Capstone Project
Credit Card Default Prediction

Name- Pranav Singhal
Batch-AlmaBetter Pro

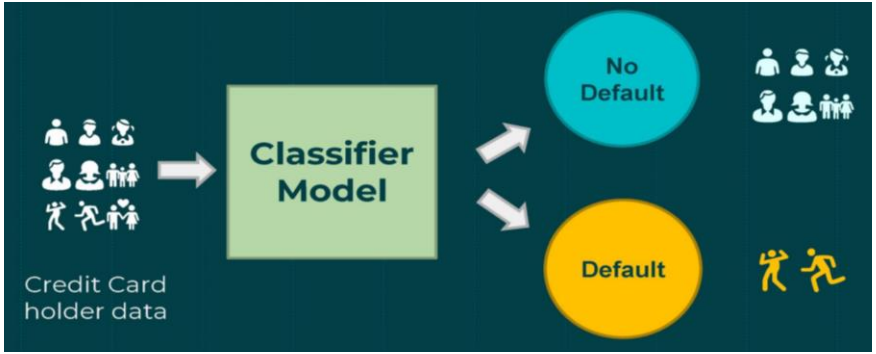
Github- https://github.com/pranav168/Credit-Card-Default-Prediction

Why Credit Card Default Risk Prediction?

Default risk is the chance that companies or individuals will be unable to make the required payments on their debt obligations. In other words, credit default risk is the probability that if you lend money, there is a chance that they won't be able to give the money back on time. Lenders and investors are exposed to default risk in virtually all forms of credit extensions. To mitigate the impact of default risk, lenders often impose charges that correspond to the debtor's level of default risk. A higher level of risk leads to a higher required return.

Predicting Credit Card Default Risk with Machine Learning

- Developments in machine learning and deep learning have made it much easier for companies and individuals to build a high-performance credit default risk prediction model for their own use.



- Knowing about machine learning, and classification problems, in particular, it is quite evident that the credit card default risk prediction problem is nothing but a binary classification problem. So any machine learning method that could be used for binary classification problems can be applied to credit default risk prediction problems as well.

The success of a machine learning model

The success of a machine learning model, however, does not depend solely on the selection of a machine learning method. Key factors contributing to the success of the machine learning model include:

- **Data**

Data is the very prerequisite for any successful machine learning model. No matter how great your machine learning models are, you cannot get a reliable high-performance model from the prediction model without a sufficient amount of rich data.

- **Feature Engineering**

Processing raw data and making it a suitable input for the machine learning models includes **data cleaning, creating new features, and feature selection**. Feature engineering usually is the most time-consuming machine learning problem, especially when it comes to building prediction models for structured data.

- **Models**

Even though there are many machine learning methods available for certain machine learning problems, such as binary classification, for example, each method has its own strengths and weaknesses. Based on our demands and requirements, we may need to choose different methods.

- **Performance Metrics**

Given two machine learning methods, how do we evaluate them to select the better one?

- We need well-designed performance metrics based on our dataset and experience. For example, AUC and F1 Score are typically used for unbalanced data and binary classification problems.
- We can use the K-S chart to evaluate which customers will default on their credit card payments

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

Objective

Predicting whether a customer will default on his/her credit card

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: Limit_bal

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.

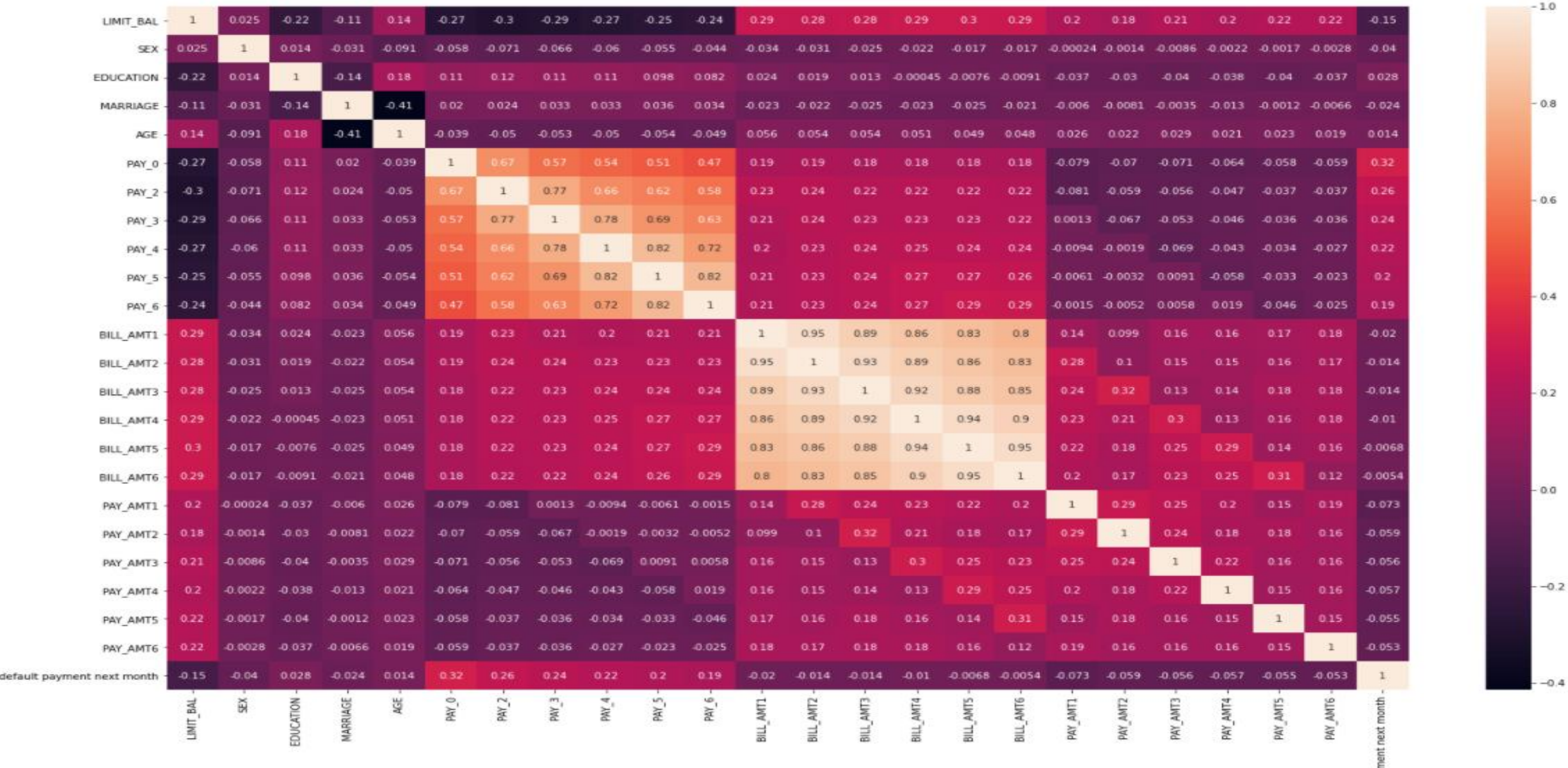
EDA(Count Dataset)

