Topic :<u>Predicting whether a customer will default on his/her credit card</u>

For capstone project using machine learning(classification)

Kundan Lal, Abhijeet Kulkarni, Pankaj Ganjare, AkshayAuti

Data Science Trainees

Alma Better

Abstract:

Our study encompasses the findings done Predicting whether a customer will default on his/her credit card (default on credit card) csv data file.

In this we tackles the problem of Predicting whether a customer will default on his/her credit .

The date set consists of 30000 rows and 25 columns .

In this vein, we investigates the efficacy of standard machine learning techniques namely classification using algorithms likeLogistic Regression, KNN classifier , XGBOOST classifier, Naïve baiyes, SVM, Decision tree, hyperparameter etc, analyzing their performance with respect to each other.

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.

Data Description

Attribute Information:

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

- X1: Amount of the given credit (NT dollar): it includes both the individual consumer credit and his/her family (supplementary) credit.
- X2: Gender (1 = male; 2 = female).
- X3: Education (1 = graduate school; 2 = university; 3 = high school; 4 = others).
- X4: Marital status (1 = married; 2 = single; 3 = others).
- X5: Age (year).

- X6 - X11: History of past payment. We tracked the past monthly payment records (from April to September, 2005) as follows: X6 = the repayment status in September, 2005; X7 = the repayment status in August, 2005; . . .;X11 = the repayment status in April, 2005. The measurement scale for the repayment status is: -1 = pay duly(properly pay); 0 = not delay;, 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.

Mount the drive and import the dataset

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

```
df = pd.read_csv('/content/drive/MyDrive/Almabetter/M
L Project Classification Credit Card/default of credi
t card clients.xls - Data.csv', header=1)
```

Below is the original actual info about the data set using df2.info() method

```
<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 payment next month  30000 non-null  int64
dtypes: int64(25)
memory usage: 5.7 MB
```

```
#Importing all necessary Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifi
er
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import roc_auc_score, confusion_
matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report , c
onfusion_matrix
from sklearn.model_selection import cross_val_score
```

Data Preprocessing

This dataset contains information on default payments, demographic factors, credit limit, history of payments, and bill statements of credit card clients in Taiwan from April 2005 to September 2005. It includes 30,000 rows and 25 columns, and there is no credit score or credit history information.

Overall, the dataset is very clean, but there are several undocumented column values.

These transformations allowed us to extract certain keyfeatures and pertinent in furtheranalysis.Such as null values, unique values, rename of columnsfor easier analysis

Data Visualization

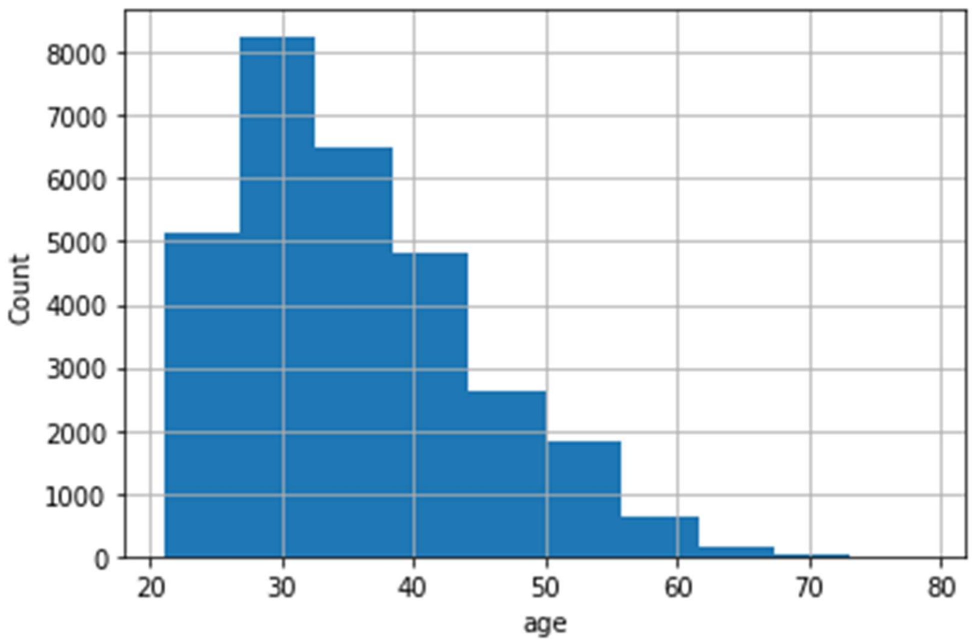
Objective

After preprocessing of the data set ,we first wanted to use data visualization on the dataset to find out,The type of relation of all independent variablesvs dependant variables, to check whether they arehow individuly affect the analysis.

