card+clients

(All the above information about the dataset is taken from this link as well)

3. Data Analysis

- First, we import all the necessary libraries which will be required in our code.

```
import pandas as pd
import numpy as np

from matplotlib import pyplot as plt
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics
from sklearn.model_selection import train_test_split,cross_val_score



from sklearn.model_selection import RandomizedSearchCV,GridSearchCV

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

- Next, we read the dataset in a variable and output its first 5 rows, using **pandas.read_csv( ).**

```
url= r'/content/default of credit card clients.xls'
dataset= pd.read_excel(url, skiprows=1)
```

- Using the **dataset.info(),** we find the information about our dataset, i.e., how many attributes are there, the datatype of each attribute, and whether is there any null value to it or not

```
# Dataset Info
dataset.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   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  int64
 13  BILL_AMT2  30000 non-null  int64
 14  BILL_AMT3  30000 non-null  int64
 15  BILL_AMT4  30000 non-null  int64
 16  BILL_AMT5  30000 non-null  int64
 17  BILL_AMT6  30000 non-null  int64
 18  PAY_AMT1   30000 non-null  int64
 19  PAY_AMT2   30000 non-null  int64
 20  PAY_AMT3   30000 non-null  int64
 21  PAY_AMT4   30000 non-null  int64
 22  PAY_AMT5   30000 non-null  int64
 23  PAY_AMT6   30000 non-null  int64
 24  default    30000 non-null  int64
dtypes: int64(25)
memory usage: 5.7 MB
```

- We use **DataFrame.isnull().sum()** to check if there are any null values in the dataset or not.

```
# Missing Values/Null Values Count
dataset[dataset.isna()].count()
```

```
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        0
dtype: int64
```

- We also use **dataset.describe( ),** which is used to view basic statistical details about our dataset such as min-max values, standard deviation, mean, etc.

```
# Dataset Describe
dataset.describe(include='all')
```

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

8 rows × 25 columns

- Minimizing data duplication through the transformation of numerical columns into categorical ones enables us to visualize the information using a bar graph.

```
#converting all 0,4,5,6 value on education to 4
dataset['EDUCATION']=dataset['EDUCATION'].map({0:4,1:1,2:2,3:3,4:4,5:4,6:4})

#Converting 0 in marriage to 3
dataset['MARRIAGE']=dataset['MARRIAGE'].map({0:3,1:1,2:2,3:3})

#Function to divide limit column into categorical variable for visualization
def limit_cat(x):
  if x >240000:
    return 'Above 240000'
  elif x >140000 :
    return 'Between 140000-240000'
  elif x >10000:
    return 'Between 10000-140000'
  else:
    return

#Function for dividing age column into categorical variable for visualization
def age_cat(x):
  if x >45:
    return 'Above 45'
  elif x > 35:
    return '35-45'
  else:
    return '21-35'

#creating new categorical variable.
dataset['Limit_cat'] = dataset['LIMIT_BAL'].apply(lambda x:limit_cat(x))

#creating new categorical variable.
dataset['Age_cat'] = dataset['AGE'].apply(lambda x: age_cat(x))
```

Created a new limit_cat column by dividing it into

- 10000 to 140000.
- 140000 to 240000.
- Above 240000.

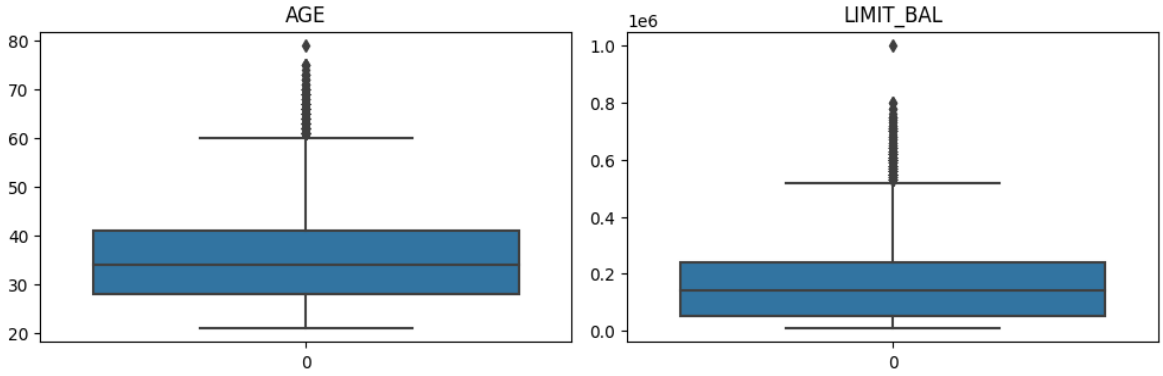
Created a new Age_cat column by dividing it into

- 21 to 35 years.
- 35 to 45 years.
- Above 45 years of age.

- Checking outliers with seaborn boxplot:

```
# checking Outliers with seaborn boxplot

plt.figure(figsize=(10,6))
n=0
for i in ['AGE','LIMIT_BAL']:
  if n<10:
    n=n+1
  plt.subplot(2,2,n)
  sns.boxplot(dataset[i],whis=1.5)
  plt.title(i)
  plt.tight_layout()
```



Ignoring statistical outliers as these values are very well accepted values of age and balance limit.

Inference:

- In our dataset, we can observe the following things:

1. The dataset contains 30000 rows & 25 columns.
2. All 25 columns have numerical values as variables.
3. Columns SEX, EDUCATION, and MARRIAGE contains categorical variable which is:-
   - **SEX**: Gender (1 = male; 2 = female).
   - **EDUCATION**: Education (1 = graduate school; 2 = university; 3 = high school; 0, 4, 5, 6 = others)
   - **MARRIAGE**: Marital status (1 = married; 2 = single; 3 = divorce; 0=others).
4. The dataset has no Null/Duplicate values.
5. **The default** column is the dependent variable while the rest are the independent variable.

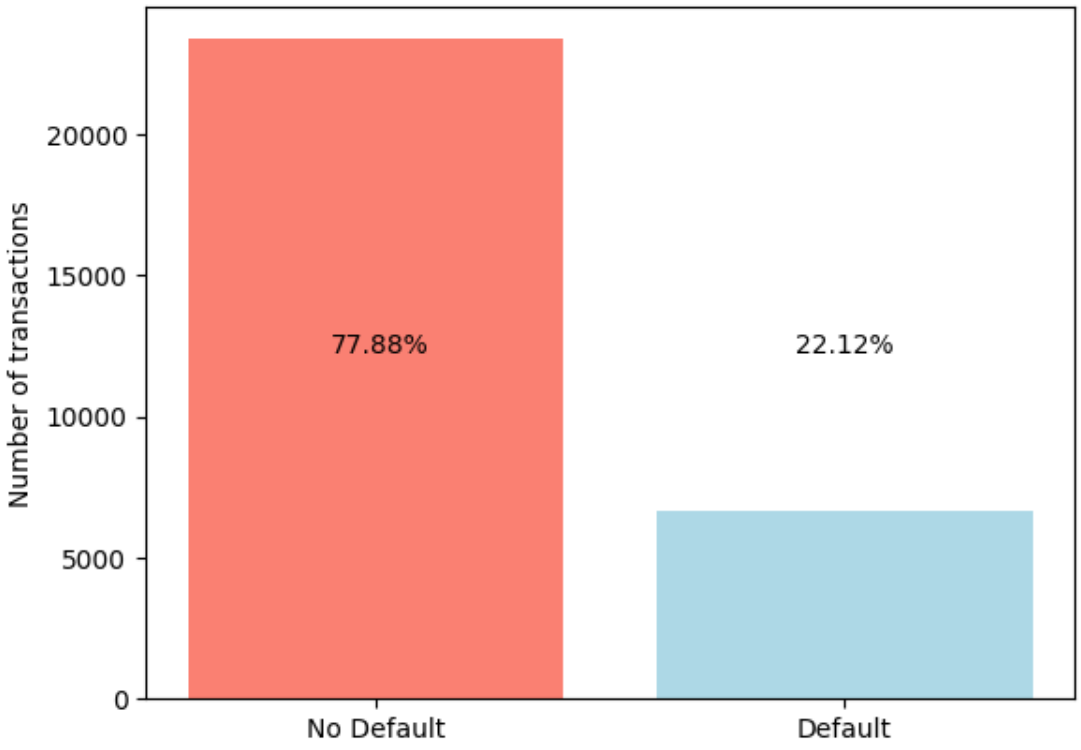
4. Data Visualization and Plots

We chose the Bar graph because it summarises a large set of data in a simple visual form. It displays each category of data in the frequency distribution. It clarifies the trend of data better than the table. It helps in estimating the key values at a glance. As we are comparing rented bike sharing demand with different seasons so bar chart is making it easy to visualize data.

4.1 Barplot of the percentage of defaulters vs non defaulters.