	LIMIT_BAL	LIMIT_BAL_count	SEX	SEX_count	EDUCATION	EDUCATION_count	MARRIAGE	MARRIAGE_count	AGE	AGE_count	PAY_0	PAY_0_count	PAY_2	PAY_2_count	PAY_3	PAY_3_count	PAY_4	PAY_4_count	PAY_5	PAY_5_count	PAY_6	PAY_6_count
0	50000	3365	2.000000	18112.000000	2.000000	14030.000000	2.000000	15964.000000	29	1605	0	14737	0	15730	0	15764	0	16455	0	16947	0	16286
1	20000	1976	1.000000	11888.000000	1.000000	10585.000000	1.000000	13659.000000	27	1477	-1	5686	-1	6050	-1	5938	-1	5687	-1	5539	-1	5740
2	30000	1610			3.000000	4917.000000	3.000000	323.000000	28	1409	1	3688	2	3927	-2	4085	-2	4348	-2	4546	-2	4895
3	80000	1567			5.000000	280.000000	0.000000	54.000000	30	1395	-2	2759	-2	3782	2	3819	2	3159	2	2626	2	2766
4	200000	1528			4.000000	123.000000			26	1256	2	2667	3	326	3	240	3	180	3	178	3	184
5	150000	1110			6.000000	51.000000			31	1217	3	322	4	99	4	76	4	69	4	84	4	49
6	100000	1048			0.000000	14.000000			25	1186	4	76	1	28	7	27	7	58	7	58	7	46
7	180000	995							34	1162	5	26	5	25	6	23	5	35	5	17	6	19
8	360000	881							32	1158	8	19	7	20	5	21	6	5	6	4	5	13
9	60000	825							33	1146	6	11	6	12	1	4	1	2	8	1	8	2

BILL_AMT1	BILL_AMT1_count	BILL_AMT2	BILL_AMT2_count	BILL_AMT3	BILL_AMT3_count	BILL_AMT4	BILL_AMT4_count	BILL_AMT5	BILL_AMT5_count	BILL_AMT6	BILL_AMT6_count	PAY_AMT1	PAY_AMT1_count	PAY_AMT2	PAY_AMT2_count	PAY_AMT3	PAY_AMT3_
0	2008	0	2506	0	2870	0	3195	0	3506	0	4020	0	5249	0	5396	0	5968
390	244	390	231	390	275	390	246	390	235	390	207	2000	1363	2000	1290	2000	1285
780	76	326	75	780	74	780	101	780	94	780	86	3000	891	3000	857	1000	1103
326	72	780	75	326	63	316	68	316	79	150	78	5000	698	5000	717	3000	870
316	63	316	72	316	62	326	62	326	62	316	77	1500	507	1000	594	5000	721
2500	59	2500	51	396	48	396	44	150	58	326	56	4000	426	1500	521	1500	490
396	49	396	51	2500	40	2400	39	396	47	396	45	10000	401	4000	410	4000	381
2400	39	2400	42	2400	39	150	39	2400	39	416	36	1000	365	10000	318	10000	312
416	29	-200	29	416	29	2500	34	2500	37	-18	33	2500	298	6000	283	1200	243
500	25	416	28	200	27	1000	33	416	36	2400	32	6000	294	2500	251	6000	241

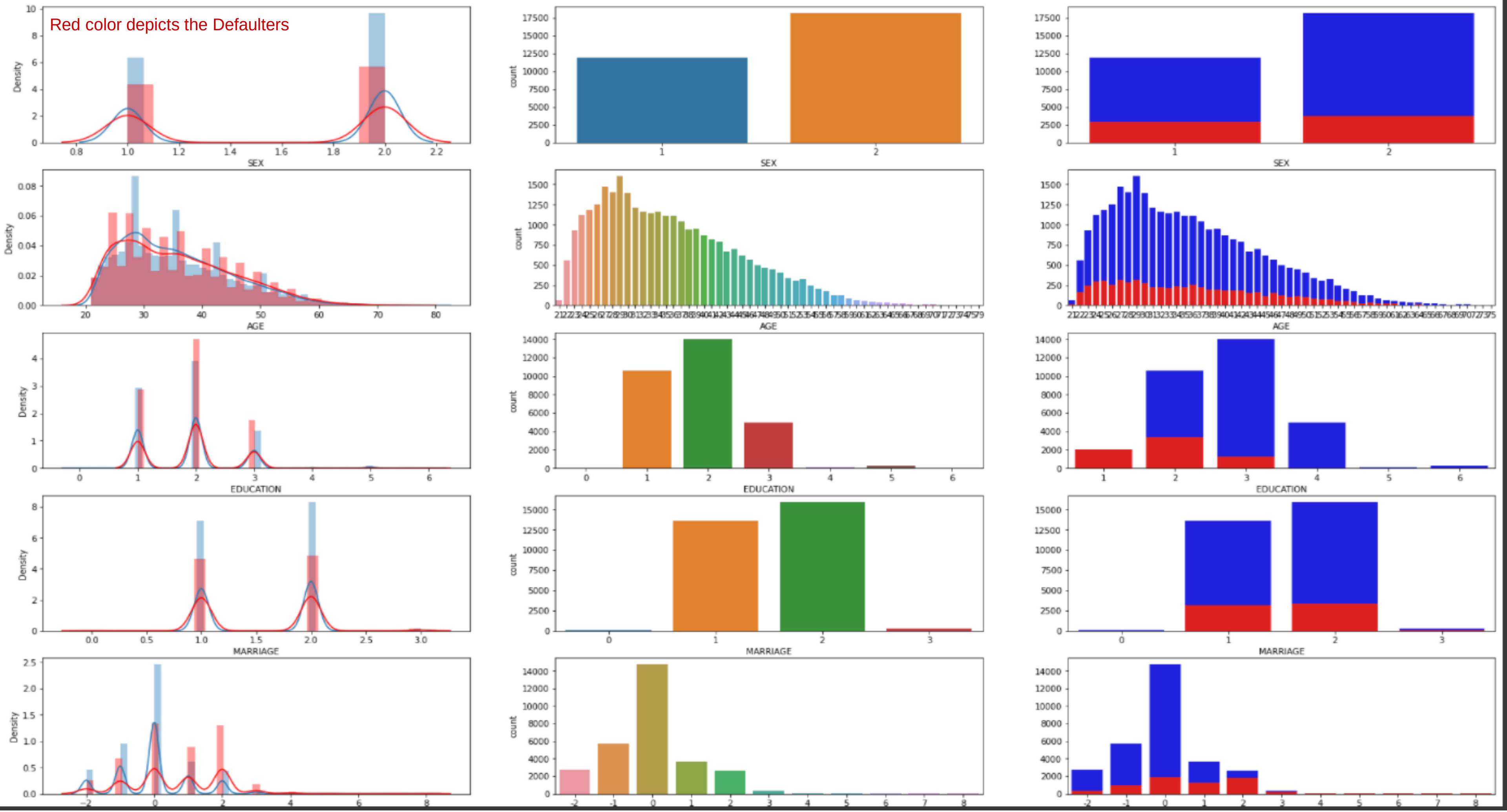
PAY_AMT4	PAY_AMT4_count	PAY_AMT5	PAY_AMT5_count	PAY_AMT6	PAY_AMT6_count	default payment next month	default payment next month_count
0	6408	0	6703	0	7173	0.000000	23364.000000
1000	1394	1000	1340	1000	1299	1.000000	6636.000000
2000	1214	2000	1323	2000	1295		
3000	887	3000	947	3000	914		
5000	810	5000	814	5000	808		
1500	441	1500	426	1500	439		
4000	402	4000	401	4000	411		
10000	341	10000	343	10000	356		