Findings

Here we can clear that count of credit card user on basis of age

```
# Analysing based on Age(in year)
df['AGE'].hist()
plt.xlabel('age')
plt.ylabel('Count')
```

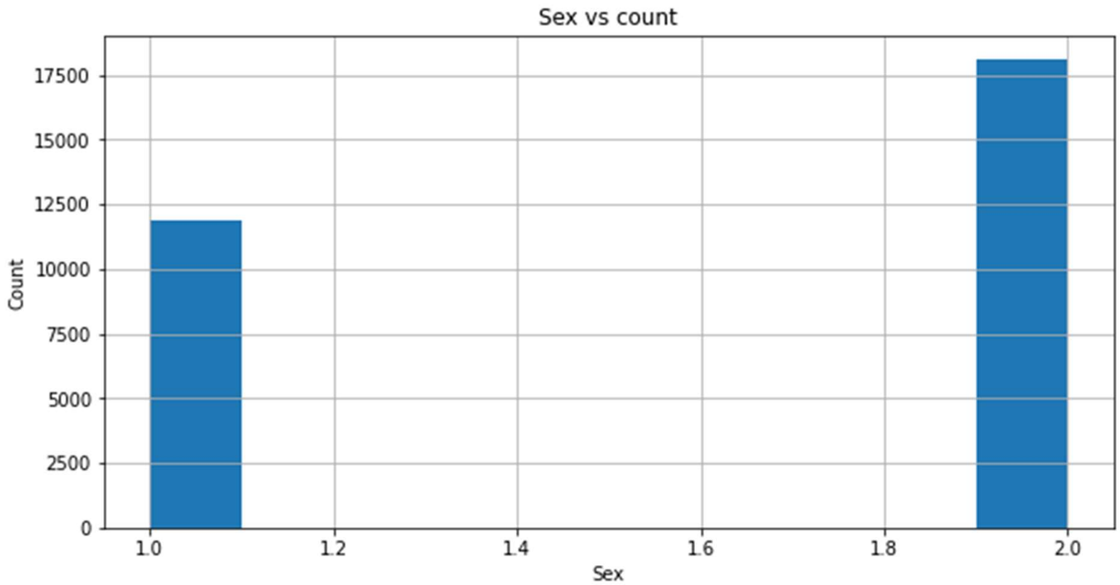


From above bar graph we can analyse that credit card holders are maximum from age 28 to 40

Finding

Here we can know that no of males and females credit card holders

```
# Analysis based on Gender (1 = male; 2 = female)
plt.figure(figsize=(10,5))
df['SEX'].hist()
plt.xlabel('Sex')
plt.ylabel('Count')
plt.title('Sex vs count')
```

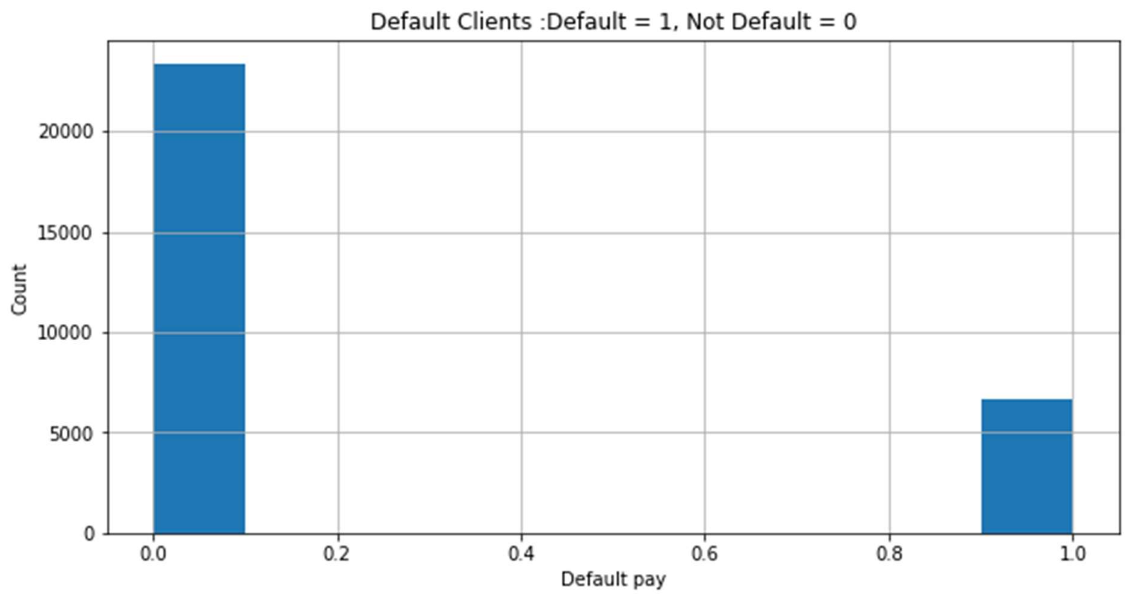


Females contains more numbers of credit cards as compare to males

Finding

Total no of default and non default credit card holders

```
#Numbers of Default and Not Default credit card holders
plt.figure(figsize=(10,5))
df['default_payment_next_month'].hist()
plt.xlabel('Default pay')
plt.ylabel('Count')
plt.title('Default Clients :Default = 1, Not Default = 0')
```

