Classification Project on Credit Default Prediction

T. Sai Harish Sarma

Contents

Problem Statement:

This project is aimed at predicting the customer's default payments in Taiwan. From the perspective of the 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 non credible clients. We can use the K-S Chart to evaluate which customers will default on their credit card payments.

Data Description

This dataset consists of 30000 rows and 25 columns which 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.

Data Description

- 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 (-2= No Consumption, -1=pay duly, 0= paid minimum & revolving credit, 1=payment delay for one month, 2=payment delay for two 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)

Data Description

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

Data Description

```
data.shape
```

```
(30000, 25)
```

```
data.isnull().sum()
```

```
LIMIT_BAL            0
SEX                  0
AGE                  0
PAY_SEP              0
PAY_AUG              0
PAY_JUL              0
PAY_JUN              0
PAY_MAY              0
PAY_APR              0
BILL_AMT_SEP         0
BILL_AMT_AUG         0
BILL_AMT_JUL         0
BILL_AMT_JUN         0
BILL_AMT_MAY         0
BILL_AMT_APR         0
PAY_AMT_SEP          0
PAY_AMT_AUG          0
PAY_AMT_JUL          0
PAY_AMT_JUN          0
PAY_AMT_MAY          0
PAY_AMT_APR          0
DEF_PAY_NEXT_MONTH   0
MARRIAGE_2           0
MARRIAGE_3           0
EDUCATION_2          0
EDUCATION_3          0
EDUCATION_4          0
dtype: int64
```

Data Handling

MARRIAGE: Marital status (1=married, 2=single, 3=others)

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

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

```
[13] # Merge 0 class to 3 class i.e., others.

data['MARRIAGE']=data['MARRIAGE'].replace({0:3})
data['MARRIAGE'].value_counts()
```

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

EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)

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

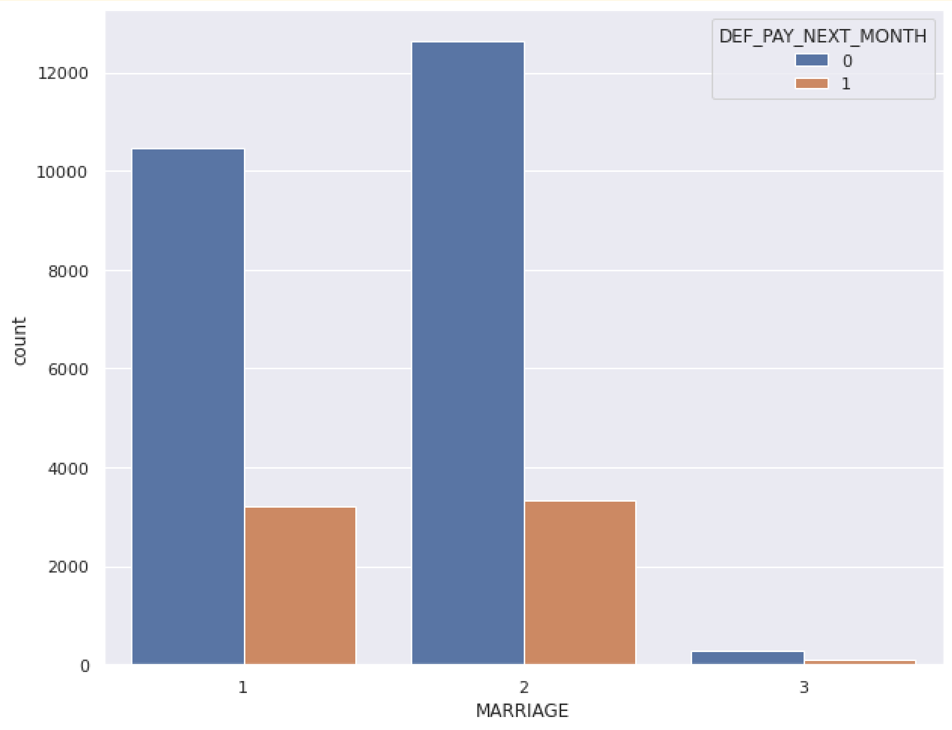
```
2    14030
1    10585
3     4917
5      280
4      123
6       51
0       14
Name: EDUCATION, dtype: int64
```

```
[ ] # Merge 5,6,0 classes to 4 class i.e., others
data['EDUCATION']=data['EDUCATION'].replace({5:4,6:4,0:4})
data['EDUCATION'].value_counts()
```