This Dataset set gives count of each and every Variable in the Original Dataset

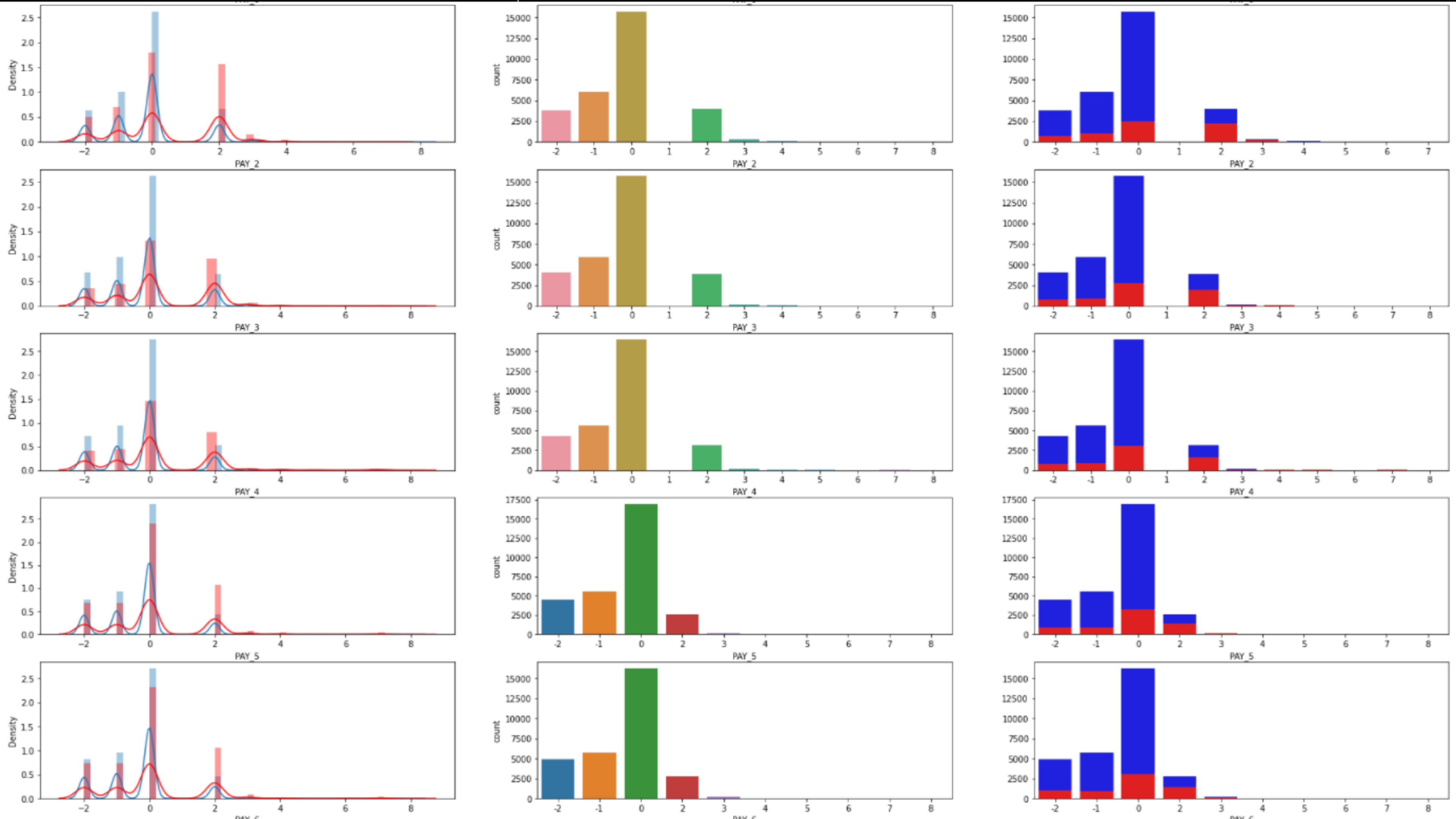


Correlation Chart

Correlation is high for Bill_amt's which is not great and will be taken care later, also relation between our dependent variable is very low with all other variables thus we should limit ourselves from reducing the features straight away