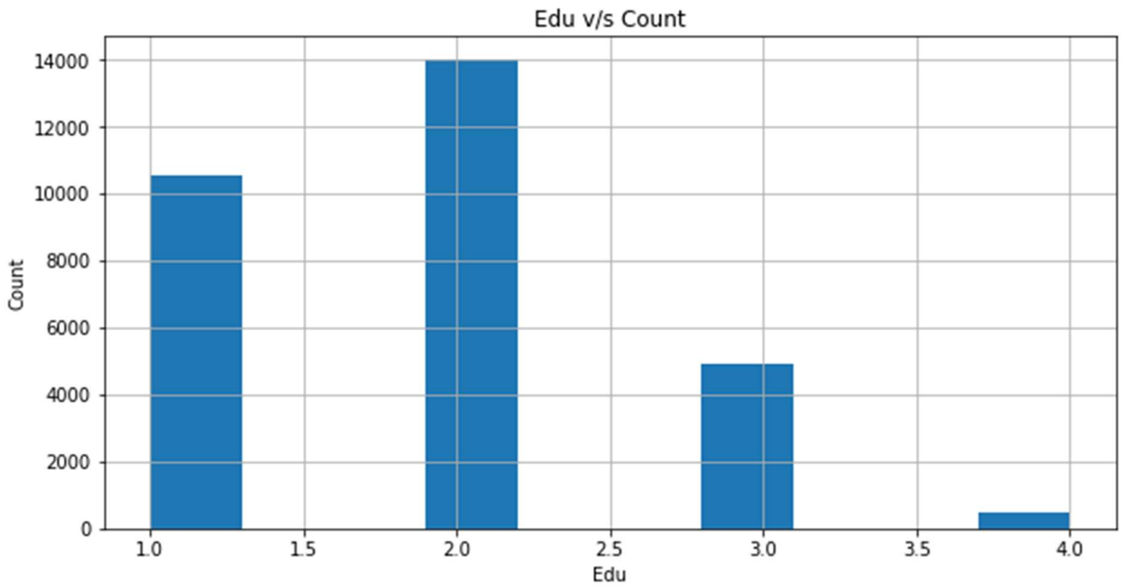


From graph we say that defaulters are less as compare to non defaulters credit card holders

Finding

We found tha no of credit card holders analysis on the basis of education

```
# Analysing on Education Basis (1 = graduate school; 2 = univ
ersity; 3 = high school; 4 = others)
plt.figure(figsize=(10,5))
df['EDUCATION'].hist()
plt.xlabel('Edu')
plt.ylabel('Count')
plt.title('Edu v/s Count')
```

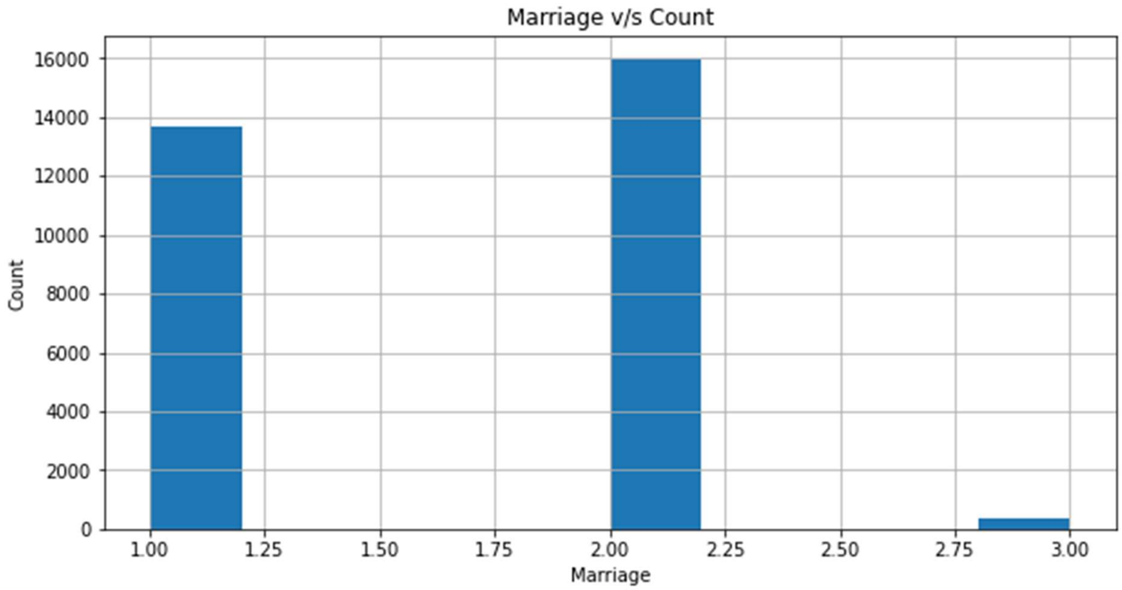


From graph it is known that no people from university school have more no of credit card holders followed by graduates school ,high school etc.

Finding

We have to find no of married and no of single people

```
# Analysing on Marriage Basis (1 = married; 2 = single; 3 = oth
ers)
plt.figure(figsize=(10,5))
df['MARRIAGE'].hist()
plt.xlabel('Marriage')
plt.ylabel('Count')
plt.title('Marriage v/s Count')
```