```
classes=dataset['default'].value_counts()
not_default=classes[0]/dataset['default'].count()*100
defaulter=classes[1]/dataset['default'].count()*100


#Barplot of percentage of defaulters vs non defaulters.
plt.bar(['No Default','Default'], classes, color=['salmon','lightblue'])
plt.ylabel('Number of transactions')
plt.annotate("{0:.4}%".format(not_default),(0.2, 0.5), xycoords='axes fraction')
plt.annotate("{0:.4}%".format(defaulter),(0.7, 0.5), xycoords='axes fraction')
plt.show()
```

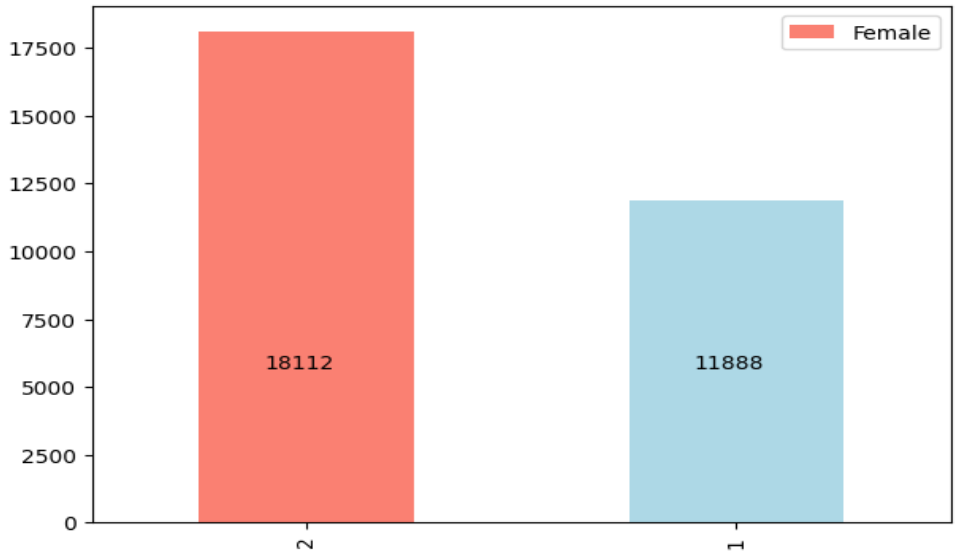


22.12% of the customers are going to default

4.2 Bar plot of the number of Male vs Female customers:

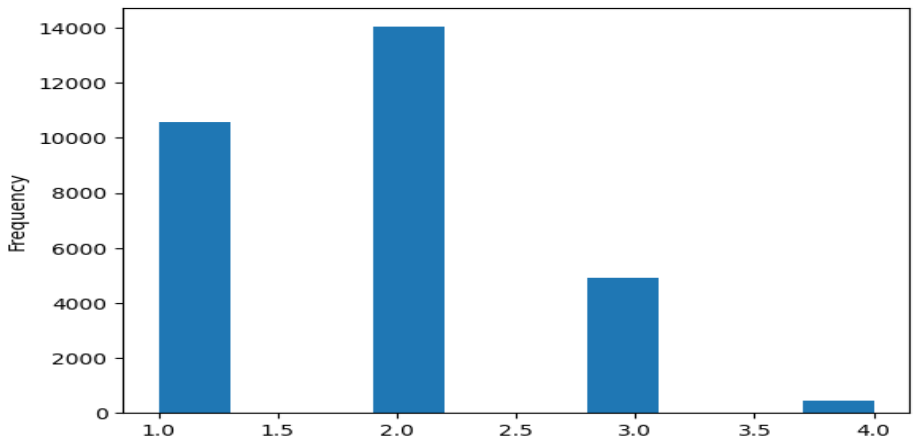
```
dataset['SEX'].value_counts().plot(kind="bar",figsize=(7,5),color=['salmon','lightblue'])