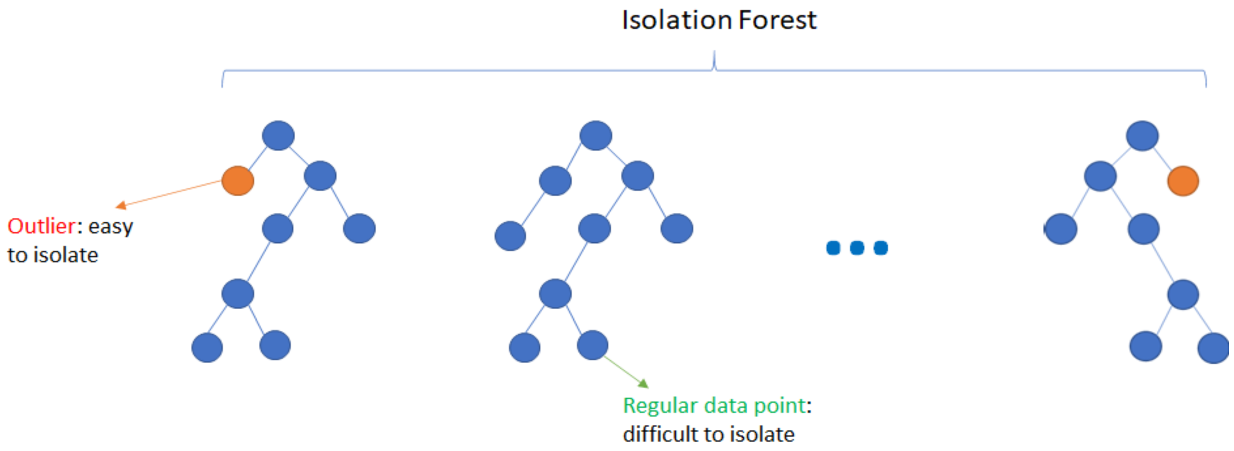


Distribution plot | count plot | comparison plot



Density
count
PAY_2
PAY_3
PAY_4
PAY_5

Outlier Detection with Isolation Forest



Lets do a Anomaly detection now, the most correlated feature with our label is PAY_0 and similar others which actually is quite logical for this prediction and thus we will use this relation to find anomalies. (here anomalies signifies to person who are very punctual but have been defaulter because of certain unavoidable reasons and vice-versa)

One Hot Encoding

```
one_hot_entity=['Pay_september','Pay_august','Pay_july','Pay_june','Pay_may','Pay_april']
column_one_hot=['PAY_0','PAY_2','PAY_3','PAY_4','PAY_5','PAY_6']        # one hot encoding
count=0
for i in column_one_hot:
  temp_df=pd.get_dummies(df[i], prefix=one_hot_entity[count])
  count+=1
  try:
    df_one_hot=pd.concat([df_one_hot, temp_df], axis=1)
  except:
    df_one_hot=temp_df
df_one_hot.head()
```

Train/Test Split

Segregating Data into Train and Test sets, so that we could check out model predictions in the later half.

```
X = df.drop(["default payment next month"],axis =1 )                    #making Final Datasets
y = df["default payment next month"]

X_train,X_test,y_train,y_test = train_test_split(X,y, test_size =0.2,random_state=0) #train test split
```

X have our independent variables and y have Dependent Variable.

No. of observation in X_train, y_train = 24000

No. of observation in X_test, y_test =6000

Decision Tree, Random Forest and XGBoost

Now our Data Looks good for Tree Based Model, I will evaluate the model without much pre-processing which will do later for other models.

Best Results For Decision Tree :

```
              precision    recall  f1-score   support

Non-Defaulter       0.85      0.96      0.90      4692
    Defaulter       0.69      0.35      0.46      1248

     accuracy                           0.83      5940
    macro avg       0.77      0.65      0.68      5940
 weighted avg       0.81      0.83      0.81      5940
```

```
At max_depth =5: MSE = 0.1696969696969697, Accuracy Score for Train
Samples =82.53% and Accuracy Score for Test Samples =83.03%
```

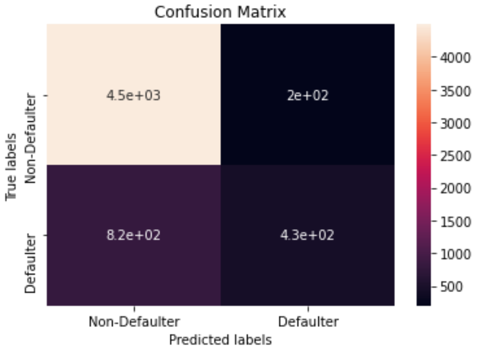
Though our Accuracy is really high and MSE is Quite low, we can still say that our model does not do a great job in predicting defaulters as F1 score is really low

```
              precision    recall  f1-score   support

Non-Defaulter       0.85      0.95      0.90      4692
    Defaulter       0.68      0.36      0.47      1248

     accuracy                           0.83      5940
    macro avg       0.76      0.66      0.68      5940
 weighted avg       0.81      0.83      0.81      5940

At max_depth =3: MSE = 0.1712121212121212, Accuracy Score
              precision    recall  f1-score   support

Non-Defaulter       0.85      0.96      0.90      4692
    Defaulter       0.69      0.35      0.46      1248

     accuracy                           0.83      5940
    macro avg       0.77      0.65      0.68      5940
 weighted avg       0.81      0.83      0.81      5940

At max_depth =5: MSE = 0.1696969696969697, Accuracy Score
              precision    recall  f1-score   support

Non-Defaulter       0.85      0.95      0.90      4692
    Defaulter       0.67      0.35      0.46      1248

     accuracy                           0.83      5940
    macro avg       0.76      0.65      0.68      5940
 weighted avg       0.81      0.83      0.81      5940

At max_depth =7: MSE = 0.1727272727272727273, Accuracy Scor
              precision    recall  f1-score   support

Non-Defaulter       0.84      0.94      0.89      4692
    Defaulter       0.61      0.34      0.44      1248

     accuracy                           0.82      5940
    macro avg       0.73      0.64      0.66      5940
 weighted avg       0.80      0.82      0.80      5940

At max_depth =10: MSE = 0.1835016835016835, Accuracy Scor
              precision    recall  f1-score   support

Non-Defaulter       0.85      0.93      0.89      4692
    Defaulter       0.59      0.37      0.46      1248

     accuracy                           0.81      5940
    macro avg       0.72      0.65      0.67      5940
 weighted avg       0.79      0.81      0.80      5940

At max_depth =12: MSE = 0.1868686868686868688, Accuracy Sco
```