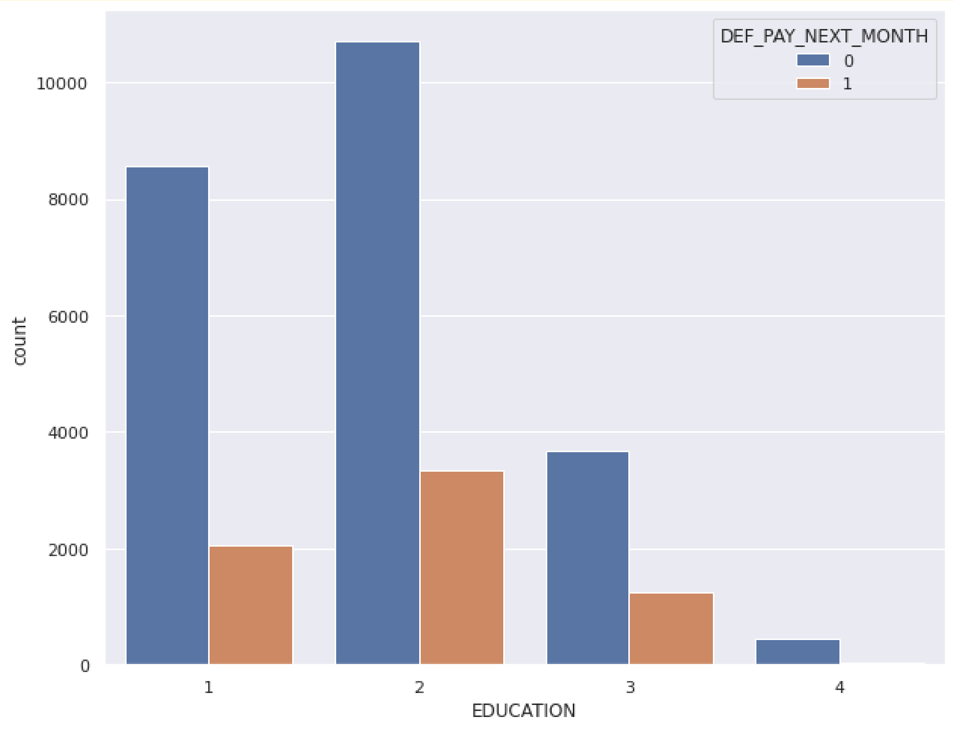
```
2    14030
1    10585
3     4917
4      468
Name: EDUCATION, dtype: int64
```

Exploratory Data Analysis

MARRIAGE: Marital status (1=married, 2=single, 3=others)



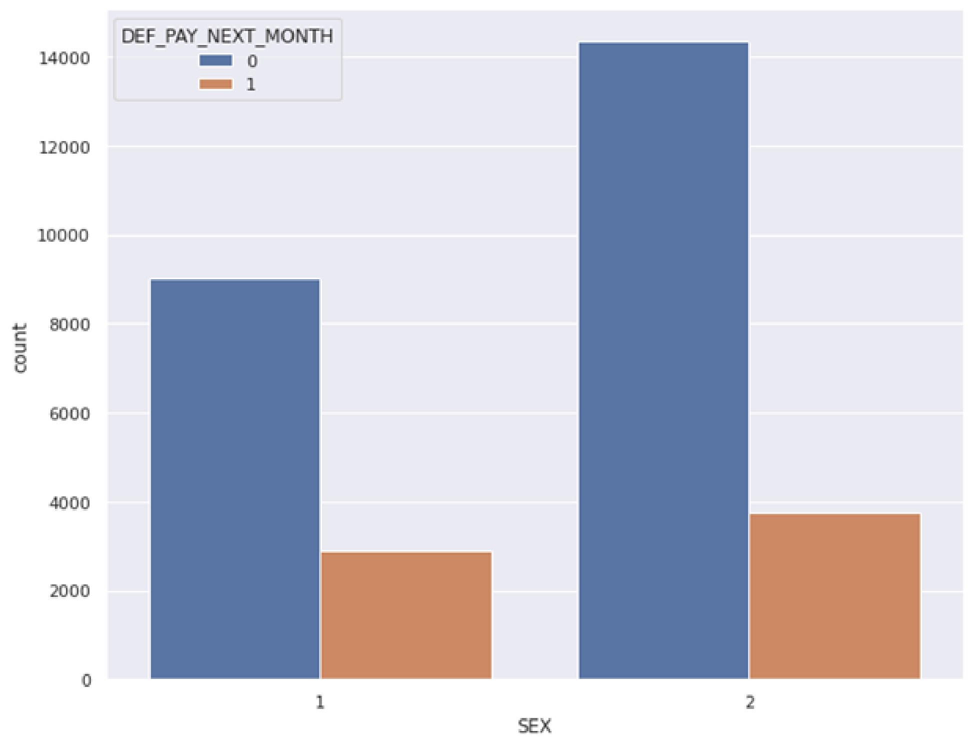
EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others)