#Calculating annotated vales
sex_df=dataset['SEX'].value_counts().reset_index()
female=sex_df['SEX'][0]
male=sex_df['SEX'][1]
plt.annotate("{}".format(female),(0.2, 0.3), xycoords='axes fraction')
plt.annotate("{}".format(male),(0.7, 0.3), xycoords='axes fraction')
plt.legend(['Female','Male'])
plt.show()
```



There are more number of female customers than male customers.

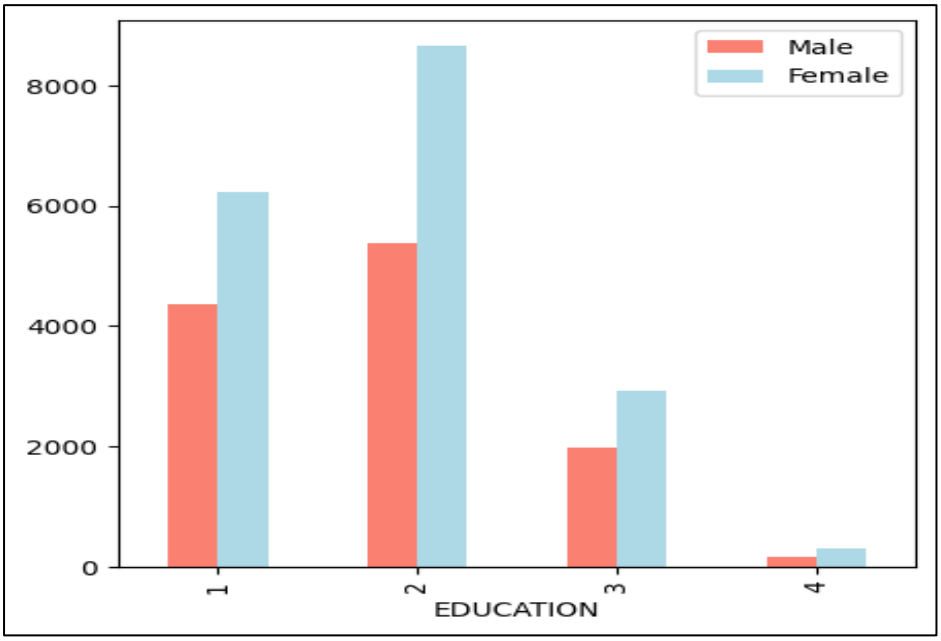
4.3 Bar graph showing the number of customers as per education using dataset.EDUCATION.plot.hist()



Maximum number of customers are university graduate while least are from others category.

4.4 Gender-wise distribution of number of customers by education.

```
dataset.groupby('EDUCATION')['SEX'].value_counts().unstack().plot(kind="bar",figsize=(5,5),color=['salmon','lightblue'])
plt.legend(['Male','Female'])
plt.show()
```

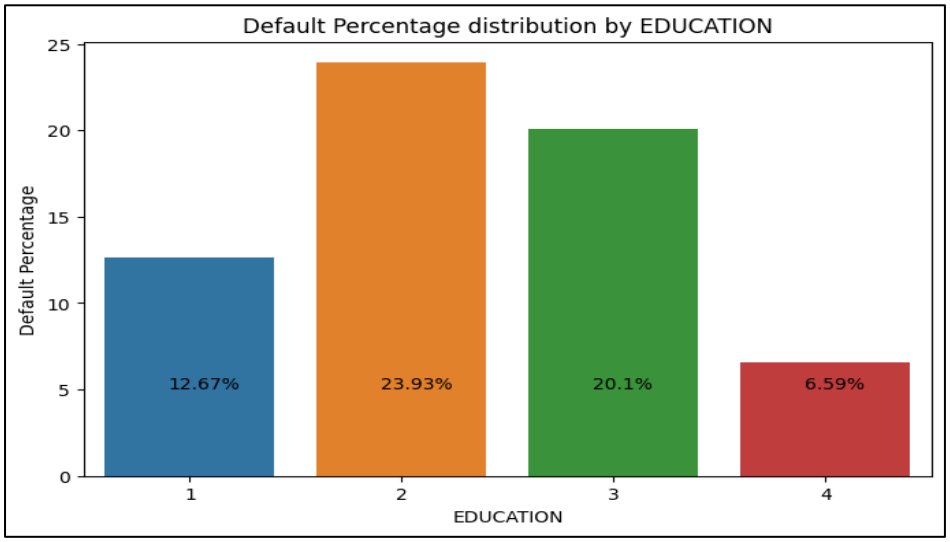


The maximum number of female customers are university graduates.

4.5 Bar graph showing default percentage distribution as per education.

```
dataset["EDUCATION"].value_counts()
df=dataset.groupby("EDUCATION")['default'].sum().reset_index()
df.rename(columns = {'default':'Default Percentage'}, inplace = True)
for i in range(dataset["EDUCATION"].nunique()):
    df['Default Percentage'][i]=round(df['Default Percentage'][i]/(df['Default Percentage'][i]+dataset["EDUCATION"].value_counts().reset_index()["EDUCATION"][i]),4)*100
#for plotting bar graph
fig = plt.subplots(figsize=(8,5))
sns.barplot(x ='EDUCATION',y ='Default Percentage',data = df).set_title('Default Percentage distribution by EDUCATION')
for i in range(df['Default Percentage'].nunique()):
   plt.annotate("{}%".format(df['Default Percentage'][i]),(0.1+i*0.25, 0.2), xycoords='axes fraction')

#plt.legend(df[x])
plt.show()
```

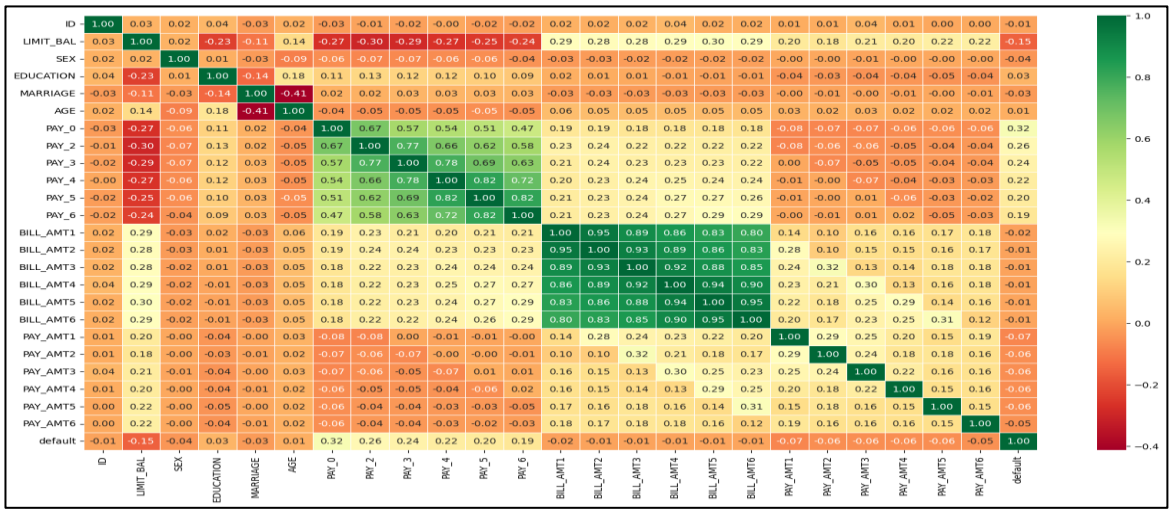


The maximum percentage of defaulters are university graduates while the least among the other categories.

4.6 Correlation Heatmap:

Heatmaps are used to show relationships between two variables, one plotted on each axis. By observing how cell colours change across each axis, you can observe if there are any patterns in value for one or both variables. Since we want to find the relationship between different variables in data frame and heatmap can be one of the ways to visualize it.