Hyper-parameter Tuning using cv

Best Results For XGBoost :

	precision	recall	f1-score	support
Non-Defaulter	0.85	0.96	0.90	4692
Defaulter	0.69	0.35	0.46	1248
accuracy			0.83	5940
macro avg	0.77	0.65	0.68	5940
weighted avg	0.81	0.83	0.81	5940




```
Best: -0.426602 using {'max_depth': 4, 'n_estimators': 100}
-0.544205 (0.002054) with: {'max_depth': 3, 'n_estimators': 5}
-0.482497 (0.003140) with: {'max_depth': 3, 'n_estimators': 10}
-0.442822 (0.004158) with: {'max_depth': 3, 'n_estimators': 20}
-0.428882 (0.004587) with: {'max_depth': 3, 'n_estimators': 50}
-0.427049 (0.005064) with: {'max_depth': 3, 'n_estimators': 100}
-0.541969 (0.002020) with: {'max_depth': 4, 'n_estimators': 5}
-0.479739 (0.003046) with: {'max_depth': 4, 'n_estimators': 10}
-0.439675 (0.004157) with: {'max_depth': 4, 'n_estimators': 20}
-0.427632 (0.004712) with: {'max_depth': 4, 'n_estimators': 50}
-0.426602 (0.005422) with: {'max_depth': 4, 'n_estimators': 100}
-0.541126 (0.002145) with: {'max_depth': 5, 'n_estimators': 5}
-0.478558 (0.003377) with: {'max_depth': 5, 'n_estimators': 10}
-0.438647 (0.004575) with: {'max_depth': 5, 'n_estimators': 20}
-0.427745 (0.005246) with: {'max_depth': 5, 'n_estimators': 50}
-0.428058 (0.005907) with: {'max_depth': 5, 'n_estimators': 100}
-0.540289 (0.002274) with: {'max_depth': 6, 'n_estimators': 5}
-0.477615 (0.003787) with: {'max_depth': 6, 'n_estimators': 10}
-0.437873 (0.005261) with: {'max_depth': 6, 'n_estimators': 20}
-0.428091 (0.006307) with: {'max_depth': 6, 'n_estimators': 50}
-0.428854 (0.006973) with: {'max_depth': 6, 'n_estimators': 100}
-0.539877 (0.002411) with: {'max_depth': 7, 'n_estimators': 5}
-0.477425 (0.003849) with: {'max_depth': 7, 'n_estimators': 10}
-0.437978 (0.005521) with: {'max_depth': 7, 'n_estimators': 20}
-0.429575 (0.006884) with: {'max_depth': 7, 'n_estimators': 50}
-0.432272 (0.007414) with: {'max_depth': 7, 'n_estimators': 100}
```

Hyper-parameter Tuning using cv

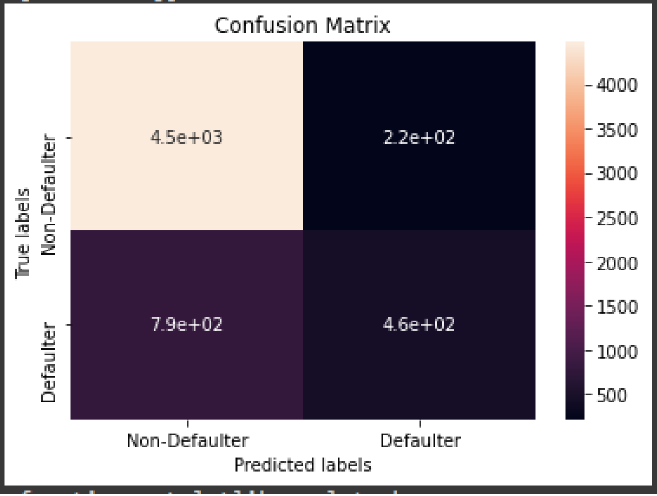
MSE = 0.168181, Accuracy Score for Train Samples =0.829 and Accuracy Score for Test Samples =0.832

Again very similar thing is happening here as well, Accuracy is really high and MSE is Quite low, we can say that our model does not do a great job in predicting defaulters as F1 score is really low, thereby making the problem is quite evident.

Yes, our Model is suffering because of class imbalance, there is a lot of information about Non-defaulter Data and really less for Defaulters thus we will improve on this later by generating synthetic data using SMOTE.

Best Results For Random Forest:

```
           0       0.85      0.95      0.90      4692
           1       0.68      0.36      0.47      1248

    accuracy                           0.83      5940
   macro avg       0.76      0.66      0.69      5940
weighted avg       0.81      0.83      0.81      5940

[[4473  219]
 [ 793  455]]
```



SE = 0.17, Accuracy Score for Train Samples =0.999242 and Accuracy Score for Test Samples =0.828for Random Forest Classifier

Processing our data for Logistic Regression, SVM and preparing dataset

- Adding all the Bill_amt's which are highly co-related to form a single entity

- I will not be doing the same with Pay columns because our Independent variable is not co-related with any of our dependent variables thus we should try to retain the maximum features

```
df['Total_BILL_AMT']=df.BILL_AMT1+df.BILL_AMT2+df.BILL_AMT3+df.BILL_AMT4+df.BILL_AMT5+df.BILL_AMT6
df.drop(df[['BILL_AMT1','BILL_AMT2','BILL_AMT3','BILL_AMT4','BILL_AMT5','BILL_AMT6']], axis=1,inplace=True)
```

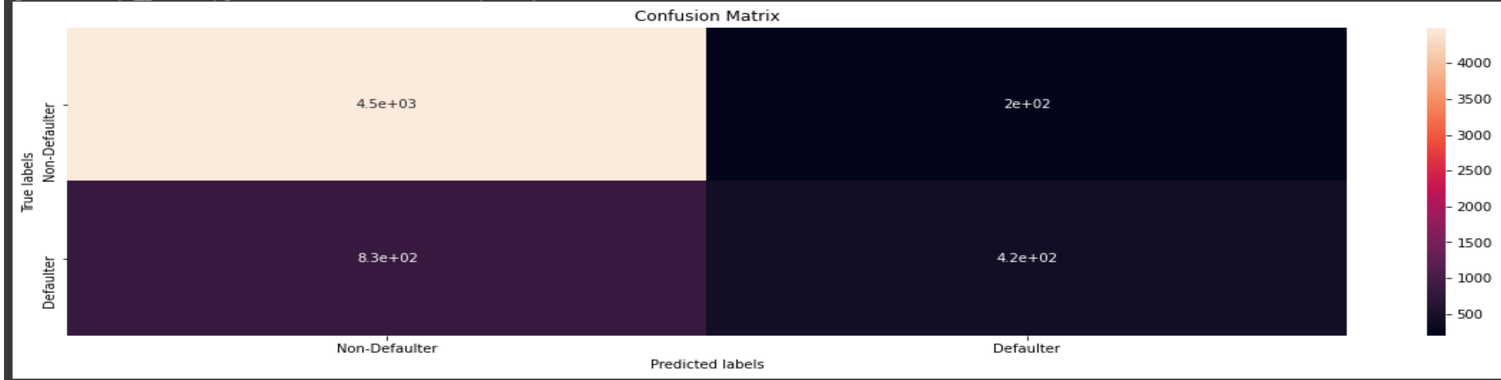
```
df.head()
```

```
one_hot_entity2=['SEX','EDUCATION','MARRIAGE','AGE']
column_one_hot=['SEX','EDUCATION','MARRIAGE','AGE']
count=0
for i in column_one_hot:
  temp_df2=pd.get_dummies(df[i], prefix=one_hot_entity2[count])
  count+=1
  try:
    df_one_hot2=pd.concat([df_one_hot2, temp_df2], axis=1)
  except:
    df_one_hot2=temp_df2
df_one_hot2.head()
```