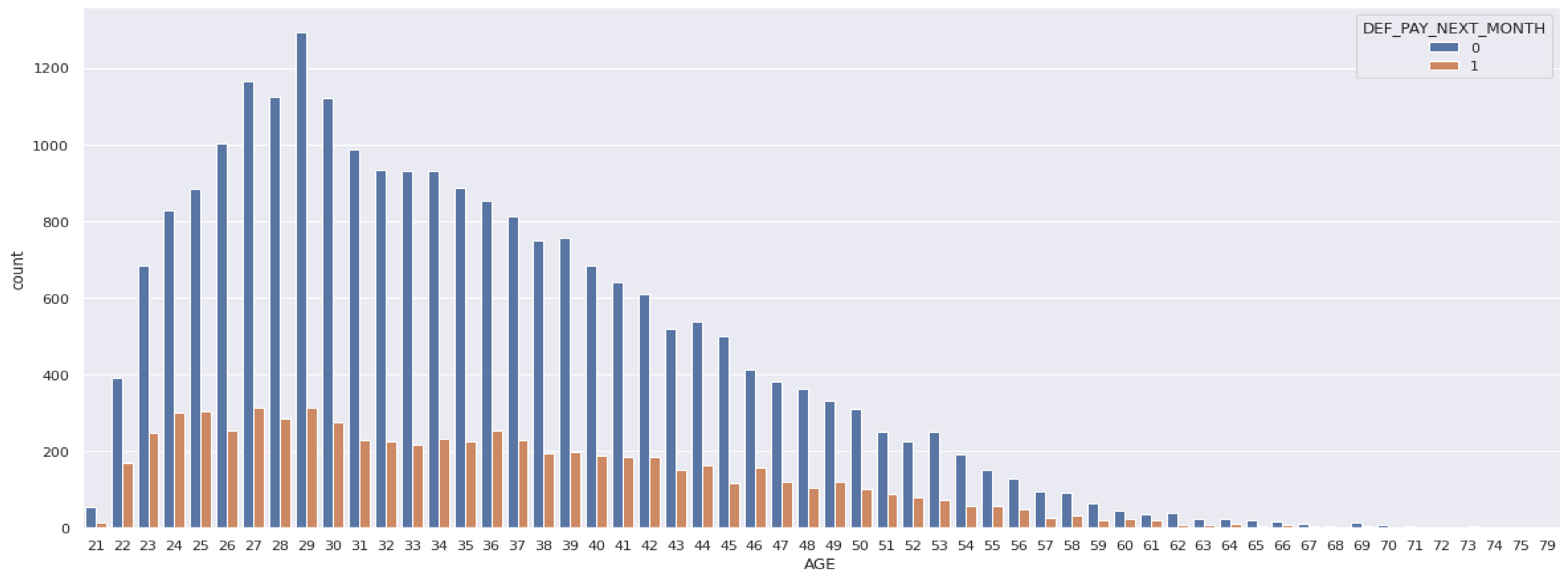
Exploratory Data Analysis

- SEX: Gender (1=male, 2=female)
- Female = 18112
- Male = 11888

- Female count is higher than Male in – taking credit card.