```
# Correlation Heatmap visualization code
corrmat = dataset.corr()
plt.figure(figsize=(20,10))
sns.heatmap(dataset[corrmat.index].corr(),annot=True,linewidths=0.5,fmt='.2f',cmap="RdYlGn")
```



- *PAY_0 to PAY_6 are highly correlated.*
- *BILL_AMT1 to BILL_AMT6 are highly correlated.*

4.7 Inference from the Exploratory Data Analysis and Visualisation:

1) Insights about Demographic Distribution:-

- 22.12% of the customers are going to default.
- Female customers are more than male customers.
- The number of female defaulters is more than male defaulters.
- There is not much difference in default percentage based on gender.

2) Insights about Education Qualification:-

- The maximum number of customers are university graduates across the gender while the least is from the others category.
- The absolute numbers and percentage-wise defaults are maximum among university graduates.
- Higher default percentage is observed in high school graduate males.

3) Insights about Marital status:-

- More customers are single than married.
- The absolute value and percentage of defaults are more among single customers.
- The maximum default percentage is observed in customers who are married and high school graduates.

4) Insights about Age Distribution

- The maximum number of customers have age between 25-35 years while very few are above 60 years of age.
- The maximum number of defaulting customers are between the age of 25 to 30.
- Maximum default percentage is observed in the age group of 20-25 yrs and 60-80 years of age category.

5. Hypothesis Testing

- After thoroughly analysing the dataset through visuals (graph Plots and Correlation Heatmap) we do Hypothesis Testing.
- **Hypothesis testing** is a statistical method used to make inferences about population parameters based on a sample of data. It involves formulating a hypothesis (Null Hypothesis H0 and Alternate Hypothesis H1), collecting and analyzing data, and drawing conclusions about the population based on the sample
- In hypothesis testing, various types of tests are employed to assess different aspects of data. However, we have employed the commonly used two types of tests for statistical analysis: **the two-tail test** and **the Chi-square test of independence**

5.1 Two-tail test

A two-tailed t-test is a statistical test used in hypothesis testing to determine if there is a significant difference between the means of two independent groups. The "two-tailed" part refers to the fact that the test considers the possibility of a difference in both directions, either a positive or a negative difference.

(a) Two-tail test for Age and Default Payment Status next month.

H0(null hypothesis): There is no significant difference in the age with respect to default payment next month.

H1(alternate hypothesis): There is a significant difference in the age with respect to default payment next month.

```
[20]
    from scipy.stats import ttest_ind

    default=dataset[dataset['default']==1]
    not_default=dataset[dataset['default']==0]

    p_val= ttest_ind(default['AGE'],not_default['AGE']).pvalue
    print(p_val)

    #95% Confidence interval
    if p_val < 0.05:
      print('We are rejecting null hypothesis')
    else:
      print('We are accepting null hypothesis')

...
    0.016136845890163835
    We are rejecting null hypothesis
```

The objective of this test is to assess whether there is a significant difference in the ages of individuals who defaulted on their credit card payments compared to those who did not.

- Two subsets of the dataset are created based on the 'default' column: one for individuals who defaulted (‘**default’**) and one for those who did not (‘**not_default’**)
- The **‘ttest_ind’** function from the **‘scipy.stats’** module is used to perform a two-sample independent t-test on the 'AGE' variable for the two groups (**default and not_default**).
- The p-value from the t-test is extracted and stored in the variable **‘p_val’**.
- The code then checks if the p-value is less than 0.05, which is a common significance level (95% confidence interval). If the p-value is less than 0.05, the null hypothesis is rejected; otherwise, the null hypothesis is accepted.

Inference: Here Null hypothesis is rejected. It implies that there is a significant difference in age with respect to default payments next month. In practical terms, this could suggest that age plays a statistically significant role in predicting credit card default. This could have implications for credit risk assessment or targeted financial planning based on age demographics. The specific direction of the difference (whether defaulters are generally older or younger) is not inferred from this test, only that a difference exists.

(b) Two t-tests for the limit balance and Default Payment Status next month.

H0(null hypothesis): There is no significant difference in the limit balance with respect to default payment next month.

H1(alternate hypothesis): There is a significant difference in the limit balance with respect to default payment next month.

```
[21]
    p_val= ttest_ind(default['LIMIT_BAL'],not_default['LIMIT_BAL']).pvalue
    print(p_val)

    #95% Confidence interval
    if p_val < 0.05:
      print('We are rejecting null hypothesis')
    else:
      print('We are accepting null hypothesis')

...
    1.3022439532597397e-157
    We are rejecting null hypothesis
```

The objective of this test is to assess whether there is a significant difference in the limit balance of individuals who defaulted on their credit card payments compared to those who did not.

Similar steps are followed as in the previous two-tailed test. But here the two-sample t-test is performed on the 'LIMIT_BAL' variable for the two groups (**default and not_default**).

Inference: Here Null hypothesis is rejected. It implies that there is a significant difference in Limit Balance with respect to default payments next month. It indicates that the limit balance is statistically significant in differentiating individuals who will default from those who won't.Depending on the direction of the significant difference, it may provide insights into the financial dynamics influencing default behavior. For example, higher or lower limit balances could be associated with increased or decreased default risk.

5.2 Chi-Square Test of Independence

The Chi-square test of independence is applied when dealing with categorical data and aims to assess whether there is a significant association between two categorical variables. It examines whether the observed distribution of frequencies differs from what would be expected under the assumption of independence between the variables. This test is commonly used in contingency table analysis.

(a) Chi-Square test for Education and Default Payment Status next month.

H0(null hypothesis): There is no significant dependency between the default payment next month and education.

H1(alternate hypothesis): There is a significant dependency between the default payment next month and education.

```
[22]
# Perform Statistical Test to obtain P-Value

from scipy.stats import chi2_contingency

# Create contingency table of price range and 4G connectivity
chi_table = pd.crosstab(dataset['default'], dataset['EDUCATION'])

# Perform chi-square test of independence
chi2_stat, p_val, dof, expected = chi2_contingency(chi_table)

#95% Confidence interval
if p_val < 0.05:
  print('We are rejecting null hypothesis')
else:
  print('We are accepting null hypothesis')

...  We are rejecting null hypothesis
```

The objective of this test is to assess whether there is a significant dependency between the education level of individuals who defaulted on their credit card payments.

- A contingency table ('**chi_table'**) is created using the '**pd.crosstab'** function, showing the counts of observations for each combination of education level and default status.
- The '**chi2_contingency'** function from '**scipy.stats'** module performs the chi-square test of independence on the contingency table. This test assesses whether the observed distribution of counts is significantly different from what would be expected under the assumption of independence between education level and default status.
- The variables **'chi2_stat', 'p_val', 'dof',** and **'expected'** represent key outputs from the chi-square test:
    - **chi2_stat (Chi-square Statistic):** The test statistic quantifies how much the observed counts in the contingency table deviate from the expected counts under the assumption of independence.
    - **p_val (p-value):** The p-value is the probability of observing a test statistic as extreme as, or more extreme than, the one calculated from the sample data, assuming that the null hypothesis (no association between variables) is true.
    - **dof (Degrees of Freedom):** The degrees of freedom are determined by the number of categories in the variables being tested. For a Chi-square test of independence in a contingency table, the degrees of freedom are calculated using the formula (number of rows - 1) * (number of columns - 1).
    - **Expected (Expected Counts):** This variable contains the expected counts for each cell in the contingency table under the assumption of independence. Expected values are calculated based on the marginal totals of the table.
- Then the code checks if the p-value is less than 0.05, which is a common significance level (95% confidence interval). If the p-value is less than 0.05, the null hypothesis is rejected; otherwise, the null hypothesis is accepted.

Inference: Here Null hypothesis is rejected. It implies that there is a significant dependency between the default payment next month and the education level of the individual. The rejection of the null hypothesis suggests that there is a statistically significant relationship or association between an individual's education level and their likelihood of defaulting on a credit card payment next month. The results may have implications for decision-making in credit risk assessment or financial planning. For instance, lenders might implement specific risk management practices for individuals with certain education levels, such as adjusting credit limits, interest rates, or payment terms based on the observed risk patterns.

(b) Chi-Square test for Marital Status and Default Payment Status next month.

H0(null hypothesis): There is no significant dependency between the default payment next month and marital status.

H1(alternate hypothesis): There is a significant dependency between the default payment next month and marital status.