Logistic, SVM, Neural Network Models

Best Result from Logistic Model:

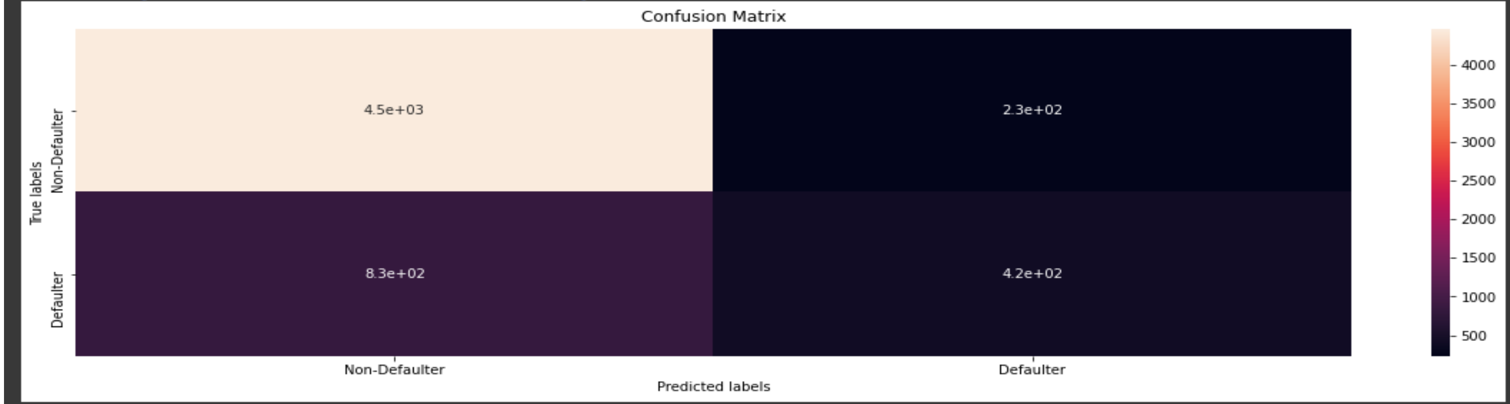
```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[CV]  ................................................................
[CV] ............ , accuracy=(train=0.823, test=0.820), total=   0.9s
[CV]  ................................................................
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.9s remaining:    0.0s
[CV] ............ , accuracy=(train=0.822, test=0.825), total=   1.2s
[CV]  ................................................................
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    2.2s remaining:    0.0s
[CV] ............ , accuracy=(train=0.822, test=0.822), total=   1.0s
[CV]  ................................................................
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    3.2s remaining:    0.0s
[CV] ............ , accuracy=(train=0.823, test=0.821), total=   1.2s
[CV]  ................................................................
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    4.4s remaining:    0.0s
[CV] ............ , accuracy=(train=0.824, test=0.817), total=   1.1s
The accuracy on train data is 82.24% and The accuracy on test data is 82.64% for model Logistic Model
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    5.5s finished
```




```
              precision    recall  f1-score   support

Non-Defaulter       0.84      0.96      0.90      4692
    Defaulter       0.68      0.33      0.45      1248

     accuracy                           0.83      5940
    macro avg       0.76      0.65      0.67      5940
 weighted avg       0.81      0.83      0.80      5940
```

Best Result from Neural Network Model:

