Predicting Credit Card Application status Using Machine Learning

Group Members-

Naiwrita Chowdhury, Guru Nanak Institute of Technology, 171430110146 (2017-2018)

Pritimoy Sarkar, Guru Nanak Institute of Technology, 171430110066 (2017-2018)

Sandipan Sau, Guru Nanak Institute of Technology, 171430110081 (2017-2018)

Snigdha Mazumdar, Guru Nanak Institute of Technology, 171430110095 (2017-2018)

Snehandu Chanda, Guru Nanak Institute of Technology, 171430110094 (2017-2018)

Triyasha Kundu, Guru Nanak Institute of Technology, 171430110121 (2017-2018)

Table of Contents

Acknowledgement

To perform this whole project, we had to take help of respected persons who deserve our true greatest gratitude. The completion of the project brought us great pleasure and we would like to show our special thanks and gratitude to our Mr. Kaushik Ghosh, our 'Machine Learning using Python' course instructor who gave us a golden opportunity to do this wonderful project on "Credit Card Approval Prediction by Machine Learning using Python". This project took us a lot of research and we came to know so many new things and we are really thankful for this giving us this opportunity.

We would also like to thanks 'Globsyn' institute for giving us this great opportunity to come across this project while our 'Machine Learning using Python' training period.

Many people, especially our team members who paid their hard work and attention and made valuable suggestions to implement things in this project that have inspired us to improve our project. We would like to thank to all those classmates and all others for their direct and indirect helping to complete our project.

Group Members: -

Naiwrita Chowdhury

Pritimoy Sarkar

Sandipan Sau

Snigdha Mazumdar

Snehandu Chanda

Triyasha Kundu

Project Objective

- The proposed model is meant to predict whether an application for credit card will be approved or not based on some features or prerequisites.
- To create a learner system which will predict approval, and learn from previous experiences
- This learner system will work behalf of a person and shortlist the approval by predicting from previous experiences.
- The model will determine the approval of credit card without any manual interfering.
- The model will improve itself with time as it receives more experiences and this will help the model to predict more accurately.

Project Scope

- This model could be implemented in banking software to predict any credit application approved or not.

- This model will work on behalf of a human worker and work automatically without any human interaction which will consume less time than manual process.

- Here our Learning machine will work just like a trained employee with the help of previous dataset and using it as its experience to create a learning machine which will try to predict the most accurately whether an applicant would be approved for credit card or not.

Data Description

- To see the total number of Predictor and predicted variable and the number of observations we use '**shape**' method

```
import pandas as p

df=p.read_csv('Project Data/Credit card Approval.csv')
print(df.shape)
```

```
(10000, 12)
```

Here we can see we have 10000 Observations and total 12 Features

- To get an overview of the type of values Predictor and predicted variables contain, we use '**info**' method

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 12 columns):
customer_id          10000 non-null int64
demographic_slice    10000 non-null object
country_reg          10000 non-null object
ad_exp               10000 non-null object
est_income           10000 non-null float64
hold_bal             10000 non-null float64
pref_cust_prob       10000 non-null float64
imp_cscore           10000 non-null int64
RiskScore            10000 non-null float64
imp_crediteval       10000 non-null float64
axio_score           10000 non-null float64
card_offer           10000 non-null bool
dtypes: bool(1), float64(6), int64(2), object(3)
memory usage: 869.2+ KB
```

Here we can see we have 3 categorical features and one Boolean feature which will be the output feature.

- To visualize the whole dataset in table from we use '**head**' method to see the first 5 observations

```
df.head()
```

	customer_id	demographic_slice	country_reg	ad_exp	est_income	hold_bal	pref_cust_prob	imp_cscore	RiskScore	imp_crediteval	axio_score	card_offer
	713782	AX03efs	W	N	33407.901749	3.000000	0.531112	619	503.249027	23.977827	0.137289	False
	515901	AX03efs	E	N	19927.533533	20.257927	0.297439	527	820.108146	22.986398	0.052264	False
	95166	AX03efs	W	Y	51222.470997	4.000000	0.018463	606	586.605795	24.939219	0.452035	False
	425557	AX03efs	E	Y	67211.587467	18.653631	0.089344	585	634.701982	24.841147	0.564619	False
	624581	AX03efs	W	N	20093.342158	4.000000	0.094948	567	631.949979	24.679363	0.917304	False

- To get a concept about the numerical column’s data at a glance we use ‘describe’ method

```
df.describe()
```

	customer_id	est_income	hold_bal	pref_cust_prob	imp_cscore	RiskScore	imp_crediteval	axio_score
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	496819.831400	65853.355259	20.962621	0.329419	662.548800	670.042869	25.692162	0.393211
std	287391.314157	31093.369592	18.841121	0.223299	90.549985	89.965854	1.889274	0.288243
min	244.000000	2.054543	-2.140206	0.001781	500.000000	324.436647	21.363123	-0.000052
25%	245172.500000	39165.786086	6.150577	0.156965	600.000000	609.231181	24.295435	0.139424
50%	495734.000000	76903.628763	11.913366	0.272263	655.000000	669.493442	25.611903	0.337841
75%	745475.250000	91032.514900	32.238914	0.459890	727.000000	730.484985	27.062519	0.624886
max	999870.000000	150538.809704	81.759632	1.144357	849.000000	1004.497869	30.131214	1.000000

- Here the column variable actually representing –
    - ‘customer_id’ represents the ID number of the customer.
    - ‘demographic_slice’ represents separate classification of users which depends on geographical position, age, occupation, income, education etc.
    - ‘country_reg’ represents the region of country (East or West) from where the user belongs.
    - ‘ad_exp’ represents the whether customer have any additional Experian which is generated from past credit history.
    - ‘est_income’ represents the estimated income of the customer
    - ‘hold_bal’ represents the average balance the customers’ account is holding currently.
    - ‘imp_cscore’ represents important credit score which is calculated from credit/loan history, including your identity information.
    - ‘RiskScore’ represents the amount of risk bank may face to provide credit card to the user.
    - ‘imp_crediteval’ represents important evaluation of the user credit card’s necessity and capacity of paying credit card bill.
    - ‘axio_score’ represents the score which is generated from in depth analysis of financial performance and transactions of the user.
    - ‘card_offer’ is our output column which says the application is approved or not.

Model Building

Exploratory Data Analysis

Univariate Analysis: -

Null Value treatment

```
df.skew(axis=0)
```

```
customer_id        0.019750
est_income        -0.535274
hold_bal           1.015113
pref_cust_prob     0.904168
imp_cscore         0.231556
RiskScore          0.023744
imp_crediteval     0.092559
axio_score         0.451390
card_offer         1.927062
dtype: float64
```

So we can see there is no null value or missing value in our dataset

- Analysis the skewness for all the numeric feature my skew() module