```
[23]
# Create contingency table of price range and 4G connectivity
chi_table = pd.crosstab(dataset['default'], dataset['MARRIAGE'])

# Perform chi-square test of independence
chi2_stat, p_val, dof, expected = chi2_contingency(chi_table)

#95% Confidence interval
if p_val < 0.05:
  print('We are rejecting null hypothesis')
else:
  print('We are accepting null hypothesis')

...  We are rejecting null hypothesis
```

The objective of this test is to assess whether there is a significant dependency between the marital status of individuals who defaulted on their credit card payments

Similar steps are followed as in the previous Chi-Square test. But here the contingency table is created for the combination of default status and the Marriage.

Inference: Here Null hypothesis is rejected. It implies that there is a significant dependency between the default payment next month and the marital status of the individual. The rejection of the null hypothesis suggests that there is a statistically significant relationship or association between an individual's marital status and their likelihood of defaulting on a credit card payment next month. The results may have implications for decision-making in credit risk assessment or financial planning. Marital status may influence financial behaviors, including default, which can impact the design and marketing of financial products. So Financial institutions might customize product offerings and marketing strategies to cater to the unique financial needs and challenges associated with different marital status groups.

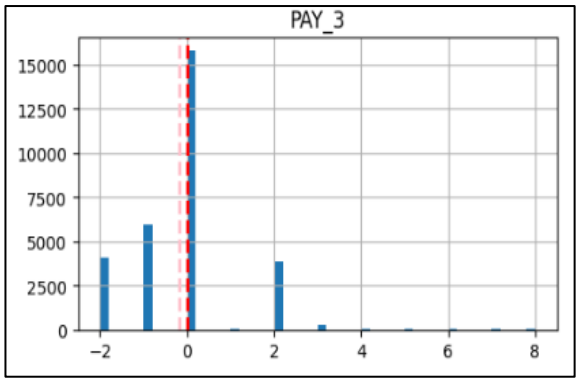
6. Data Pre-Processing

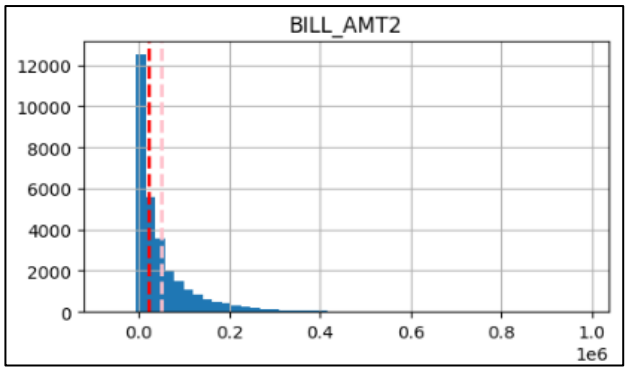
- After doing the Hypothesis Testing with a two-tailed test and Chi-Square test for Independence, we do the data pre-processing.
- At first Histograms are plotted for numerical features of the dataset, followed by normalization which is done with the help of MinMax Scaling.
- After Data splitting is done the dataset is split into training and testing subsets, followed by the StandardScaling for training and testing subsets.

6.1 Histogram Plots

```
#Checking for distribution of data using histogram
numeric_col = ['LIMIT_BAL', 'AGE', 'PAY_0','PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']
rows=5
cols=4
counter=1
fig = plt.figure(figsize=(20,15))
for col in numeric_col:
  plt.subplot(rows,cols,counter)
  ax = fig.gca()
  dataset[col].hist(bins=50, ax = ax)
  ax.axvline(dataset[col].mean(), color = 'pink',linestyle='dashed', linewidth=2)
  ax.axvline(dataset[col].median(), color = 'red',linestyle='dashed', linewidth=2)
  ax.set_title(col)
  counter=counter+1
  plt.tight_layout()
[24]
```

In the above code, Histograms are plotted for each numerical field in the dataset with the help of **'plt (pyplot)'** imported from the module **'matplotlib',** Two vertical lines are added to each histogram: one representing the mean (in pink) and another representing the median (in red).





Histograms for PAY_3 and BILL_AMT2

Here it can be seen the features PAY_3 and BILL_AMT2 have different ranges. We have to ensure that no features dominate others simply because of their scale, so it is required to normalize the features to a common scale.

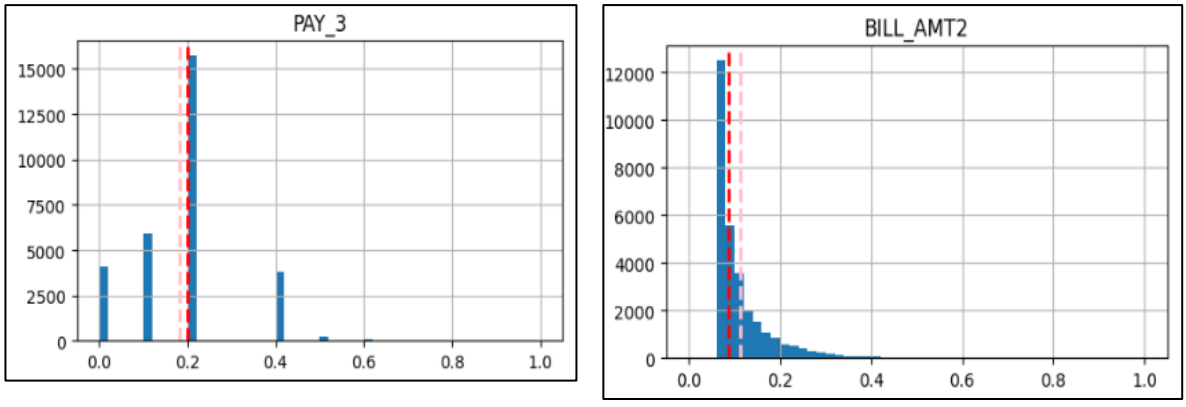
6.2 MinMax Scaling (Normalization)