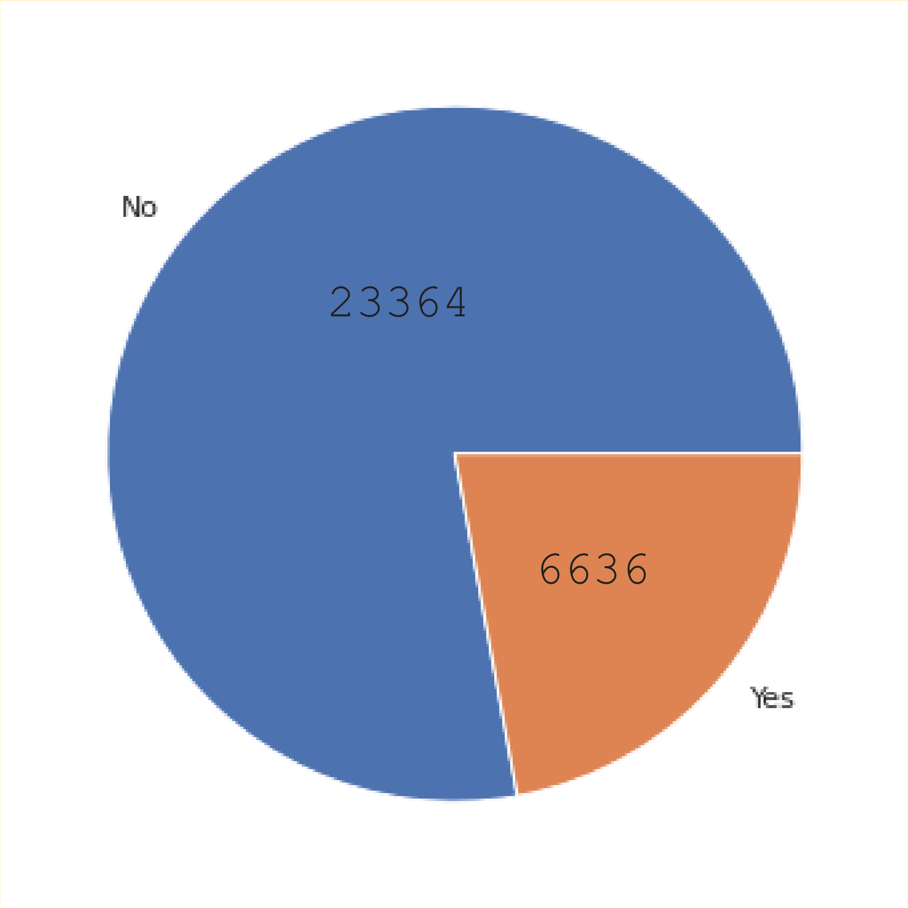


Exploratory Data Analysis

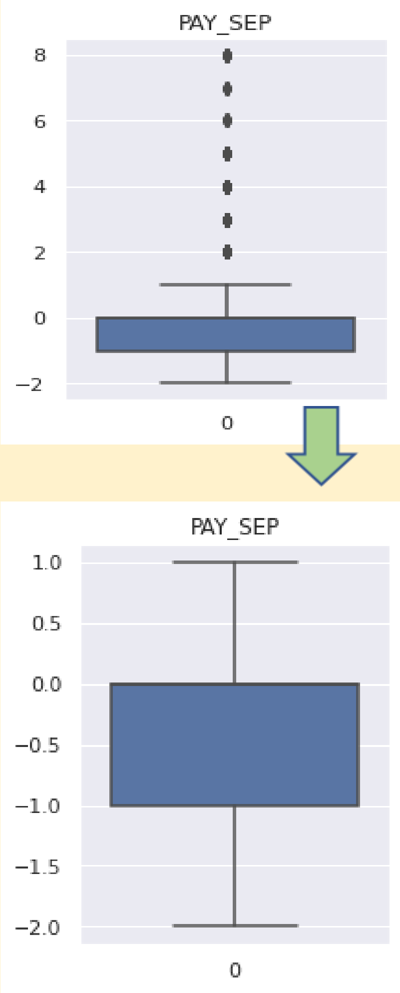


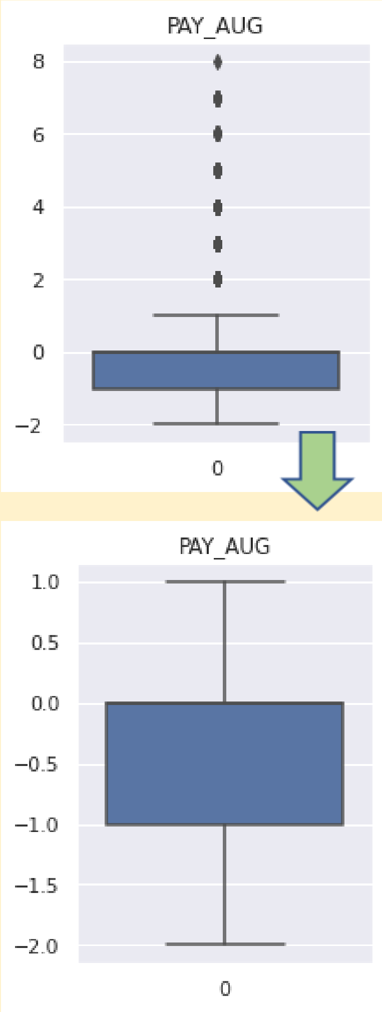
Exploratory Data Analysis

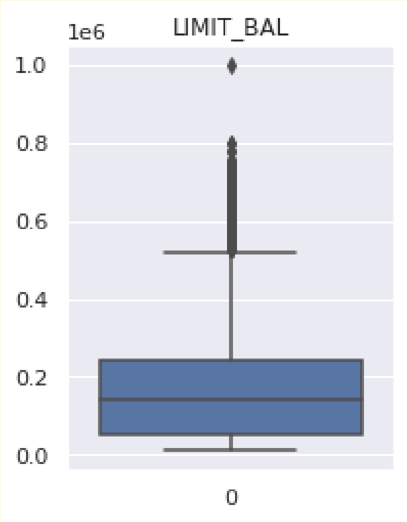
77.88 % people are not the defaulters of credit card. 22.12 % (6636 out of 30000) people are the defaulters.



Handling Outliers







Using Z Score

Handling Imbalance of Data

```
# Creating Input Variable as x:
x= data.drop(['DEF_PAY_NEXT_MONTH'], axis=1)
```

```
# Creating Target Variable as y:
y=data['DEF_PAY_NEXT_MONTH']
```

```
y.value_counts()
```

```
0    18629
1     2712
Name: DEF_PAY_NEXT_MONTH, dtype: int64
```

If the ouput classes in dataset are not in proportion, That will result poor in evaluating metrics of the model. For this purpose, we use SMOTE(Synthetic Minority Oversampling Technique).

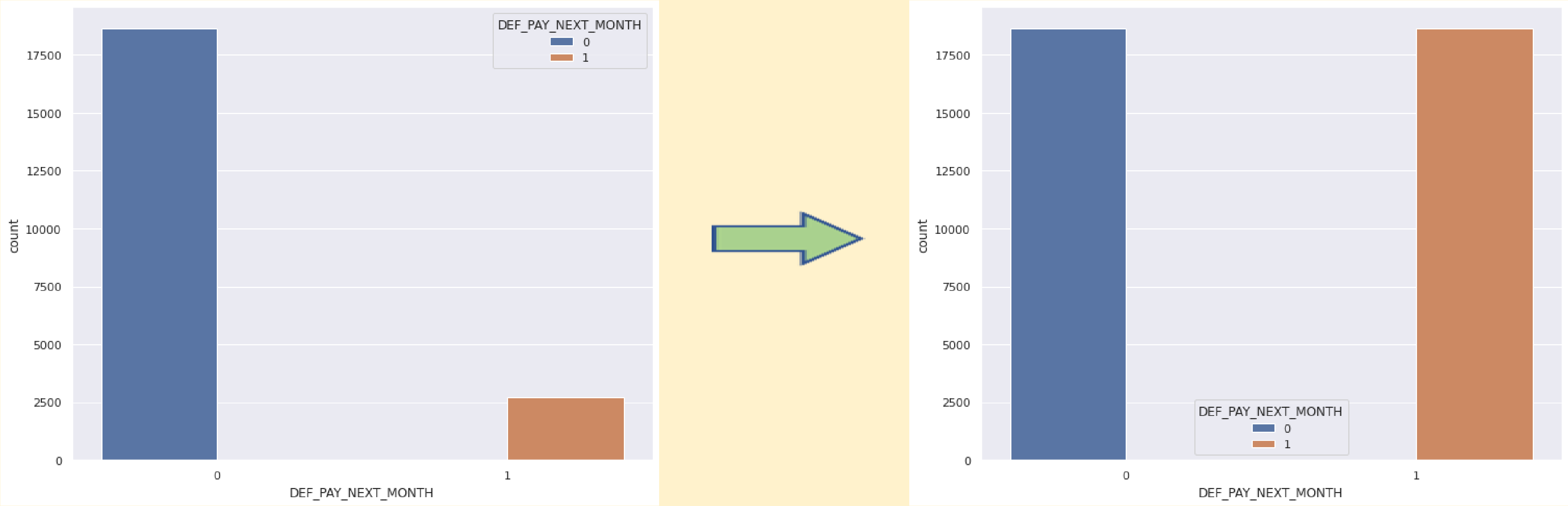
```
# Using Module SMOTE
smote =SMOTE()
```

```
# Fit the SMOTE() to the x,y data
x_smote, y_smote = SMOTE().fit_resample(x,y)
```

```
#Verify y variable
y_smote.value_counts()
```

```
0    18629
1    18629
Name: DEF_PAY_NEXT_MONTH, dtype: int64
```