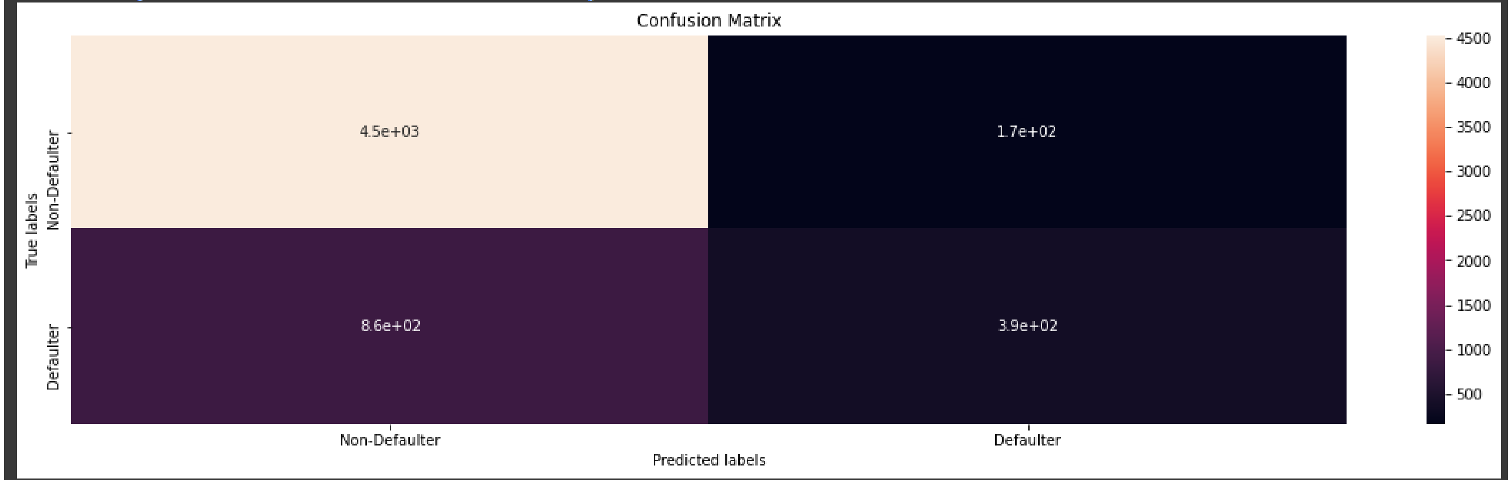
```
Epoch 1/10
1188/1188 [==============================] - 6s 2ms/step - loss: 0.4540 - accuracy: 0.8099 - val_loss: 0.4365 - val_accuracy: 0.8209
Epoch 2/10
1188/1188 [==============================] - 2s 2ms/step - loss: 0.4353 - accuracy: 0.8225 - val_loss: 0.4380 - val_accuracy: 0.8203
Epoch 3/10
1188/1188 [==============================] - 2s 2ms/step - loss: 0.4322 - accuracy: 0.8247 - val_loss: 0.4340 - val_accuracy: 0.8213
Epoch 4/10
1188/1188 [==============================] - 2s 2ms/step - loss: 0.4289 - accuracy: 0.8246 - val_loss: 0.4360 - val_accuracy: 0.8205
Epoch 5/10
1188/1188 [==============================] - 2s 2ms/step - loss: 0.4256 - accuracy: 0.8263 - val_loss: 0.4362 - val_accuracy: 0.8201
Epoch 6/10
1188/1188 [==============================] - 2s 2ms/step - loss: 0.4233 - accuracy: 0.8275 - val_loss: 0.4414 - val_accuracy: 0.8213
Epoch 7/10
1188/1188 [==============================] - 2s 2ms/step - loss: 0.4194 - accuracy: 0.8293 - val_loss: 0.4394 - val_accuracy: 0.8220
Epoch 8/10
1188/1188 [==============================] - 2s 2ms/step - loss: 0.4160 - accuracy: 0.8303 - val_loss: 0.4416 - val_accuracy: 0.8211
Epoch 9/10
1188/1188 [==============================] - 2s 2ms/step - loss: 0.4121 - accuracy: 0.8329 - val_loss: 0.4439 - val_accuracy: 0.8188
Epoch 10/10
1188/1188 [==============================] - 2s 2ms/step - loss: 0.4091 - accuracy: 0.8333 - val_loss: 0.4480 - val_accuracy: 0.8192
The accuracy on train data is 83.4% and The accuracy on test data is 82.17% for model Neural Network
```




```
               precision    recall  f1-score   support

Non-Defaulter       0.84      0.95      0.89      4692
    Defaulter       0.65      0.34      0.44      1248

     accuracy                           0.82      5940
    macro avg       0.74      0.64      0.67      5940
 weighted avg       0.80      0.82      0.80      5940
```

Best Result from SVM Model:

```
The accuracy on train data is 82.16% and The accuracy on test data is 82.69% for model SVM
```

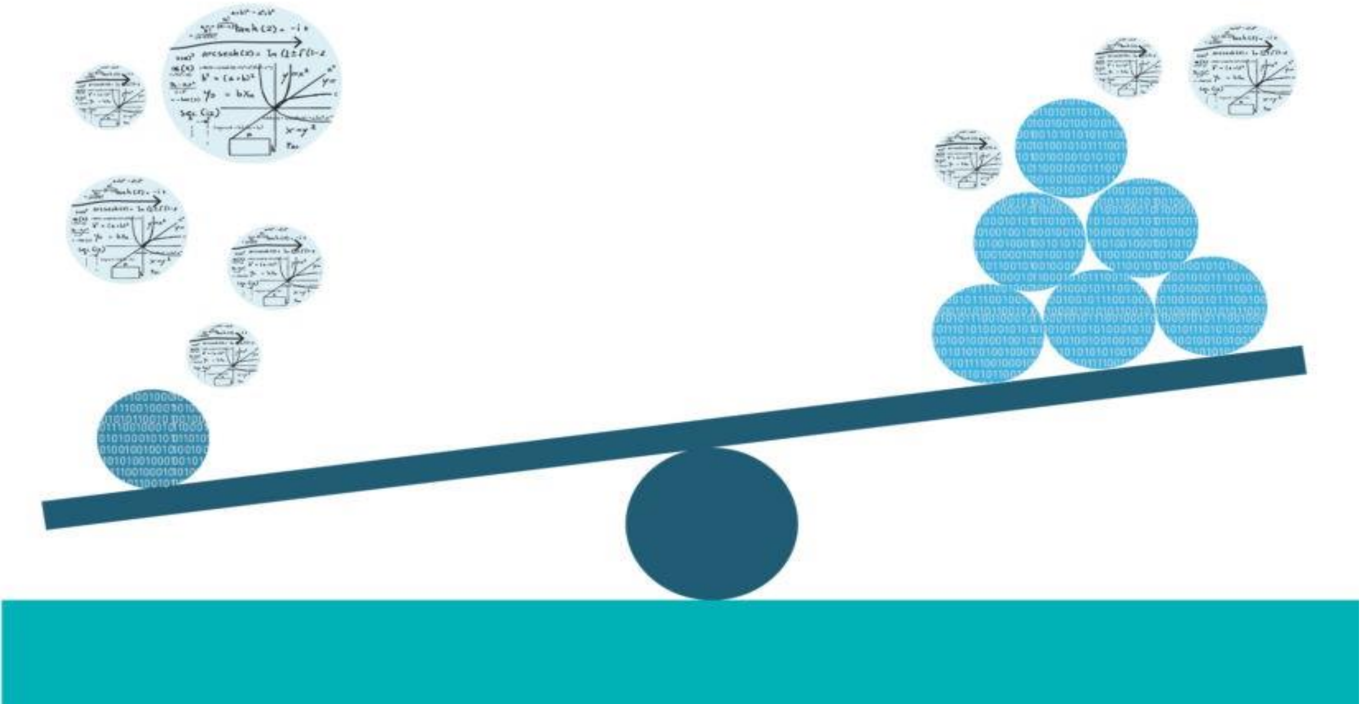


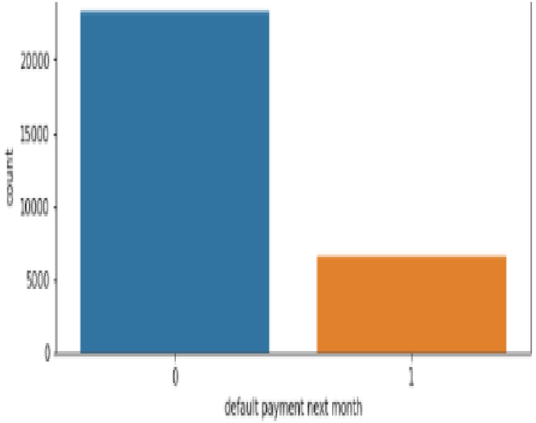
	precision	recall	f1-score	support
Non-Defaulter	0.84	0.96	0.90	4692
Defaulter	0.70	0.31	0.43	1248
accuracy			0.83	5940
macro avg	0.77	0.64	0.66	5940
weighted avg	0.81	0.83	0.80	5940