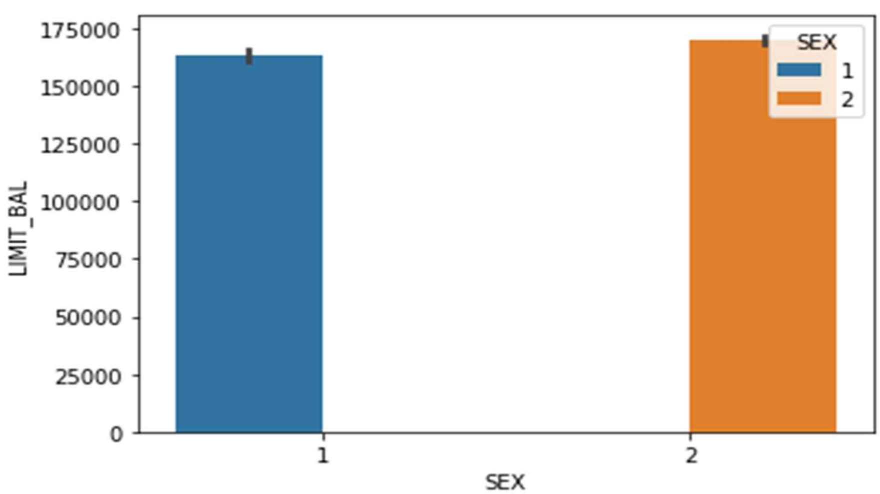


More number of credit cards holder are Singles followed by Married ones.

Finding

Here we found that limit balance on the basis of gender 1 is for male and 2 is for females

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

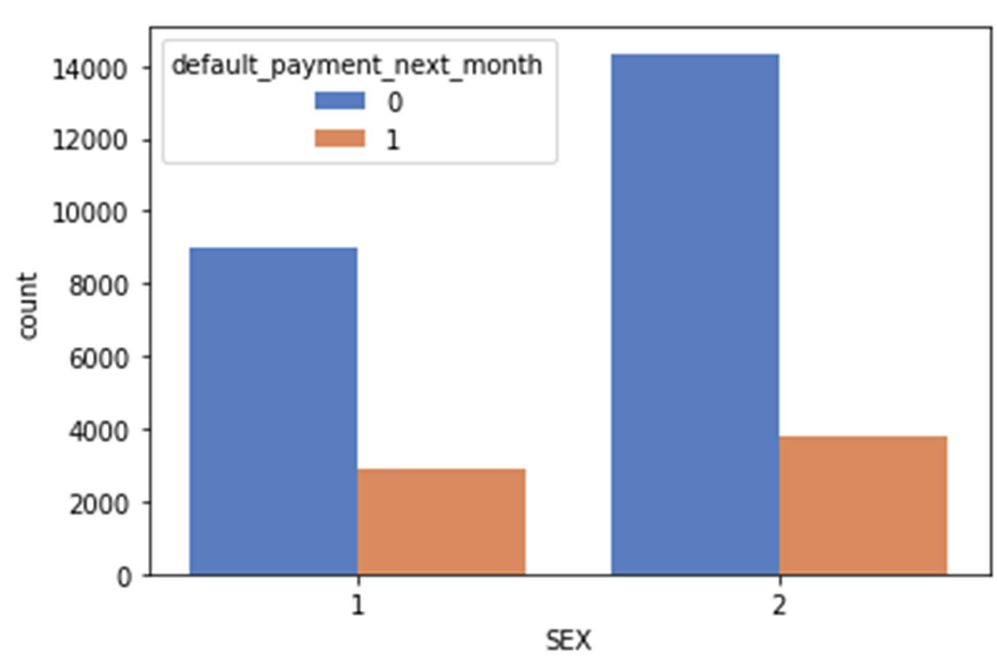


From graph we know that females have more no f credit card holders as compared to males

Finding

We have find that no of non default credit card holders in gender .

```
sns.countplot(x='SEX, data=df,hue="default_payment_next_month", palette="muted")
```

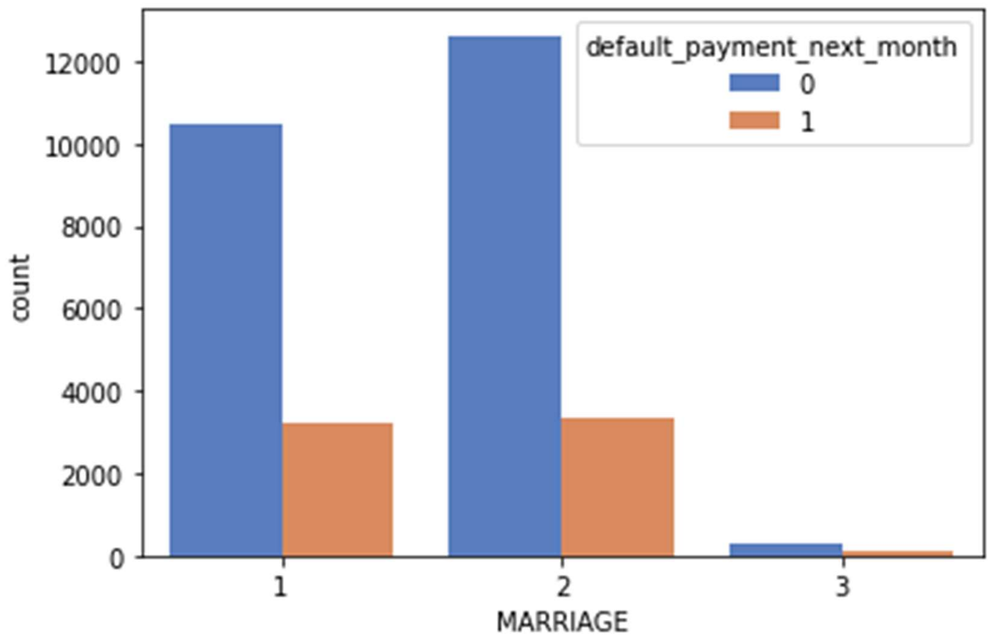


In Males, Non Default credit card holders has highest numbers present. In Females, non Default credit card holders has highest numbers present.