Handling Imbalance of Data



Feature Scaling

```
[60] from sklearn.preprocessing import StandardScaler
```

```
[61] # Renaming SMOTE Variables for Modelling as X as input and Y as output
     X= x_smote
     Y= y_smote
```

```
[62] scaler=StandardScaler()
```

```
[63] scaler.fit_transform(X)
```

```
array([[-0.68428018,  0.99491336, -0.17024104, ...,  1.32137052,
        -0.32975302, -0.10587313],
       [-0.99857214,  0.99491336,  0.19564282, ...,  1.32137052,
        -0.32975302, -0.10587313],
       [-0.99857214, -1.00511265,  2.63486851, ...,  1.32137052,
        -0.32975302, -0.10587313],
       ...,
       [-0.68428018,  0.99491336, -1.2678926 , ..., -0.75679   ,
        -0.32975302, -0.10587313],
       [ 0.18002271, -1.00511265, -0.41416361, ..., -0.75679   ,
        -0.32975302, -0.10587313],
       [-1.15571812, -1.00511265,  0.43956539, ...,  1.32137052,
        -0.32975302, -0.10587313]])
```

Creating Training and Testing Data Set

Creating Train-Test-Split

```
from sklearn.model_selection import train_test_split
```

```
[65] X_train,X_test,Y_train,Y_test =train_test_split(X,Y, test_size=0.25, random_state=42)
```

```
[66] X_train.shape ,X_test.shape, Y_train.shape ,Y_test.shape
```

```
((27943, 26), (9315, 26), (27943,), (9315,))
```

Parameter Tuning & Implementation

Tuning the Parameters

```
[ ] #Creating Model Estimator for Tuning
    Model2= SVC()
```

```
[ ] #Mentioning Logistic Regression parameters for Tuning
    parameters= ({'C':[0.1,1.0]})
```

```
[ ] Model2_Grid= GridSearchCV(Model2, parameters,n_jobs=1,scoring='precision',verbose=3)
```

```
[ ] #Model2_Grid.fit(X_train,Y_train)
```

```
Fitting 5 folds for each of 2 candidates, totalling 10 fits
[CV 1/5] END ............................C=0.1;, score=0.578 total time=  42.9s
[CV 2/5] END ............................C=0.1;, score=0.577 total time=  42.8s
[CV 3/5] END ............................C=0.1;, score=0.566 total time=  43.2s
[CV 4/5] END ............................C=0.1;, score=0.578 total time=  44.1s
[CV 5/5] END ............................C=0.1;, score=0.581 total time=  43.2s
[CV 1/5] END ............................C=1.0;, score=0.595 total time=  41.6s
[CV 2/5] END ............................C=1.0;, score=0.593 total time=  41.6s
[CV 3/5] END ............................C=1.0;, score=0.587 total time=  41.7s
[CV 4/5] END ............................C=1.0;, score=0.593 total time=  41.3s
[CV 5/5] END ............................C=1.0;, score=0.593 total time=  42.8s
▸ GridSearchCV
▸ estimator: SVC
  ▸ SVC
```

```
[ ] Model2_Grid.best_params_
```

```
{'C': 1.0}
```

```
Model2_Grid.best_score_
```

```
0.5921928629458251
```

```
# Implementing Optimal Parameters in Model
# Model2_Grid.best_score is 0.5921928629458251
#{'C': 1.0}

Model2= SVC(C=1.0)
Model2.fit(X_train,Y_train)
```

```
▾ SVC
SVC()
```

```
[ ] # Predicting the model with Training and Testing Data
    Y_train_predict=Model2.predict(X_train)
    Y_test_predict=Model2.predict(X_test)
```

Model Evaluation Metrics

- Train Accuracy
- Test Accuracy
- Recall Score
- Precision Score
- AUC ROC Score
- F1 Score

Model Evaluation Metrics

Evaluating the Metrics of the model.

```
# Printing Evaluation Metrics

Accuracy_Score =accuracy_score(Y_test_predict,Y_test)
print("Accuracy_Score: ",Accuracy_Score)

Recall_Score =recall_score(Y_test_predict,Y_test)
print("Recall_Score: ",Recall_Score)

Precision_Score =precision_score(Y_test_predict,Y_test)
print("Precision_Score: ",Precision_Score)

AUC_ROC_Score =roc_auc_score(Y_test_predict,Y_test)
print("AUC_ROC_Score: ",AUC_ROC_Score)

F1_Score =f1_score(Y_test_predict,Y_test)
print("F1_Score :",F1_Score)
```

```
Accuracy_Score:  0.5755233494363929
Recall_Score:  0.6078832116788321
Precision_Score:  0.44363946303004476
AUC_ROC_Score:  0.5822947467562242
F1_Score : 0.5129342202512934
```

```
Eval_DF=pd.DataFrame(Dict, index=[1])
```