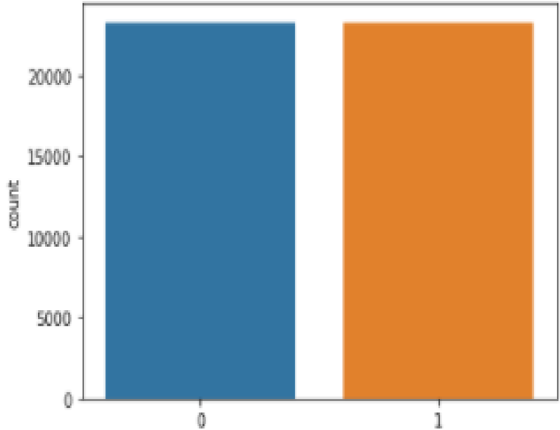
Every model is suffering from class imbalance, lets get rid of it now!!

SMOTE for Imbalanced Classification

Synthetic Minority Oversampling Technique



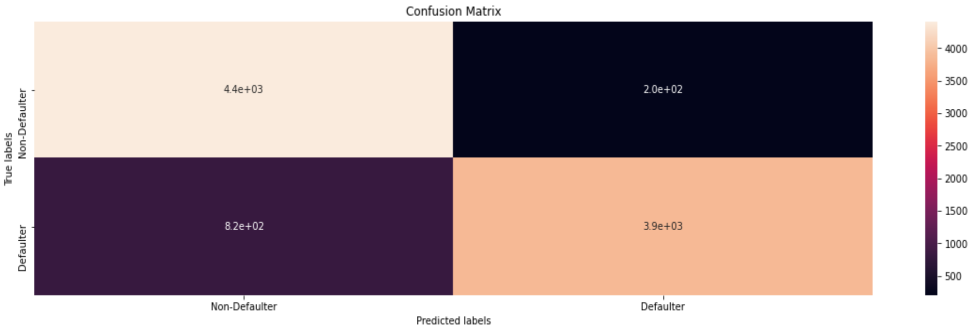




Best results after SMOTE

Logistic Model:

	precision	recall	f1-score	support
Non-Defaulter	0.84	0.96	0.90	4613
Defaulter	0.95	0.82	0.88	4689
accuracy			0.89	9302
macro avg	0.90	0.89	0.89	9302
weighted avg	0.90	0.89	0.89	9302



The accuracy on train data is 88.61% and The accuracy on test data is 88.94% for model Logistic Model

Now, f1 score is 89% thus our model perform really vey well and could be deployed for the prediction of defaulters.

Performance chart Before SMOTE

Model	Auc %		F1 score	
	Train	Test	Non-Defaulter	Defaulter
DecisionTreeClassifier	82.42	82.78	0.9	0.44
XGBClassifier	83.0	82.9	0.9	0.47
RandomForestClassifier	99.9	82.3	0.9	0.47
LogisticRegression	82.3	82.64	0.9	0.46
Neural Network	83.6	82.19	0.9	0.44
SVM	82.14	82.69	0.9	0.43

Performance chart After SMOTE

Model	Auc %		F1 score	
	Train	Test	Non-Defaulter	Defaulter
DecisionTreeClassifier	86.36	82.49	0.83	0.82
XGBClassifier	97.6	87.3	0.88	0.87
RandomForestClassifier	99.9	87.5	0.88	0.87
LogisticRegression	88.61	88.94	0.9	0.88
Neural Network	88.5	88.01	0.88	0.88
SVM	88.58	88.79	0.90	0.88

What is KS Statistics?

It stands for Kolmogorov–Smirnov which is named after Andrey Kolmogorov and Nikolai Smirnov. It compares the two cumulative distributions and returns the maximum difference between them. It is a non-parametric test which means you don't need to test any assumption related to the distribution of data. In KS Test, Null hypothesis states null both cumulative distributions are similar. Rejecting the null hypothesis means cumulative distributions are different.

In data science, it compares the cumulative distribution of events and non-events and KS is where there is a maximum difference between the two distributions. In simple words, it helps us to understand how well our predictive model is able to discriminate between events and non-events.

KS analysis for our Model

Decile	min_prob	max_prob	events	nonevents	event_rate	nonevent_rate
1	0.999997	1.000000	931	0	22.87%	0.00%
2	0.999941	0.999997	930	0	22.85%	0.00%
3	0.998595	0.999940	929	1	22.83%	0.02%
4	0.935319	0.998587	536	394	13.17%	7.53%
5	0.893799	0.935260	35	895	0.86%	17.11%
6	0.868299	0.893789	34	896	0.84%	17.13%
7	0.845118	0.868280	46	884	1.13%	16.90%
8	0.809964	0.845072	65	865	1.60%	16.53%
9	0.715946	0.809885	148	782	3.64%	14.95%
10	0.500182	0.715765	416	515	10.22%	9.84%

Decile	cum_eventrate	cum_noneventrate	KS
1	22.87%	0.00%	22.9
2	45.72%	0.00%	45.7
3	68.55%	0.02%	68.5
4	81.72%	7.55%	74.2
5	82.58%	24.66%	57.9
6	83.42%	41.78%	41.6
7	84.55%	58.68%	25.9
8	86.14%	75.21%	10.9
9	89.78%	90.16%	-0.4
10	100.00%	100.00%	0.0

KS is 74.2% at decile 4

Really Great!! we can definately differentiate between the both classes our Final Model will be the most Linear Regression Model as it surpasses every other model despite to being the most basic model

Conclusion

After building Various models to achieve our object, I conclude that all the tree based model works really good even with less data and class imbalance, but once number of training observation is increased and class imbalance is reduced, other classification models outshine them as accuracy growth in Tree model remains stagnant when compared with other.

Our Final logistic model has 89% accuracy as well as f1 score with a great score of 74.2% at decile 4 on Ks analysis, anything above 50% is considered good, thus we can say that our model could really classify between the Defaulter and Non-Defaulters with a huge margin.

THANK YOU