```
df.skew(axis=0)
```

```
customer_id        0.019750
est_income        -0.535274
hold_bal           1.015113
pref_cust_prob     0.904168
imp_cscore         0.231556
RiskScore          0.023744
imp_crediteval     0.092559
axio_score         0.451390
card_offer         1.927062
dtype: float64
```

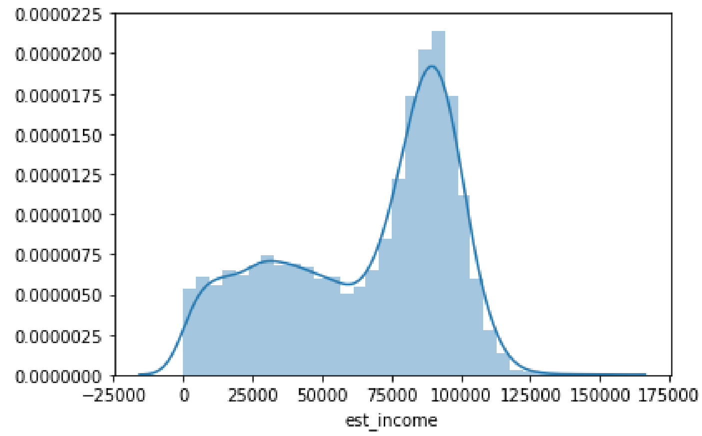
We can see most of the columns are positively skewed and one is negatively skewed.

- To visualize each numeric feature by distplot() or boxplot() & to remove skewness
    1. We did sqrt() or log() [for left skewness]
    2. And **2 or **3 [for right skewness
- All the distplot here shows the distribution here for each numeric feature also

- Feature – ‘est_income’

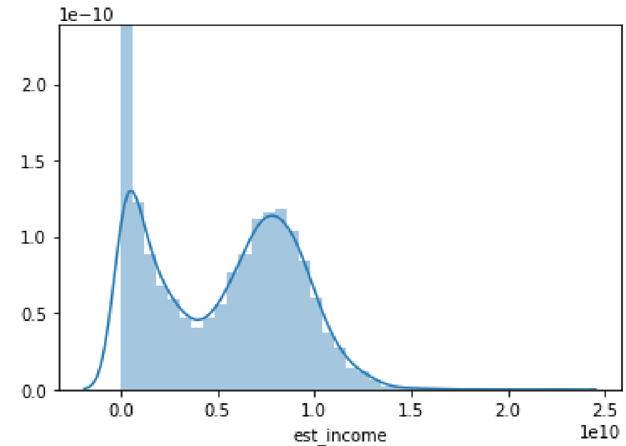
```
sb.distplot(df['est_income'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x126ff4d4128>
```



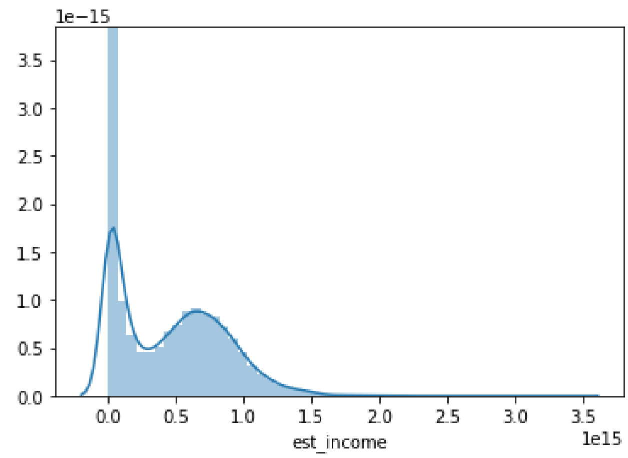

```
df['est_income']=(df['est_income']**2)
print((df['est_income']).skew(axis=0))
sb.distplot(df['est_income'])
pl.show()
```

```
0.056566074848827815
```




```
sb.distplot((df['est_income'])**3)
print((df['est_income']**3).skew(axis=0))
pl.show()
```

```
0.6251649317995985
```

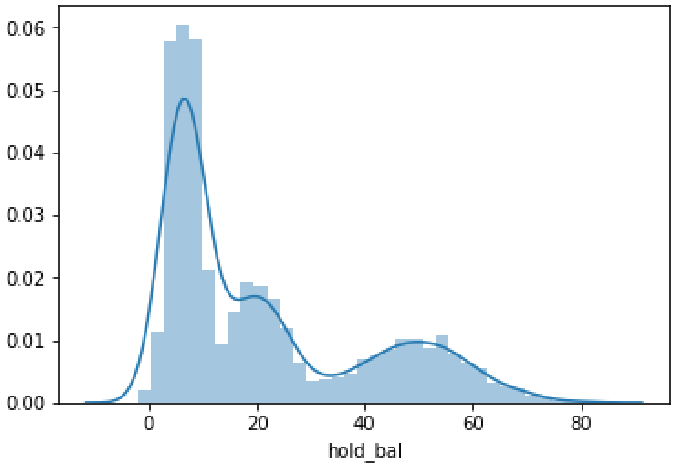


As we can see **2 method removed better skewness we keep the **2 method here

- Feature – ‘hold_bal’

```
sb.distplot(df['hold_bal'])
```

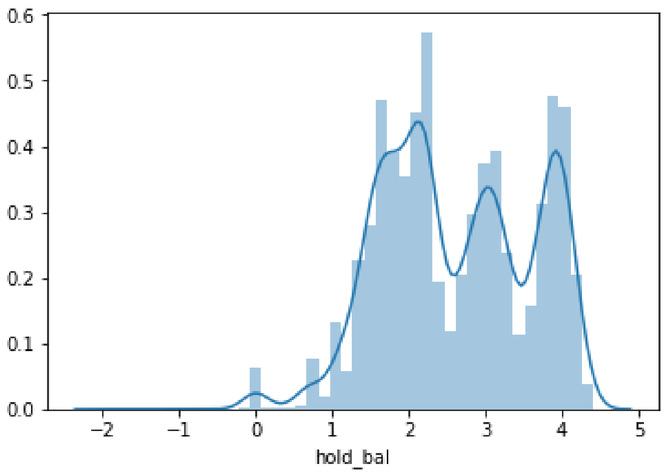
```
<matplotlib.axes._subplots.AxesSubplot at 0x18fe8
```




```
df['hold_bal']=n.log(df['hold_bal'])
df.loc[(df.hold_bal == float('inf')) | (df.hold_bal == float('-inf')), 'hold_bal'] = n.nan
df['hold_bal'].fillna(n.mean(df['hold_bal']),inplace=True)

print((df['hold_bal']).skew(axis=0))
sb.distplot(df['hold_bal'])
pl.show()
```

```
-0.08323215218111003
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: divide by z
  """Entry point for launching an IPython kernel.
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarning: invalid val
  """Entry point for launching an IPython kernel.
```

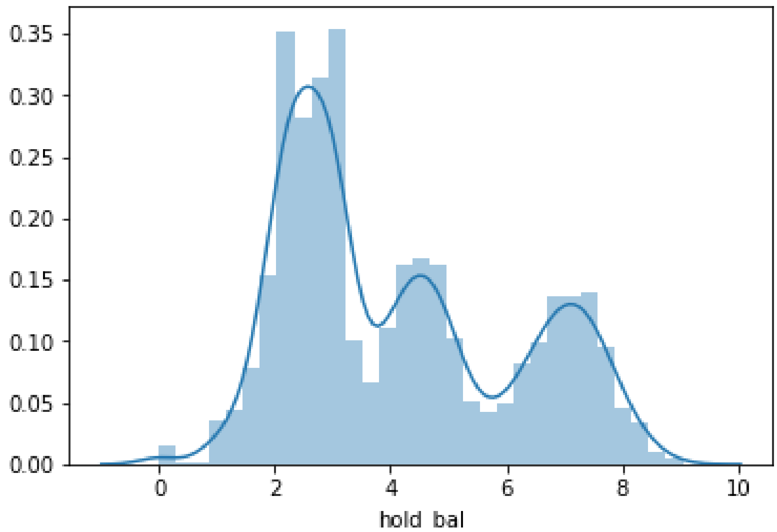


After getting log() of the feature we get some ‘-inf’ and ‘inf’ value so we replaced them by mean of the column