```
[25]
    scaler = MinMaxScaler()
    dataset[numeric_col]= scaler.fit_transform(dataset[numeric_col])
```

- **MinMax** scaling is particularly useful when the features in the dataset have different ranges, and you want to bring them to a common scale without distorting the differences in the ranges.
- Here the line **'MinMaxScaler()'** function is used from the **'sklearn. preprocessing'** module. The **'MinMaxScaler'** is used for scaling numerical features to a specific range, usually between 0 and 1
- **'scaler.fit_transform(dataset[numeric_col])'** fits the scaler to the selected numeric columns and transforms the values using the Min-Max Scaling. The **'fit_transform'** method calculates the minimum and maximum values of each column and scales the values accordingly.

After executing this code, the specified numeric columns in the dataset will be transformed such that their values are within the range [0, 1]. **The main reason behind doing MinMax Scaling is that "some machine learning models, such as support vector machines (SVMs) and k-nearest neighbors (KNN), are sensitive to the scale of input features. MinMax scaling can improve the performance and convergence of these models.**

Again, the Histograms are plotted for numerical features of the dataset after doing the MinMax Scaling. Now the numeric features are normalized to a common scale between 0 to 1.



Histograms for PAY_3 and BILL_AMT2 after MinMax Scaling (Normalization)

6.3 Data Splitting

Data Splitting involves dividing a dataset into at least two subsets: a training set and a test set. The primary purpose of data splitting is to have a designated subset (the training set) on which the machine learning model is trained. The training set is crucial for the model to adjust its parameters and make predictions. The test set is reserved for evaluating the performance of the trained model on data it has never seen before. This helps to assess how well the model generalizes to new, unseen data. It provides an estimate of how the model is expected to perform on real-world, future data.

```
X=dataset.drop(['default','Limit_cat','Age_cat','ID'],axis=1)#Dropping categorical variable created during feature engineering
y=dataset['default']


# Split your data to train and test. Choose Splitting ratio wisely.
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 = 0)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

[27]

```
(24000, 23)
(6000, 23)
(24000,)
(6000,)
```

- **Selecting the features first**. **X = dataset.drop(['default', 'Limit_cat', 'Age_cat', 'ID'], axis=1)** It creates a new DataFrame X by dropping the specified columns ('default', 'Limit_cat', 'Age_cat', 'ID') from the original dataset, the **axis=1** parameter specifies that the columns are to be dropped. **y = dataset['default'],** y containing the target variable ('default' column) that the model aims to predict.
- Splitting of the dataset is done into training and testing sets with the help of **'train_test_split'** imported from the module **'sklearn.model_selection'.** The parameters are
  - **X:** features
  - **y:** Target variable,
  - **test_size:** The proportion of the dataset to include in the test split. Here, it's set to 20%, meaning 80% of the data will be used for training, and 20% for testing.
  - **random_state:** It is set to 0 for reproducibility, ensuring that the same split is obtained each time the code is run.
- Then shapes of training and testing sets are printed.

```
[28]
train_scaler = StandardScaler()
test_scaler=StandardScaler()

#Fit on data
X_scaled=train_scaler.fit_transform(X)
X_train_scaled=train_scaler.fit_transform(X_train)
X_test_scaled=test_scaler.fit_transform(X_test)

#converting scaled data into dataframe
X_scaled_df=pd.DataFrame(data=X_scaled,columns=X.columns,index=X.index)
X_scaled_train_df=pd.DataFrame(data=X_train_scaled,columns=X_train.columns,index=X_train.index)
X_scaled_test_df=pd.DataFrame(data=X_test_scaled,columns=X_test.columns,index=X_test.index)
```

- Now standardization or z-score normalization of features is done, standardization is a common practice to transform the features of the dataset so that they have a mean of 0 and a standard deviation of 1.
- It is done by **'StandardScaler'** imported from the '**sklearn.preprocessing'** module. Two instances of the **StandardScaler** class are created. These instances will be used to scale the features of the training set (**train_scaler**) and the test set (**test_scaler**) separately.
- **train_scaler.fit_transform(X)** fits the **train_scaler** on the entire feature set **X** and transforms the features, ensuring that the mean and standard deviation are computed based on the entire set. A similar thing is done for the training set **train_scaler.fit_transform(X_train)** and test set **train_scaler. fit_transform(X_test)**. But a separate scaler is used for the test set, it ensures that the scaling parameters are consistent with the training set but computed independently to avoid data leakage.
- After this process, the standardized features are stored in the DataFrames **X_scaled_df**, **X_scaled_train_df**, and **X_scaled_test_df**. These standardized features can then be used for training and evaluating machine learning models.

```
[29]
X_scaled_df.head(2)
```

Python

	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	PAY_5	...	BILL_AMT3	BILL_AMT4	BILL_AMT5	BILL_
0	-1.136720	0.810161	0.21187	-1.068797	-1.246020	1.794564	1.782348	-0.696663	-0.666599	-1.530046	...	-0.667993	-0.672497	-0.663059	-0.6
1	-0.365981	0.810161	0.21187	0.849131	-1.029047	-0.874991	1.782348	0.138865	0.188746	0.234917	...	-0.639254	-0.621636	-0.606229	-0.5

2 rows × 23 columns

When X_scaled_df.head(2) is executed, we see a tabular representation of the first two rows of the standardized features in X_scaled_df. The **'.head(2)'** method displays the first two rows of the data frame. The number **2** passed to the **head** specifies the number of rows to be shown. If no parameter is provided, it defaults to showing the first five rows. Each column will represent a feature, and the values will be the standardized versions of the original feature values from X. **This gives a quick glimpse of how the data looks after the standardization process.**

7. Machine Learning Classification Model

- After the data pre-processing, the training and testing set is passed to different ML models for fitting (training) and scoring (performance or Accuracy)
- First the comparison of Model is done using cross Validation then the fit and score of the models are done
- We have used different ML classification models: LogisticRegression, SupportVectorClassifier(SVC), KNeighborsClassifier, and RandomForestClassifier.

7.1 Comparing the Models using Cross Validation

```
[30]
models=[LogisticRegression(max_iter=1000), SVC(kernel='linear'),KNeighborsClassifier(), RandomForestClassifier(random_state=0,
criterion='entropy', n_estimators=4)]
```

```
[31]
def compare_models_cross_validation():
  model_scores={}

  for model in models:

    cv_score = cross_val_score(model, X_train_scaled, y_train, cv=5)

    mean_accuracy = sum(cv_score)/len(cv_score)
    mean_accuracy = mean_accuracy*100
    mean_accuracy = round(mean_accuracy, 2)

    print('Cross Validation accuracies for the',model,'=', cv_score)
    print('Acccuracy score of the ',model,'=',mean_accuracy,'%')
    print('----------------------------------------------------------------')
```