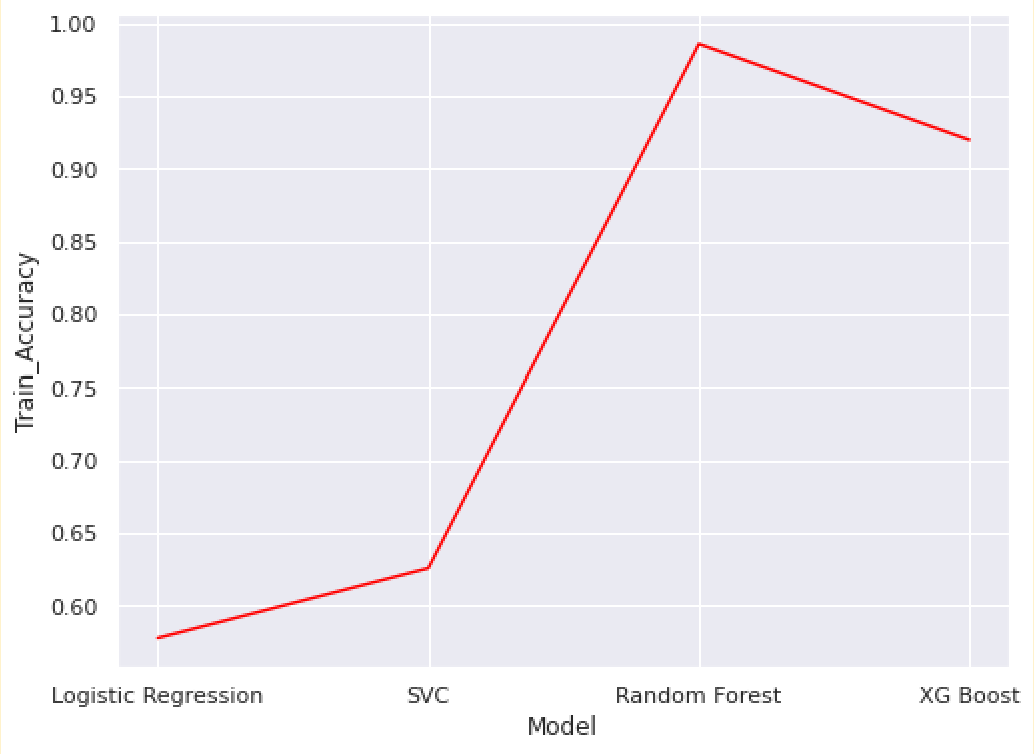
```
Eval_DF
```

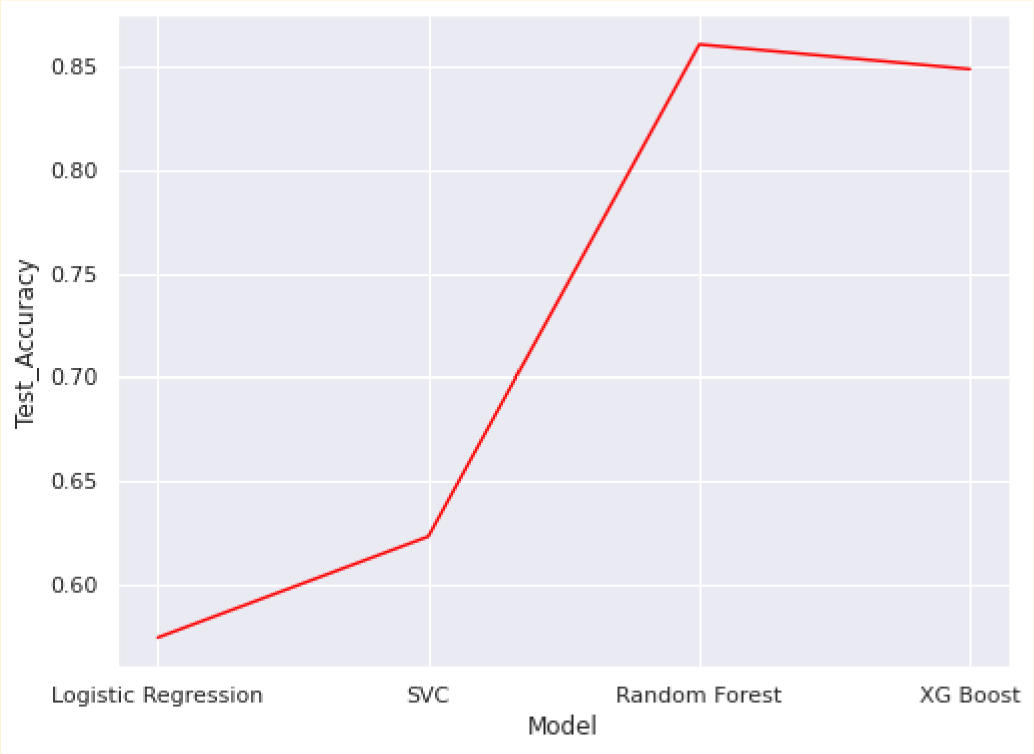
	Model	Accuracy_Score	Recall_Score	Precision_Score	AUC_ROC_Score	F1_Score
1	Logistic Regression	0.576	0.608	0.444	0.582	0.513