```
df['hold_bal']=n.sqrt(df['hold_bal'])
df['hold_bal'].fillna(n.mean(df['hold_bal']),inplace=True)
print((df['hold_bal']).skew(axis=0))

sb.distplot(df['hold_bal'])
pl.show()
```

```
0.5485373172178414
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py
  """Entry point for launching an IPython kernel.
```



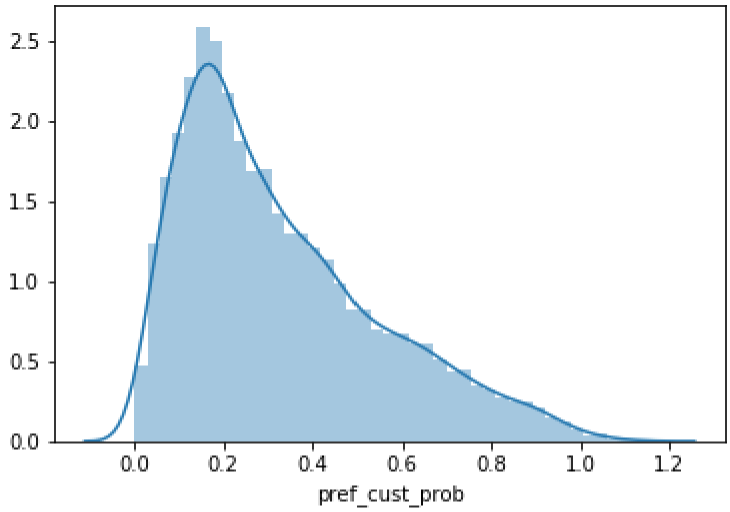
After making sqrt of the column we get some NaN value so we replaced them with mean of the column

Here the n.log() method removes the skewness better so we are keeping the log method here.

- Feature – ‘pref_cust_prob’

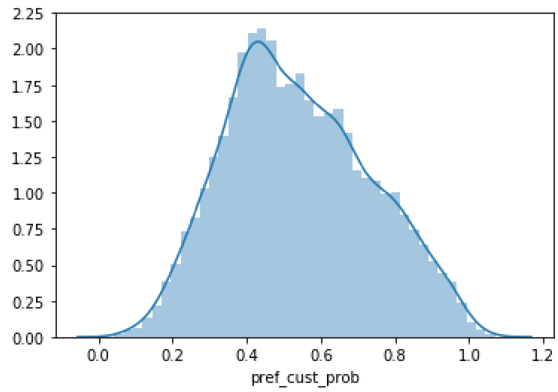
```
sb.distplot(df['pref_cust_prob'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18fe
```



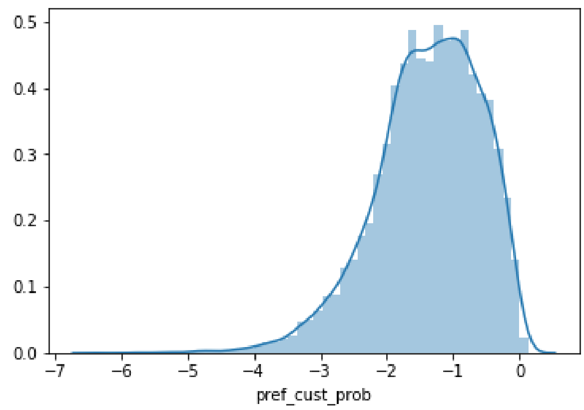

```
df['pref_cust_prob']=n.sqrt(df['pref_cust_prob'])
print((df['pref_cust_prob']).skew(axis=0))
sb.distplot(df['pref_cust_prob'])
```

```
0.24487668783970917

<matplotlib.axes._subplots.AxesSubplot at 0x28c9b8f4208>
```




```
df['pref_cust_prob']=n.log(df['pref_cust_prob'])
sb.distplot(df['pref_cust_prob'])
print(df['pref_cust_prob'].skew(axis=0))
pl.show()
```

```
-0.7844399108293496
```

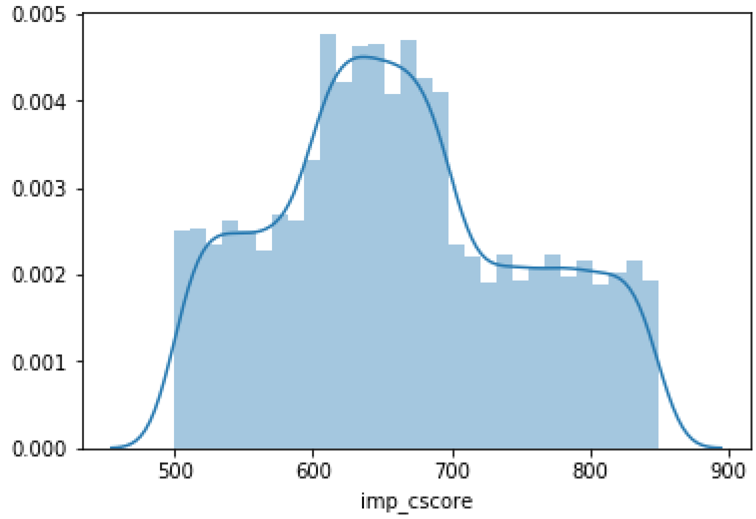


As here the n.sqrt() method removes skewness better we are using the sqrt to remove skewness

- Feature – ‘imp_cscore’

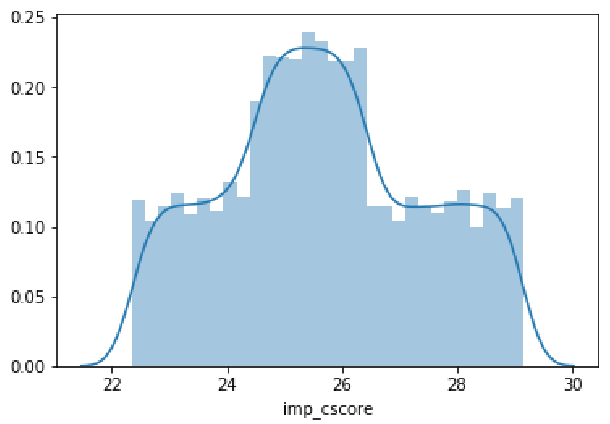
```
sb.distplot(df['imp_cscore'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x18fe93
```



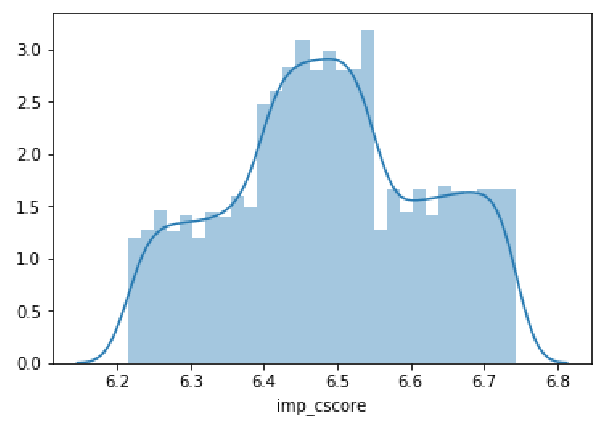