Finding

Finding that counting of default payment next month on married and single credit card holder

```
g=sns.countplot(x="MARRIAGE", data=df,hue="default_payment_next_month", palette="muted")
```

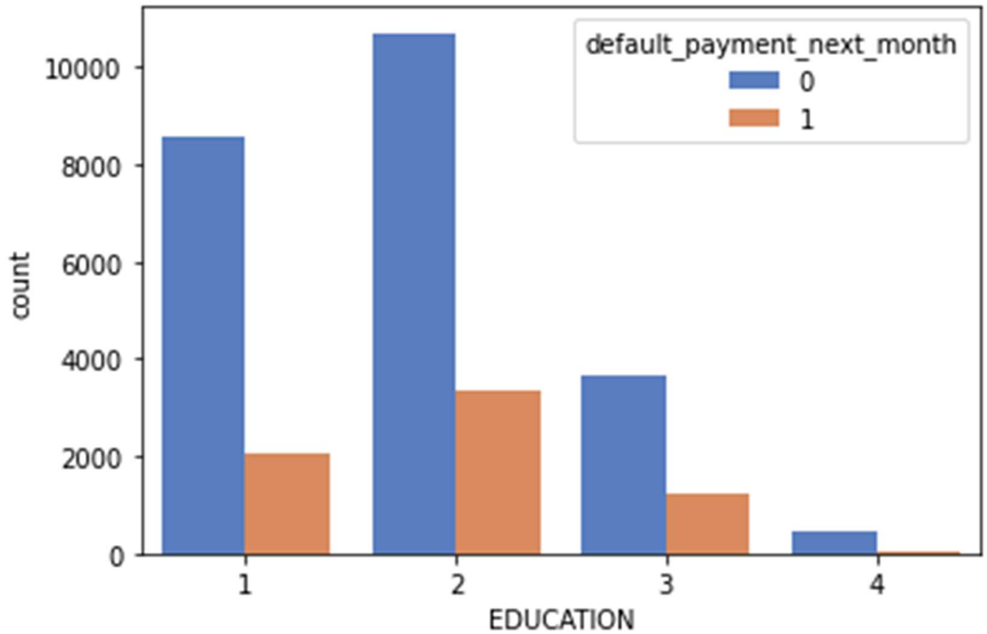


From plot it is clear that people who have marital as status single have more default payment wrt married status people

Finding

Count of default payment next month on education basis

```
g=sns.countplot(x="EDUCATION", data=df,hue="default_payment_next_month", palette="muted")
```



From plot it is clear that people from university have more default payment wrt to all other

Dropping Unwanted Columns

```
# Removing ID Columns from the datasets
df.drop('ID', axis=1, inplace=True)
```

Dropping not required column 'ID' as there was no help from this column in our Machine learning models so we planned to drop it ..,

'ID' This column had Customer Id numbers so was of no help for us for our Train and test Algorithms so we dropped this independent variable ..,

Standard Normalization Process

```
# Normalizing done on all independent variables
from sklearn.preprocessing import StandardScaler
```

```
scaling = StandardScaler()
X = df.iloc[:,0:-1]
X = scaling.fit_transform(X)
```

Normalization is a scaling technique in Machine Learning applied during data preparation to change the values of numeric columns in the dataset to use a common scale. It is not necessary for all datasets in a model. It is required only when features of machine learning models have different ranges.

StandardScaler library is used **to resize the distribution of values** so that the mean of the observed values is 0 and the standard deviation is 1.

As usually the dataset has some of the columns which are not on similar scale as others so before pushing it into machine Algorithms it important to scale them on common scale .

Splitting the data in 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=0.3, random_st
ate=0)
print(X_train.shape)
print(X_test.shape)
```

The train test split technique **can be used for classification and regression problems to test machine learning algorithms**. The procedure takes the given dataset and splits it into two subsets: namely Training Dataset & Train Dataset,

Training dataset: it is used to train the algorithm and fit the machine learning model.

one of the most important mechanisms in machine learning is to train your algorithm on a training set that is separate and distinct from the test set .

Test dataset is used to gauge the model's accuracy. If the testing accuracy is low , it means the model's accuracy of prediction is low .

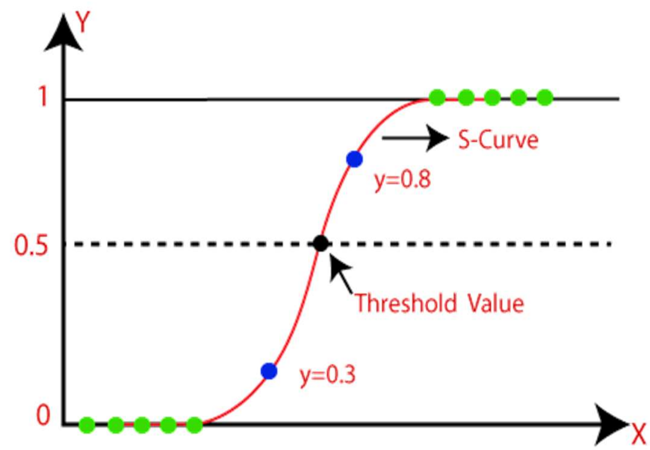
Algorithms used for machine learning by us,

- Logistics Regression
- KNN Classifier
- XGBoost
- Naïve Bayes
- SVM
- Decision Tree

Logistics Regression