Comparing Metrics

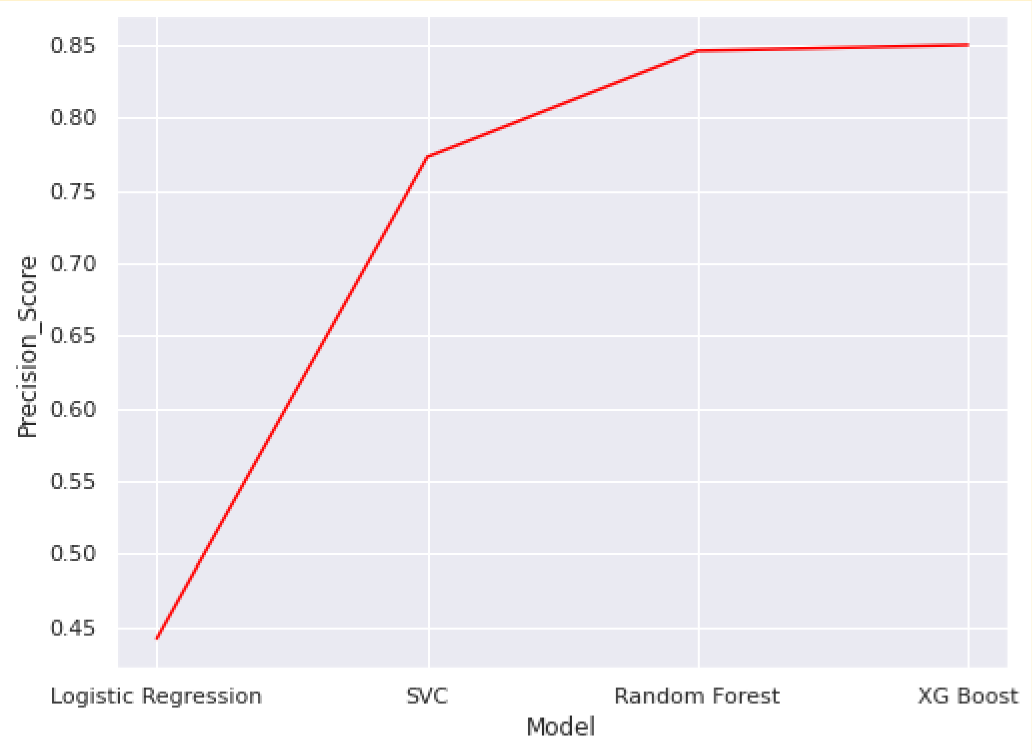
	Model	Train_Accuracy	Test_Accuracy	Recall_Score	Precision_Score	AUC_ROC_Score	F1_Score
0	Logistic Regression	0.575	0.578	0.615	0.437	0.586	0.511
1	SVC	0.624	0.620	0.593	0.786	0.634	0.676
2	Random Forest	0.988	0.869	0.879	0.859	0.869	0.869
3	XG Boost	0.919	0.845	0.847	0.845	0.845	0.846