```
sb.distplot(n.sqrt(df['imp_cscore']))
print((n.sqrt(df['imp_cscore'])).skew(axis=0))
pl.show()
```

```
0.11203877217994881
```




```
df['imp_cscore']=n.log(df['imp_cscore'])
print((df['imp_cscore']).skew(axis=0))
sb.distplot((df['imp_cscore']))
pl.show()
```

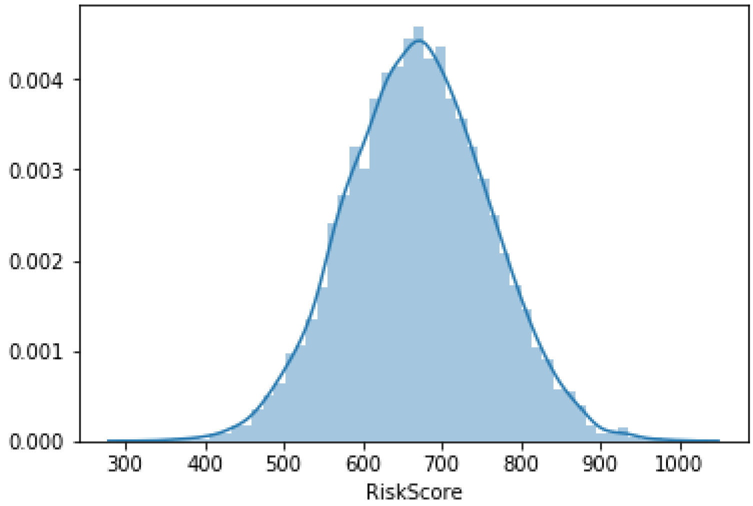
```
-0.009432715044892107
```



As here the n.log() method removes skewness better we are using the log to remove skewness

- Feature – 'RiskScore'

```
sb.distplot(df['RiskScore'])
pl.show()
```

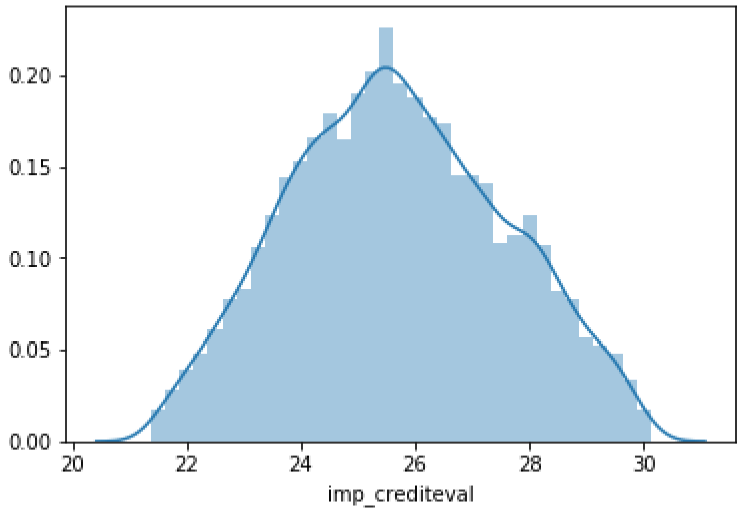


Skewness for RiskScore is very less so we don't change this column

- Feature – 'imp_crediteval'

```
sb.distplot(df['imp_crediteval'])
print((df['imp_crediteval']).skew(axis=0))
pl.show()
```

```
0.09255868507235189
```

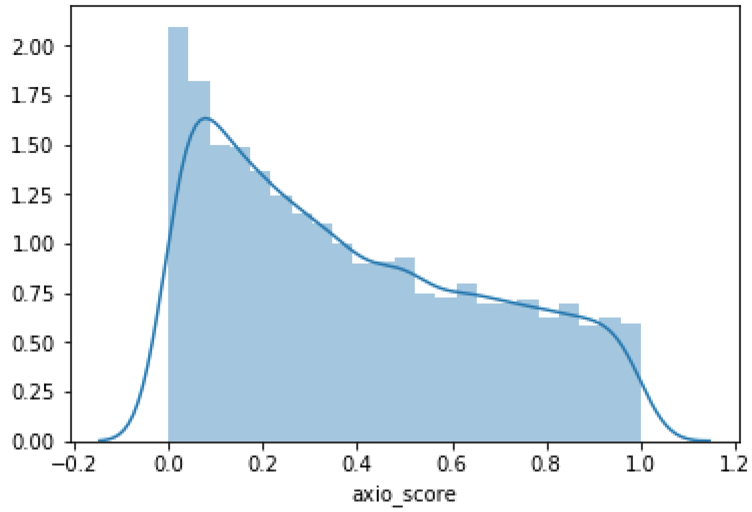


Skewness for imp_crediteval is very less so we don't change this column

- Feature – 'axio_score'

```
sb.distplot(df['axio_score'])
```

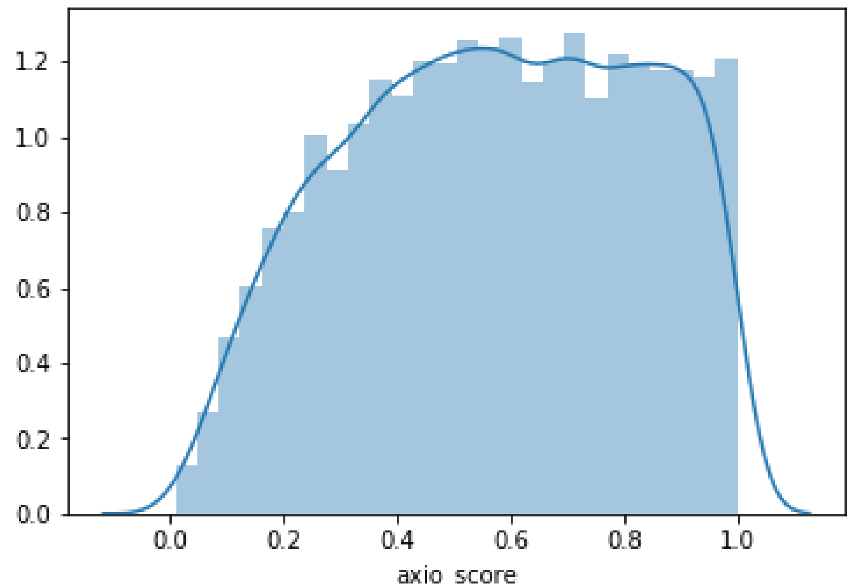
```
<matplotlib.axes._subplots.AxesSubplot at 0x18fe97
```