- The **compare_models_cross_validation()** function evaluates multiple machine learning models using cross-validation and prints the cross-validation accuracies for each model. This helps in comparing how well the models perform on different subsets of the training data and aids in selecting the best-performing model for further analysis or fine-tuning.
- The **cross_val_score** method imported from the **'sklearn.model_selection'** module is used to perform cross-validation on the current model (**model**) using the training set (**X_train_scaled** and **y_train**). **cv=5** specifies 5-fold cross-validation, dividing the training set into 5 subsets (folds) and using them in turns as a validation set while the model is trained on the rest
- The mean accuracy across the folds is calculated and expressed as a percentage. The accuracy is rounded to two decimal places for clarity.
- **The cross-validation accuracies for the current model and its mean accuracy are printed to the console.**

```
[32]  compare_models_cross_validation()
                                                                                    Python
...   Cross Validation accuracies for the LogisticRegression(max_iter=1000) = [0.808125   0.81270833 0.80395833 0.80416667 0.80708333]
      Acccuracy score of the  LogisticRegression(max_iter=1000) = 80.72 %
      ----------------------------------------------------------------
      Cross Validation accuracies for the SVC(kernel='linear') = [0.80770833 0.785625   0.80791667 0.80520833 0.80645833]
      Acccuracy score of the  SVC(kernel='linear') = 80.26 %
      ----------------------------------------------------------------
      Cross Validation accuracies for the KNeighborsClassifier() = [0.78520833 0.79583333 0.79645833 0.790625   0.790625  ]
      Acccuracy score of the  KNeighborsClassifier() = 79.18 %
      ----------------------------------------------------------------
      Cross Validation accuracies for the RandomForestClassifier(criterion='entropy', n_estimators=4, random_state=0) = [0.79166667 0.80479167 0.795625   0.7
      Acccuracy score of the  RandomForestClassifier(criterion='entropy', n_estimators=4, random_state=0) = 79.67 %
      ----------------------------------------------------------------
```

Cross Validation and Mean Accuracy for Different ML Classification Models

- Logistic regression Classifier Has Accuracy of 80.72 %
- Support Vector Classifier Has Accuracy of 80.26 %
- KNeighbors Classifier Has an Accuracy of 79.18 %
- RandomForest Classifier Has an Accuracy of 79.67 %

Logistic Regression Classifier has the highest Mean Accuracy, Calculated by Cross Validation

7.2 Fitting and Scoring of ML classification models

Fitting a model refers to the process of training or teaching the model on a dataset. During the fitting process, the model adjusts its internal parameters based on the provided training data. The model learns patterns, relationships, and underlying structures in the data that allow it to make predictions. The primary purpose of fitting a model is to enable it to generalize and make accurate predictions on new, unseen data.

Scoring a model involves evaluating its performance on a specific dataset using a predefined metric. Once a model is trained, it needs to be assessed on how well it can make predictions on data it hasn't seen during training. A scoring metric is used to quantify the model's performance, and the result is a numerical value representing how well the model is doing. The purpose of scoring a model is to assess its effectiveness, understand how well it generalizes to new data, and compare its performance with other models.

```
[36]  models={
          'Logistic Regression':LogisticRegression(),
           'KNN':KNeighborsClassifier(),
          'Support Vector':SVC(),
           'Random Forest':RandomForestClassifier(criterion='entropy', n_estimators=4, random_state=0)
      }
```

```
[37]
#Create function to fit and score models
def fit_and_score(models,X_train,X_test,y_train,y_test):

  # Dictionary to keep model scores
  models_trained={}
  model_scores={}
  clf_report={}

  #Loop through models
  for name, model in models.items():
    #Fit the model to the data
    model=model.fit(X_train,y_train)
    models_trained[name]=model

    #Evaluate the model and append its accuracy score to model_scores
    model_scores[name]=model.score(X_test,y_test)

    #Predicting with tuned model
    y_preds=model.predict(X_test)

    #Creating classification report and storing it in dictionary
    clf_report[name]=classification_report(y_test,y_preds)

  #Saving all model accuracy scores in dataframe
  model_compare=pd.DataFrame(model_scores,index=['accuracy'])

  return model_scores,model_compare,clf_report, models_trained
```

- The function **'fit_and_score'** takes a dictionary of machine learning models, training, and testing data. The purpose of this function is to streamline the process of training multiple models, evaluating their performance, and storing relevant information for later analysis.
- **Three dictionaries** are initialized to store the trained models **(models_trained),** accuracy scores**(model_scores),** and classification reports **(clf_reports)** for each model. The function iterates through each model in the provided dictionary.
- The current model is fitted to the training data by **model.fit(X_train,y_train**) , and the trained model is stored in the **models_trained** dictionary. The accuracy score of the model on the test data is calculated by **model.score(X_test,y_test)** and stored in the **model_scores** dictionary.
- The model makes predictions on the test data (**X_test**) by **model.predict(X_test),** and a classification report is generated using the **classification_report** function imported from module **sklearn.metrics**.
- A data frame (**model_compare**) is created to store the accuracy scores of each model. The function returns the accuracy scores, model comparison DataFrame, classification reports, and the trained models.

```
[38]
#Model implementation using function
model_scores=fit_and_score(models=models,X_train=X_scaled_train_df,X_test=X_scaled_test_df,y_train=y_train,y_test=y_test)

[39]
#Printing accuracy score of all the models
pd.DataFrame([model_scores[0]]).rename(index={0: 'Score'})
```

	Logistic Regression	KNN	Support Vector	Random Forest
Score	0.820833	0.790833	0.825833	0.799333

After fitting and scoring the model by fit_and_score function on scaled trained and test dataset. We have reported the accuracy score for all the models.

- Accuracy score for Linear Regression is 0.820833
- Accuracy score for KNN is 0.790833
- Accuracy score for Support Vector is 0.825833
- Accuracy score for Random Forest is 0.79933

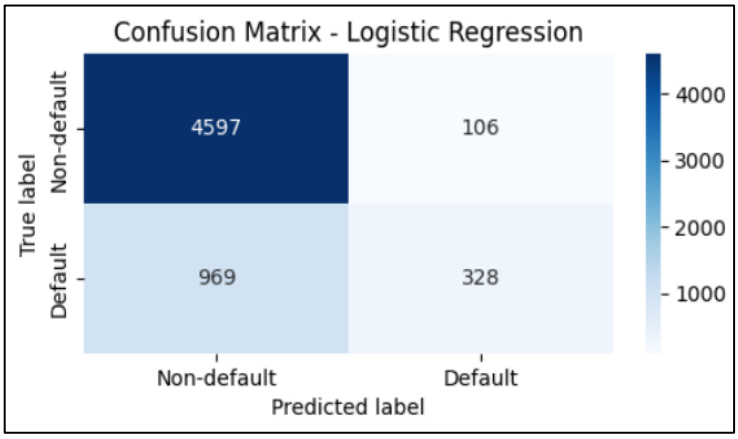
On our given dataset, the Support Vector Classifier performs the best.