```
# Importing the Logistic Regression Model from the Li
brary
from sklearn.linear_model import LogisticRegression
logmodel = LogisticRegression(random_state=1)
logmodel.fit(X_train,y_train)
```

Logistic regression is an example of supervised learning. It is used to calculate or predict the probability of a binary (yes/no) event occurring.



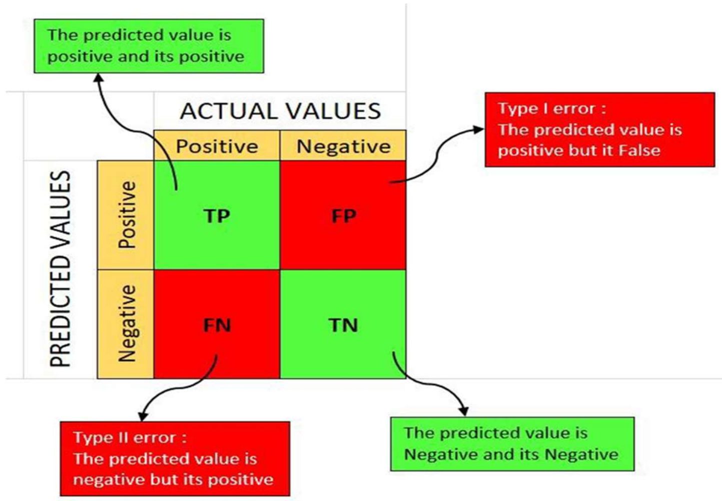
Y-Train Results

Model	Accuracy	Precision	Recall	F1 Score	ROC
Logistic Regression	0.807429	0.70529	0.238501	0.356461	0.604898

Y –Test Results

Model	Accuracy	Precision	Recall	F1 Score	ROC
Logistic Regression	0.816556	0.738056	0.230928	0.351786	0.604203

```
# Confusion Matrix
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```



Confusion Matrix of Logistics Regression

- [[6901 159]

  [1492 448]]

Hyper Parameter Tuning Done

```
logmodel_params = {'C': [0.001, 0.01, 0.1, 1, 10], 'c
lass_weight': [None, 'balanced'], 'penalty': ['l1', '
l2']}
grid_search_log = GridSearchCV(estimator=logmodel,par
am_grid=logmodel_params,scoring='accuracy',cv=10,n_jo
bs=-1)
grid_search_log = grid_search_log.fit(X_train,y_train
)
best_accuracy = grid_search_log.best_score_
print('Accuracy on Cross Validation set :',best_accur
acy)

best_parameters = grid_search_log.best_params_
best_parameters
```

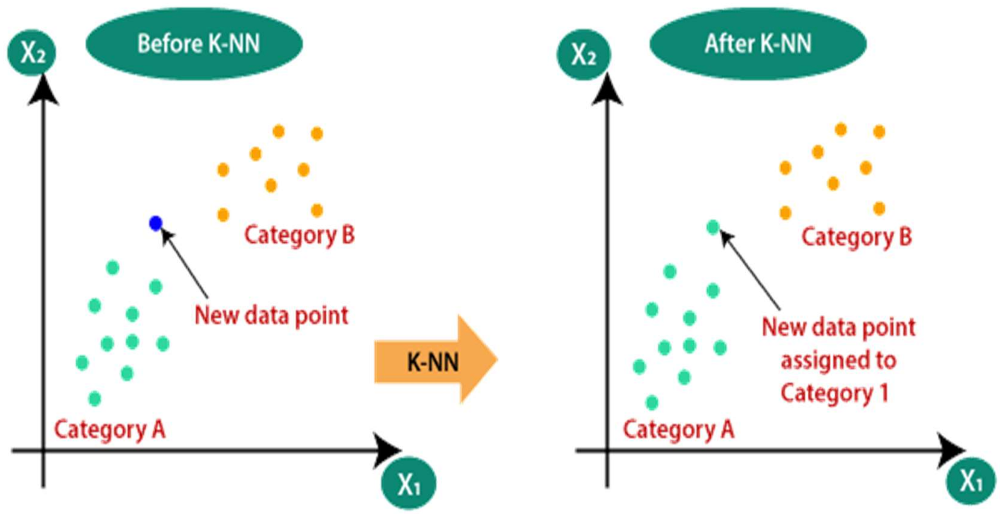
Hyper parameter tuning is an essential part of controlling the behaviour of a machine learning model. If we don't correctly tune our hyperparameters, our estimated model parameters produce suboptimal results, as they don't minimize the loss function. This means our model will makes more errors if not tuned.