```
df['axio_score']=n.sqrt(df['axio_score'])
df['axio_score'].fillna(n.mean(df['axio_score']),inplace=True)
print((df['axio_score']).skew(axis=0))
sb.distplot(df['axio_score'])
pl.show()
```

```
-0.14061423775386978
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1:
  """Entry point for launching an IPython kernel.
```

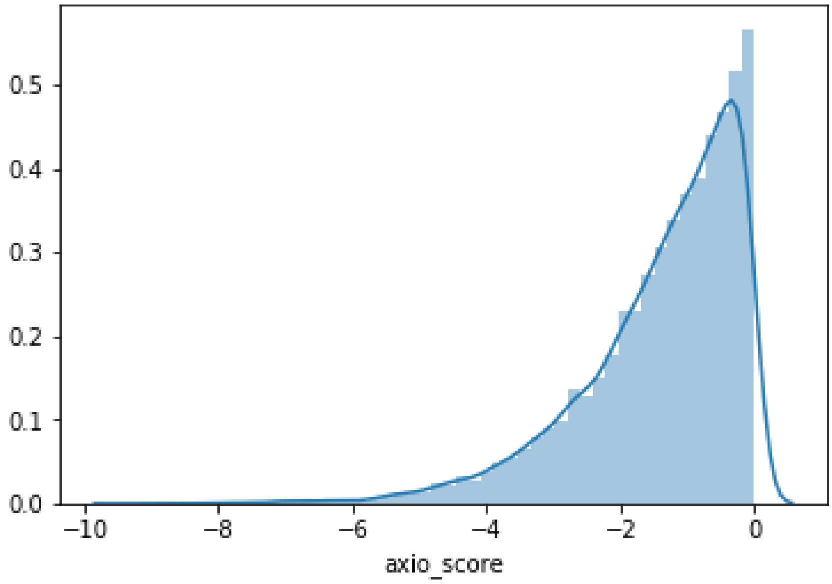


After making sqrt of the column we get some NaN value so we replaced them with mean of the column

```
df['axio_score']=n.log(df['axio_score'])
df['axio_score'].fillna(n.mean(df['axio_score']),inplace=True)
print((df['axio_score']).skew(axis=0))
sb.distplot(df['axio_score'])
pl.show()
```

```
-1.4600846155961356
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:1:
  """Entry point for launching an IPython kernel.
```



After making log of the column we get some NaN value so we replaced them with mean of the column

As here the n.sqrt() method removes skewness better we are using the log to remove skewness

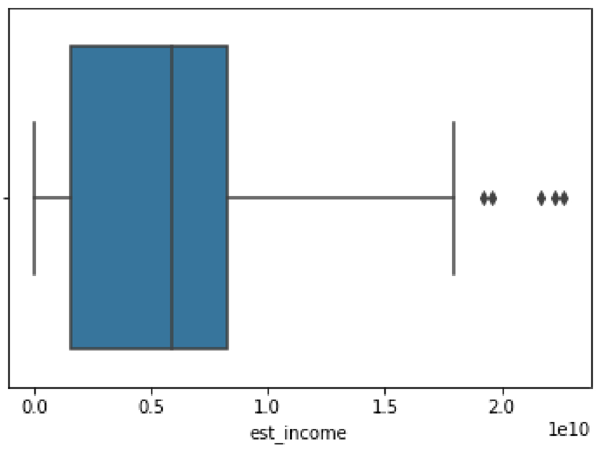
- After removing skewness from all numerical column we check the skewness again to see the improved result

```
df.skew(axis=0)
```

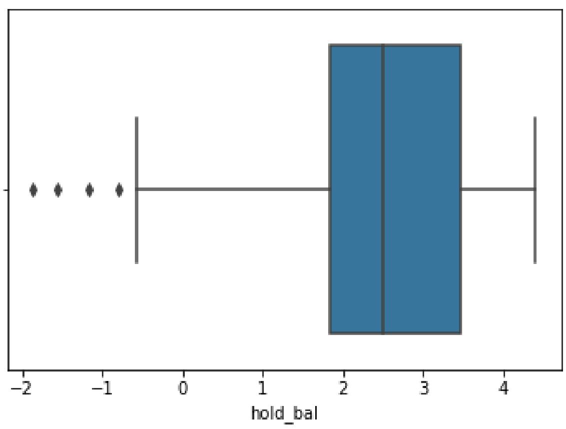
```
customer_id        0.020232
est_income         0.056650
hold_bal           0.033465
pref_cust_prob     0.244403
imp_cscore        -0.009450
RiskScore          0.025514
imp_crediteval     0.092576
axio_score        -0.140682
card_offer         1.926571
dtype: float64
```

- Now we will find whether there is any outliers in any column or not

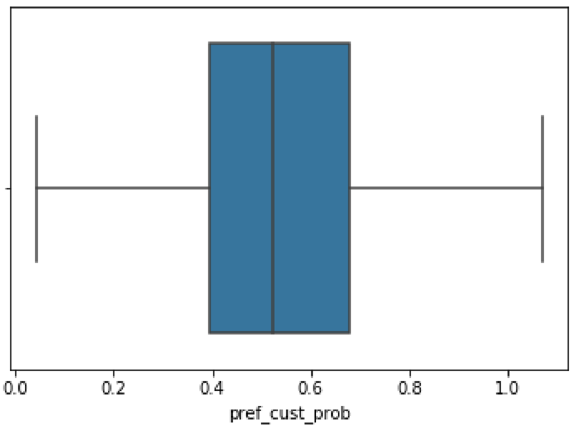
```
sb.boxplot(df['est_income'])
pl.show()
```



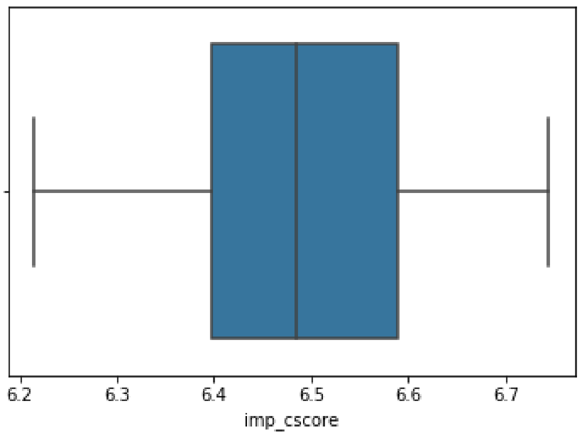