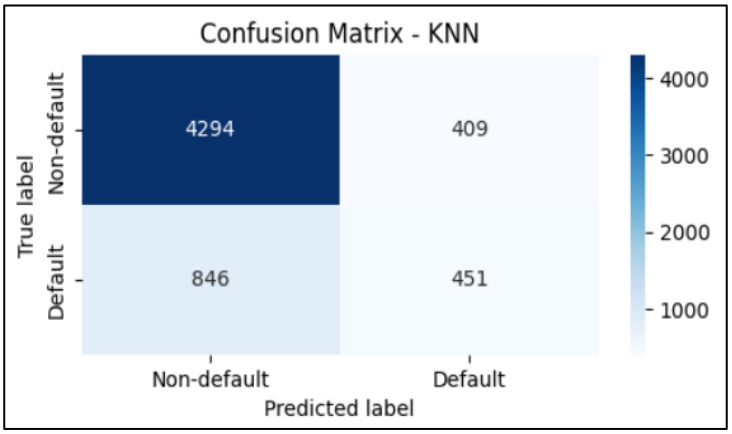
8. Model Evaluation Using Confusion Matrix

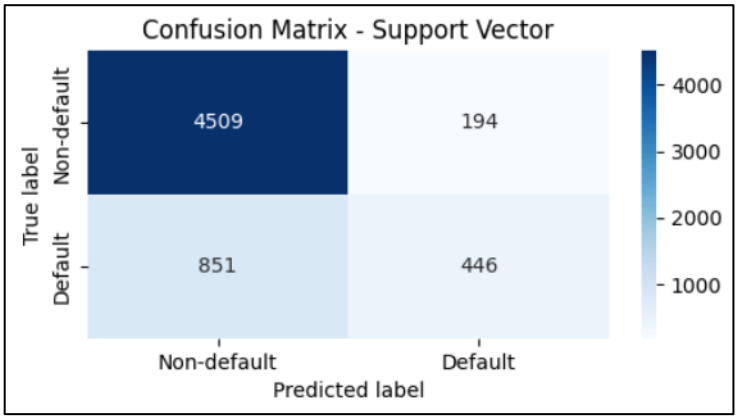
A confusion matrix is a performance measurement tool in machine learning, particularly in classification tasks, that provides a comprehensive summary of a model's predictions. The matrix compares predicted class labels to actual class labels, breaking down the outcomes into four categories: true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN).

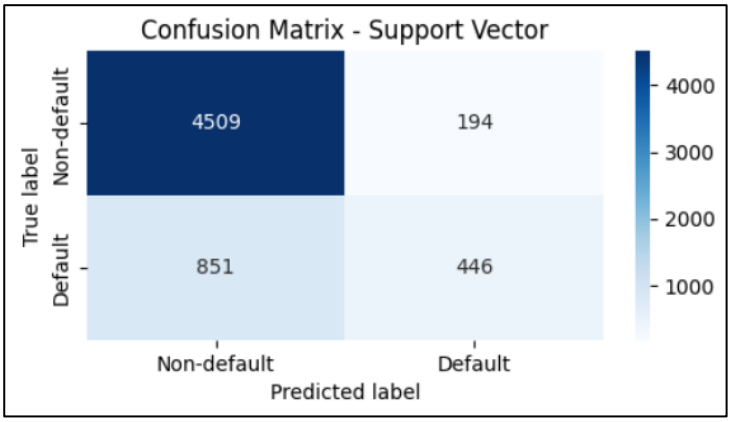
```
[40]
#Confusion matrix for all the implemented models
rows=3
cols=3
counter=1
fig = plt.figure(figsize=(15,8))
for ind,key in enumerate(model_scores[3].keys()):
    y_score = model_scores[3][key].predict(X_scaled_test_df)
    ConfMatrix = confusion_matrix(y_test,y_score)
    plt.subplot(rows,cols,counter)
    sns.heatmap(ConfMatrix,annot=True, cmap="Blues", fmt="d", xticklabels = ['Non-default', 'Default'],
    yticklabels = ['Non-default', 'Default'])
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.title(f"Confusion Matrix - {key}")
    counter=counter+1
    plt.tight_layout()
```

Confusion Matrix is plotted for all four classifiers that we have used. The code used the classification report got from the 'fit_and_score' function, correlation heatmap, the **'confusion_matrix'** imported from **sklear.metrics** module to plot the Confusion Matrix for all four classifiers.









Confusion Matrix for Logistic Regression, KNN, Support Vector, Random Forest Classifiers

```
#Printing classification report of all implemented models
for key in model_scores[2].keys():
  print(f'{key}:\n-------------------------------------------------------
  \n-------------------------------------------------------\n{model_scores[2][key]}')
```

[42]

At last printing the classification report for all the implemented models.

```
Logistic Regression:
------------------------------------------------------------
------------------------------------------------------------
              precision    recall  f1-score   support

           0       0.83      0.98      0.90      4703
           1       0.76      0.25      0.38      1297

    accuracy                           0.82      6000
   macro avg       0.79      0.62      0.64      6000
weighted avg       0.81      0.82      0.78      6000
```

KNN:

	precision	recall	f1-score	support
0	0.84	0.91	0.87	4703
1	0.52	0.35	0.42	1297
accuracy			0.79	6000
macro avg	0.68	0.63	0.65	6000
weighted avg	0.77	0.79	0.77	6000

Support Vector:

	precision	recall	f1-score	support
0	0.84	0.96	0.90	4703
1	0.70	0.34	0.46	1297
accuracy			0.83	6000
macro avg	0.77	0.65	0.68	6000
weighted avg	0.81	0.83	0.80	6000

Random Forest:

	precision	recall	f1-score	support
0	0.82	0.95	0.88	4703
1	0.59	0.24	0.34	1297
accuracy			0.80	6000
macro avg	0.70	0.60	0.61	6000
weighted avg	0.77	0.80	0.77	6000

Logistic Regression: Achieves good accuracy among the models (82%). Struggles with recall for default class, indicating challenges in correctly identifying instances of the positive class.

K-Nearest Neighbours (KNN): Shows reasonable accuracy but has a lower recall for default class, suggesting difficulty in capturing true positive instances.

Support Vector: Outperforms other models in terms of accuracy (83%). Demonstrates better precision and recall for Class 1, making it effective in identifying positive instances.

Random Forest: Provides a good balance between precision and recall but has lower overall accuracy compared to the Support Vector model.

9. Conclusion

In this project, we have performed Data Analysis or Visualisation, Hypothesis Testing, and Data Pre-processing on the **default credit card client's** dataset**.** Then we trained on different Machine learning Classification models: Logistic Regression, KNN, Support Vector, and Random Forest. **The Support Vector** model emerges as the most effective in this context, showcasing the highest accuracy and better precision-recall trade-off for positive instances. This characteristic can be valuable in scenarios where false positives have significant consequences. **Challenges with defaulters Class Prediction**: Across models, there is a common challenge in effectively predicting instances of the defaulter's class, as indicated by lower recall and F1-score for defaulters Class.

10. References

1. https://archive.ics.uci.edu/
2. https://online.stanford.edu/courses/cs229-machine-learning
3. https://www.datacamp.com/blog/classification-machine-learning
4. IDS course material