Comparing Metrics



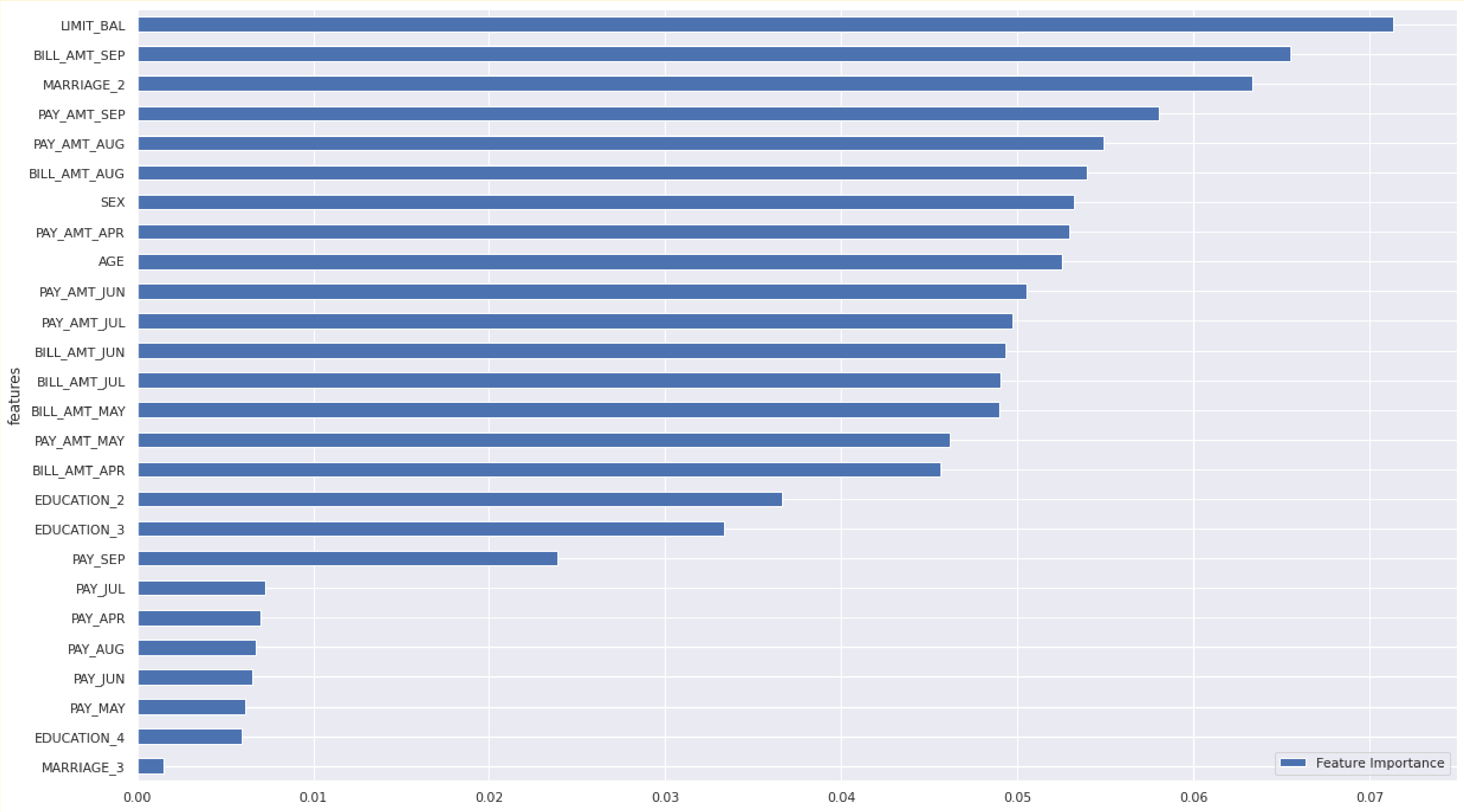


Comparing Metrics

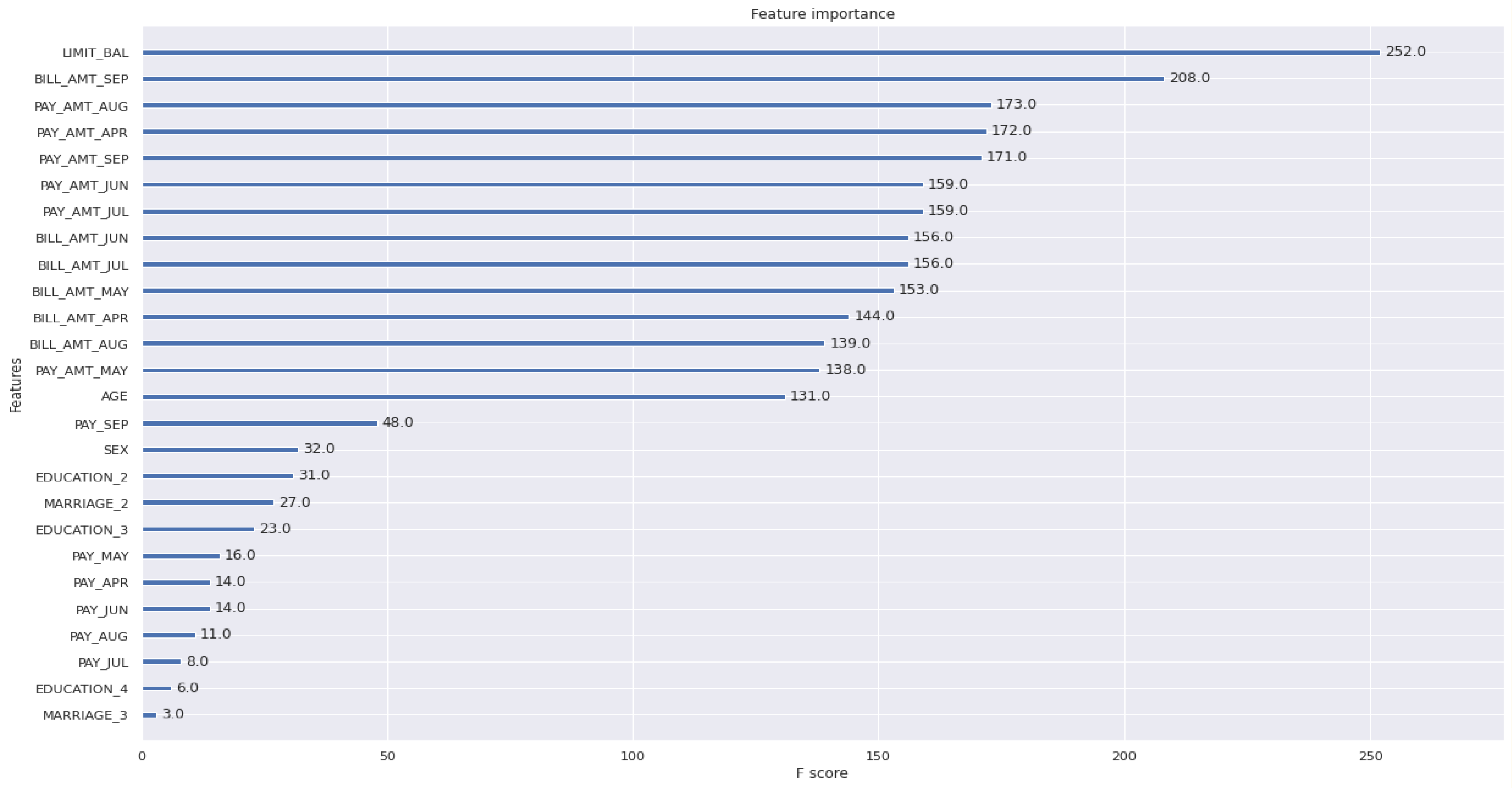


Comparing Metrics

Feature Importance – Random Forest



Feature Importance – XG Boost



Conclusion

EDA Outcomes:

- People who are not married taking credits slightly higher than the married people
- Age of most using og credit is in the range of 24 and 40. After the age 60, almost there is a declinebusing credit.
- 77.88 % people are not the defaulters of credit card. 22.12 % (6636 out of 30000) people are the defaulters.
- Female count is higher than Male in taking credit card.

Challenges:

- Outliers detected by Boxplot and Treated
- Imbalance of Data treated by SMOTE Approach
- Some Data not having proper explanation
- Removed Irrelavent classes in columns.

Conclusion

Model Outcomes:

- Among all the models, Random Forest Model given good results.
- Random Forest Classifier & XG Boost given the best precision Score i.e. 85.9% & 84.5% respectively.
- Random Forest Classifier given the best recall Score i.e. 87.9%
- The Columns 'LIMIT_BAL','BILL_AMT_SEP' & 'PAY_AMT_AUG' are the most important features for our Target Feature i.e., to check whether the user Defaulter or not.
- In Overall, Random Forest Classifier best to fit on this data, But Execution of CrossValidation using GridSearchCV for hyper parameters takes long time.

Thank you