```
sb.boxplot(df['hold_bal'])
pl.show()
```



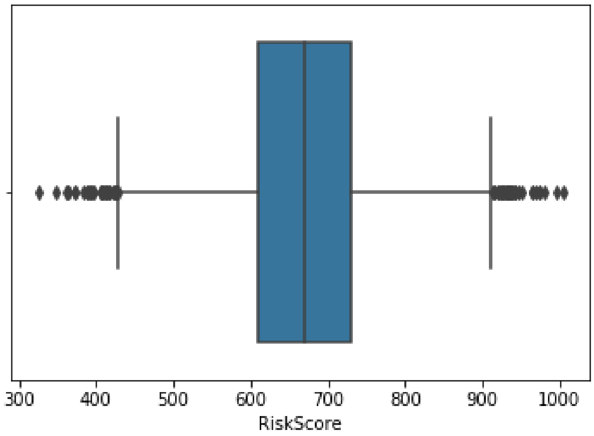

```
sb.boxplot(df['pref_cust_prob'])
pl.show()
```



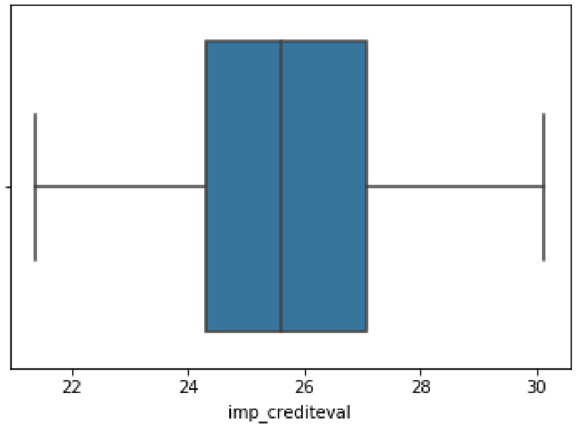

```
sb.boxplot(df['imp_cscore'])
pl.show()
```



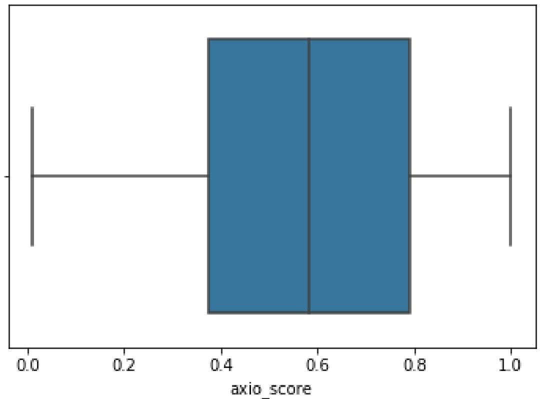
```
sb.boxplot(df['RiskScore'])
pl.show()
```




```
sb.boxplot(df['imp_crediteval'])
pl.show()
```




```
sb.boxplot(df['axio_score'])
pl.show()
```

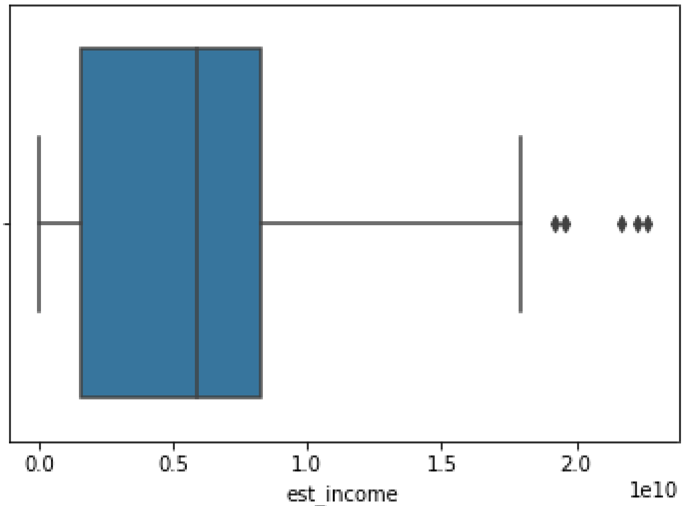


- Here we can see 3 of the features have outliers in their columns

Now we will count the percentage of outliers present in those columns

Est_income

```
sb.boxplot(df['est_income'])
pl.show()
```




```
q1,q3=n.percentile(df['est_income'],[25,75])
IQR=q3-q1
UB=q3 + 1.5*IQR
LB=q1 - 1.5*IQR

median = n.median(df['est_income'])
```

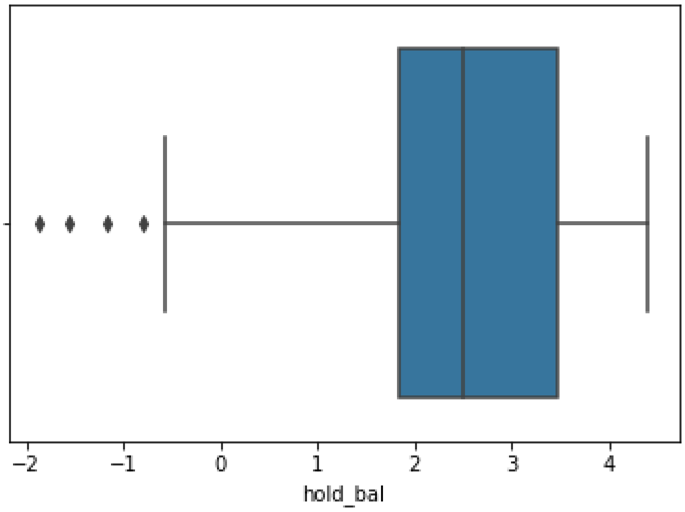
```
count=0
for i in range (len(df['est_income'])):
    if (df.iloc[i]['est_income']>UB) or (df.iloc[i]['est_income']<LB):
        count=count+1
print((count/len(df['est_income']))*100)
```

```
0.05
```

As the percentage of outliers in ‘est_income’ is very less (less than 1%) we don’t remove the outliers here

'hold_bal'

```
sb.boxplot(df['hold_bal'])
pl.show()
```




```
q1,q3=n.percentile(df['hold_bal'],[25,75])
IQR=q3-q1
UB=q3 + 1.5*IQR
LB=q1 - 1.5*IQR

median = n.median(df['hold_bal'])
```

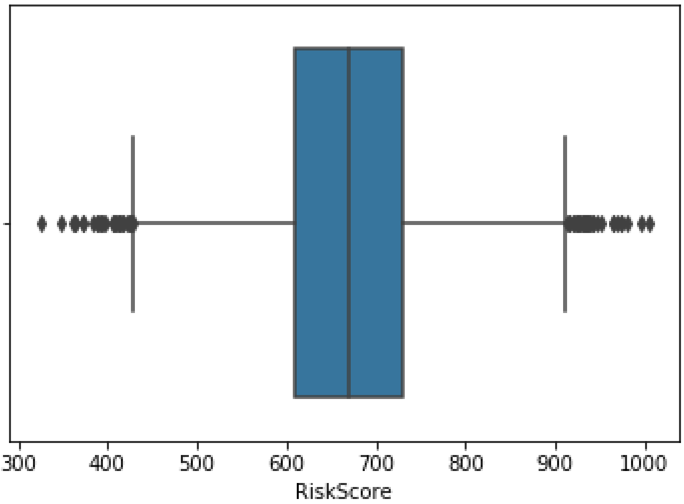
```
count=0
for i in range (len(df['hold_bal'])):
    if (df.iloc[i]['hold_bal']>UB) or (df.iloc[i]['hold_bal']<LB):
        count=count+1
print((count/len(df['hold_bal']))*100)
```