KNN (k-nearest neighbors)

```
y_train_pred = classifier.predict(X_train)

from sklearn.metrics import confusion_matrix
from sklearn.metrics import  accuracy_score, f1_score, precision_score, recall_
score, roc_auc_score
roc=roc_auc_score(y_train, y_train_pred)
acc = accuracy_score(y_train, y_train_pred)
prec = precision_score(y_train, y_train_pred)
rec = recall_score(y_train, y_train_pred)
f1 = f1_score(y_train, y_train_pred)

results = pd.DataFrame([['KNN Classifier', acc,prec,rec, f1,roc]],
             columns = ['Model', 'Accuracy', 'Precision', 'Recall', 'F1 Score','ROC'])
results
```

The abbreviation KNN stands for "K-Nearest Neighbour ".

It is a supervised machine learning algorithm. The algorithm can be used to solve both classification and regression problem statements. The number of nearest neighbors to a new unknown variable that has to be predicted or classified is denoted by the symbol 'K'



Y-Train Results

Model	Accuracy	Precision	Recall	F1 Score	ROC
KNN Classifier	0.843238	0.727184	0.478492	0.57719	0.713394

Y-Test Results

Model	Accuracy	Precision	Recall	F1 Score	ROC
KNN Classifier	0.789444	0.5179	0.335567	0.407257	0.624866

```
# Confusion Matrix
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

Classification Report

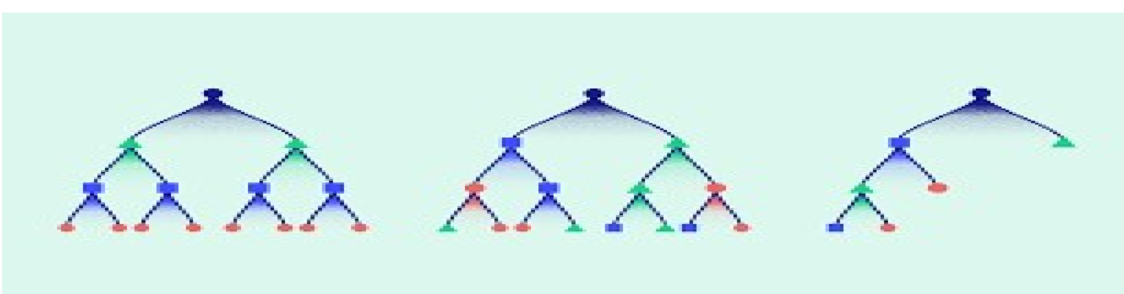
	precision	recall	f1-score	support
0	0.83	0.91	0.87	7060
1	0.52	0.34	0.41	1940
accuracy			0.79	9000
macro avg	0.68	0.62	0.64	9000
weighted avg	0.77	0.79	0.77	9000

Confusion Matrix of KNN

```
[6454  606]
[1289  651]
```

XGBoost Classifier

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It provides a parallel tree boosting to solve many data science problems in a fast and accurate way.



```
# Importing the XGBoost Model
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train, y_train)
```

Y-train result

Model	**Accuracy**	**Precision**	**Recall**	**F1Score**	**ROC**
XGBOOST Classifier	0.8237140	0.696443	0.375213	0.487683	0.664054

Y-test result

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**	**ROC**
XGBOOST Classifier	0.824778	0.676043	0.359278	0.469202	0.655985

 # Confusion Matrix

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data

```
from sklearn.metrics import classification_report

from sklearn.metrics import confusion_matrix

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

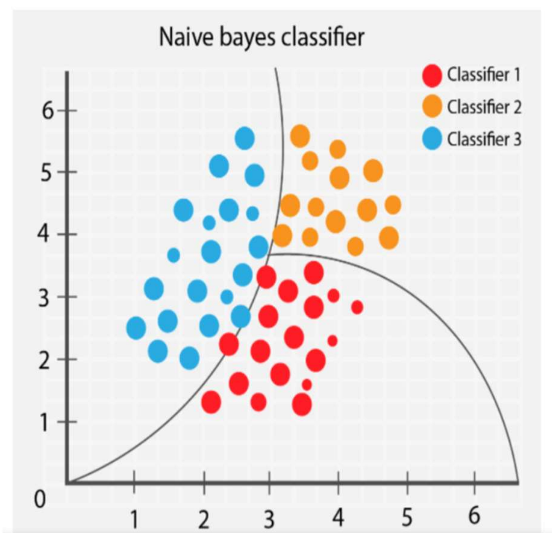
	precision	recall	f1-score	support
0	0.84	0.95	0.90	7060
1	0.68	0.36	0.47	1940
accuracy			0.82	9000

macro avg	0.76	0.66	0.68	9000
weighted avg	0.81	0.82	0.80	9000

```
[[6726  334]
 [1243  697]]
```

Naive Bayes Classifier

Naive Bayes Classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong independence assumptions between the features. They are among the simplest Bayesian network models, but coupled with kernel density estimation, they can achieve high accuracy levels.




```
# Importing the Naive Bayes Model
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train,y_train)
```

Y-train result

Model	Accuracy	Precision	Recall	F1Score	ROC
Gaussian Naive Bayes	0.588571	0.321506	0.756388	0.45122	0.648312

Y-test result

Model	Accuracy	Precision	Recall	F1Score	ROC
Gaussian Naive Bayes	0.584778	0.309276	0.751031	0.43813	0.645062