```
0.04
```

As the percentage of outliers in ‘hold_bal’ is very less (less than 1%) we don’t remove the outliers here

'RiskScore'

```
sb.boxplot(df['RiskScore'])
pl.show()
```




```
q1,q3=n.percentile(df['RiskScore'],[25,75])
IQR=q3-q1
UB=q3 + 1.5*IQR
LB=q1 - 1.5*IQR

median = n.median(df['RiskScore'])
```

```
count=0
for i in range (len(df['RiskScore'])):
    if (df.iloc[i]['RiskScore']>UB) or (df.iloc[i]['RiskScore']<LB):
        count=count+1
print((count/len(df['RiskScore']))*100)
```

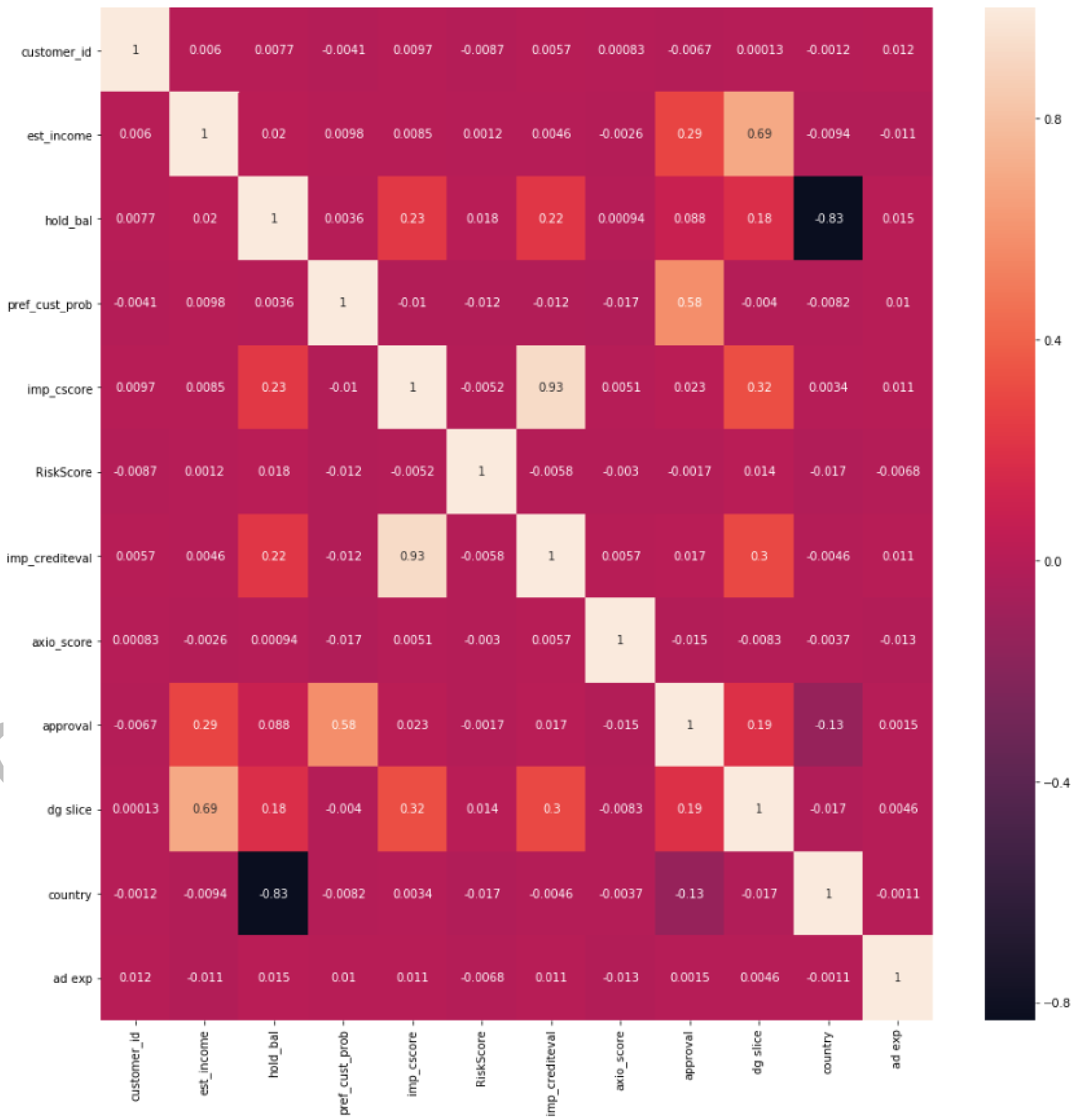
```
0.74
```

As the percentage of outliers in ‘Risk_score’ is very less (less than 1%) we don’t remove the outliers here

Bivariate analysis: -

- To analyse two variable we have to observe relation between each two variable (feature). For this purpose, we plotted heatmap with the correlation for all the variables of the dataset and analysed the correlation of the variables.
    - The heatmap was plotted to visualize the correlation and identify higher correlation easily.


```python
pl.figure(figsize=(16,16))
sb.heatmap(df.corr(),annot=True)
pl.show()
```



- In this plot we can see correlation of each variable with each other and as this plot has same variables on both the axis, the plot is symmetric for upper & lower triangle. So we can work with only one triangle of this plot. For easy navigation we will observe the lower half as this half is closer to the axis's.
- Here we can see for 0 correlation the color is almost 'Red' and Darker color represents High Negative correlation and Lighter color represents high positive correlation.
- The corner value of the heatmap is showing value '1'with the lightest color that shows each variable has the maximum positive correlation with itself which makes perfect sence.

- Here we see a darker cell with correlation value '-0.83' and two light color cell with '0.69' and '0.93' correlation value.

- Here we can drop any one of the feature from a pair of features which shows high positive or negative correlation value. Cause both of these features shows similar effect on the output column. So we can drop one of these paired features to remove 'redundancy' which will overcome 'Curse of Dimensionality'.

- This is why we drop 1 feature from each pair of features for all the 3 pairs.