Confusion Matrix

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data

```
from sklearn.metrics import classification_report

from sklearn.metrics import confusion_matrix

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

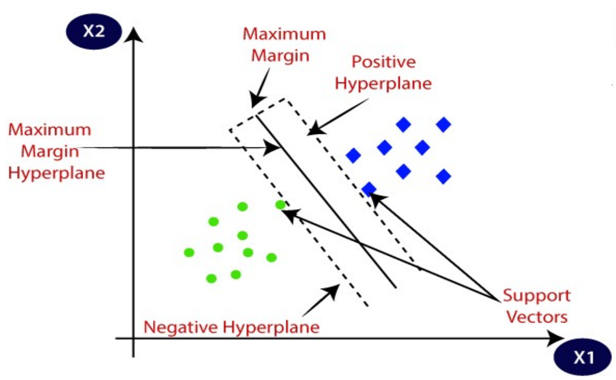
	precision	recall	f1-score	support
	0.89	0.54	0.67	7060
	0.31	0.75	0.44	1940
accuracy			0.58	9000
macro avg	0.60	0.65	0.55	9000

weighted avg 0.76 0.58 0.62 9000

[[3806 3254]
 [483 1457]]

SVM(Support Vector Machine)

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.




```
# import svm model
from sklearn import svm
clf = svm.SVC(kernel='linear') # linear kernel

# train the model using training set
clf.fit(X_train, y_train)
```

Y-train result

Model	Accuracy	Precision	Recall	F1Score	ROC
SVM	0.806524	0.703799	0.232751	0.349816	0.602269

Y-test result

Model	Accuracy	Precision	Recall	F1Score	ROC
SVM	0.815	0.718601	0.23299	0.351888	0.603959

Confusion Matrix

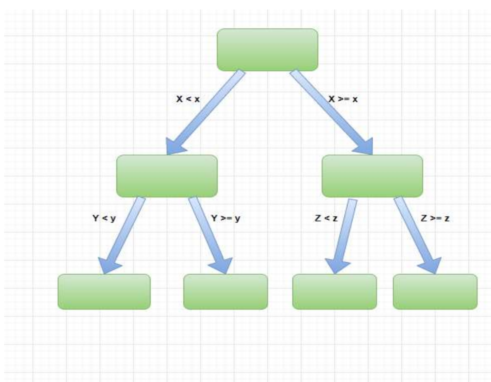
The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.97      0.89      7060
           1       0.72      0.23      0.35      1940

    accuracy                           0.81      9000
   macro avg       0.77      0.60      0.62      9000
weighted avg       0.80      0.81      0.78      9000

[[6883  177]
 [1488  452]]
```

Decision Tree

A decision tree is a graphical representation of possible solutions to a decision based on certain conditions. It's called a decision tree because it starts with a single box (or root), which then branches off into a number of solutions, just like a tree.




```
# import Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
model.fit(X_train, y_train)
```

Y-train result

Model	Accuracy	Precision	Recall	F1Score	ROC
Decision Tree	0.999714	0.999787	0.998935	0.999361	0.999437

Y-test result

Model	Accuracy	Precision	Recall	F1Score	ROC
Decision Tree	0.731111	0.385932	0.418557	0.401583	0.617777

Confusion Matrix

The confusion matrix is a matrix used to determine the performance of the classification models for a given set of test data

```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score  support

           0       0.84      0.82      0.83      7060
           1       0.39      0.42      0.40      1940

    accuracy                           0.73      9000
   macro avg       0.61      0.62      0.61      9000
weighted avg       0.74      0.73      0.73      9000

[[5768 1292]
 [1128  812]]
```

Decision Tree Hypertuning

```
params = {
    "criterion" : ["gini", "entropy"],
```

```
    "max_depth" : [1,2,3,4,5,6,7,None],
    "splitter":['best','random'],
    "class_weight" : ['balanced',None],
    'max_depth':[2,4,6,8,10],
    'min_samples_leaf':[2,4,6,8,10],
    'min_samples_split':[2,4,6,8,10]
}


from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(model,param_grid=params, cv=10, n
_jobs=-1)

best_accuracy_1 = grid.best_score_
print('Accuracy on Cross Validation set :',best_accur
acy_1)
best_parameters_2 = grid.best_params_
best_parameters_2
y_pred_dct = grid.predict(X_test)
roc=roc_auc_score(y_test, y_pred_dct)
acc = accuracy_score(y_test, y_pred_dct)
prec = precision_score(y_test, y_pred_dct)
rec = recall_score(y_test, y_pred_dct)
f1 = f1_score(y_test, y_pred_dct)
model =  pd.DataFrame([['Decision Tree Tuned', acc,pr
ec,rec, f1,roc]],
               columns = ['Model', 'Accuracy', 'Preci
sion', 'Recall', 'F1 Score','ROC'])
model_results = model_results.append(model, ignore_in
dex = True)
model_results
```

Model	Accuracy	Precision	Recall	F1Score	ROC
Decision Tree Tuned	0.825889	0.686687	0.353608	0.46685	0.654637

Conclusion:

- XGBoost s able to predict 82% accuracy, followed by logistic classifier and svm
- Marrid, more educated credit ard users, whose age is bet 28 to 40 are likely to default on their payments.
- Single men less educated whose age are less than 28 or more than 40 are likely to default on payments.