```
df=df.drop(['imp_crediteval'],axis=1)
df=df.drop(['country'],axis=1)
df=df.drop(['dg slice'],axis=1)
```

Variable Transformation: -

```
#This part is written to convert the catgorical value into numerical value

from sklearn.preprocessing import LabelEncoder as LE

df['dg_slice']=LE().fit_transform(df['demographic_slice'])
df['country']=LE().fit_transform(df['country_reg'])
df['ad_exp']=LE().fit_transform(df['ad_exp'])
```

As there was 3 categorical feature in our dataset we did numeric encoding on those columns to proceed further

```
#this cell is created to discard the columns contained catagorical values

df.drop(['demographic_slice','country_reg','ad_exp'],axis=1,inplace=True)
df.head()
```

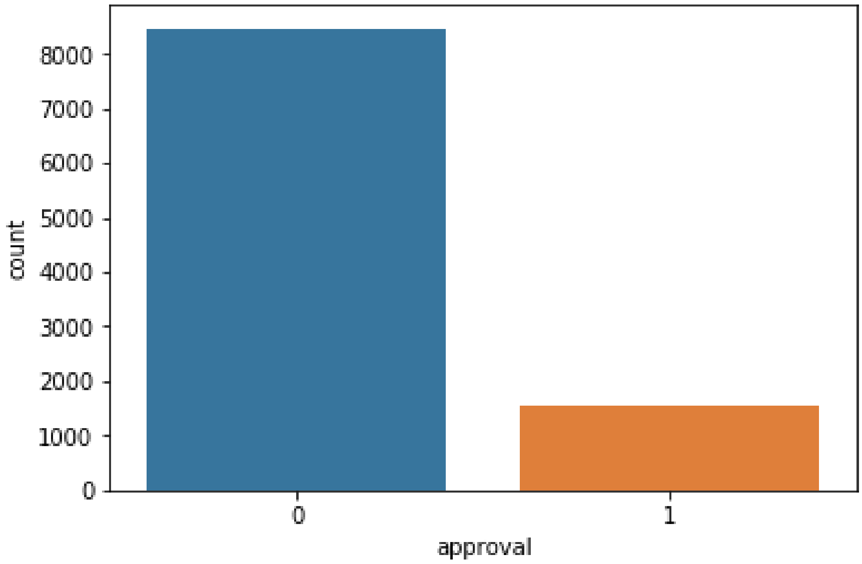
Then we dropped the actual categorical columns

```
approval=df['card_offer'].astype(int)
df.drop(['card_offer'],axis=1,inplace=True)
df['approval']=approval
df.head()
```

Then we converted the Boolean output feature into int for further process.

Class Imbalance: -

```
sb.countplot(df['approval'])
pl.show()
```



Here we can see our output feature is highly imbalanced which may lead us to a biased prediction latter.

So we have to apply class imbalance over here

- Here we use upsampling

upsampling

```
from sklearn.utils import resample as rs

print(df['approval'].value_counts())
major=df[df.approval==0]
minor=df[df.approval==1]
```

```
0    8469
1    1531
Name: approval, dtype: int64
```

```
upsampled=rs(minor,replace=True,n_samples=8466,random_state=1)
upsampled=p.concat([upsampled,major])
upsampled['approval'].value_counts()

y=upsampled['approval']
x=upsampled.drop(['approval'],axis=1)
Xtrn,Xtst,Ytrn,Ytst=tts(x,y,test_size=0.3,random_state=1)
```

Now we create all the four models which will work on classification type of problem.

1. Logistic Regression
2. Decision Tree Model
3. K Nearest Neighbor Model
4. Naïve Bayes Model

The codes are discussed in the next part

Then we also did downsampling

downsampling ¶

```
downsampled=rs(major,replace=False,n_samples=1531,random_state=1)
downsampled=p.concat([downsampled,minor])
downsampled['approval'].value_counts()

y=downsampled['approval']
x=downsampled.drop(['approval'],axis=1)
Xtrn,Xtst,Ytrn,Ytst=tts(x,y,test_size=0.3,random_state=1)
```

Again we create all the four models which will work on classification type of problem and then latter we will evaluate which one is giving us better result among upsampling, downsampling and without resampling.

Code

The modules we used for creating models are shown below

```
from sklearn.model_selection import train_test_split as tts

from sklearn.linear_model import LogisticRegression as LG
from sklearn.tree import DecisionTreeClassifier as dtc
from sklearn.preprocessing import StandardScaler as SC
from sklearn.neighbors import KNeighborsClassifier as KNC
from sklearn.naive_bayes import GaussianNB as GB

from sklearn.metrics import confusion_matrix as cm
from sklearn.metrics import accuracy_score as acc
from sklearn.metrics import recall_score as rcc
from sklearn.metrics import precision_score as prr
from sklearn.metrics import classification_report as cr
```

At first we split the dataset into train and test set by train_test_split() method

```
#this cell is created to split the dataset into training and testing set

train_x,test_x,train_y,test_y=tts(x,y,test_size=0.3,random_state=1)
```

Logistic Regression

As the problem is classification type of problem we use Logistic Regression

```
#this cell is created to split create the model

model=LG()
model.fit(train_x,train_y)
predicted_approval=model.predict(test_x)
```

By the codes below we stored the accuracy, precession, recall, TPR, TNR, FPR & FNR to a list to evaluate it further

```
m1=cm(test_y,predicted_approval)
print(m1)

lg0=[acc(test_y,predicted_approval)]
lg0.append(prr(test_y,predicted_approval))
lg0.append(rcc(test_y,predicted_approval))
lg0.append(m1[0][0]/(m1[0][0]+m1[0][1])) #TPR
lg0.append(m1[1][1]/(m1[1][1]+m1[1][0])) #TNR
lg0.append(m1[1][0]/(m1[1][1]+m1[1][0])) #FPR
lg0.append(m1[0][1]/(m1[0][0]+m1[0][1])) #FNR
```

For a confusion matrix analysis

```
[[2491   44]
 [  60  405]]
```

Here the axis 1 is the predicted axis and the axis 0 is the observed axis because we passed the test parameter first and then passed the predicted parameter in the confusion matrix module.
Here ‘0’ is taken as true output and ‘1’ Is taken as false output So,

In the matrix position [0][0] contains TP value
In the matrix position [0][1] contains FN value
In the matrix position [1][0] contains FP value
In the matrix position [1][1] contains TN value
Now for other 3 models the codes are shown below

Decision Tree

```
entropy=dtc(criterion='entropy',random_state=100)
entropy.fit(train_x,train_y)
predict_y=entropy.predict(test_x)
m2=cm(test_y,predict_y)
print(m2)
```

K Nearest Neighbor

```
train_x=SC().fit_transform(train_x)
test_x=SC().fit_transform(test_x)

classifier=KNC(n_neighbors=83,p=2,metric='euclidean')
classifier.fit(train_x,train_y)
y_pred=classifier.predict(test_x)
m3=cm(test_y,y_pred)
print(m3)
```

Naïve Bayes

```
model_b=GB().fit(train_x,train_y)
predict_b=model_b.predict(test_x)
m4=cm(test_y,predict_b)
print(m4)
```

➢ Then we create model using Logistic Regression and evaluated the matrix

Logistic Regression (no resample)

```
#this cell is created to split create the model

model=LG()
model.fit(train_x,train_y)
predicted_approval=model.predict(test_x)
```

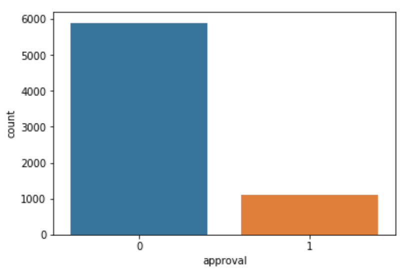
```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\line
to 'lbfgs' in 0.22. Specify a solver to silence this wa
  FutureWarning)
```

```
m1=cm(test_y,predicted_approval)
print(m1)

print(cr(test_y,predicted_approval))
```

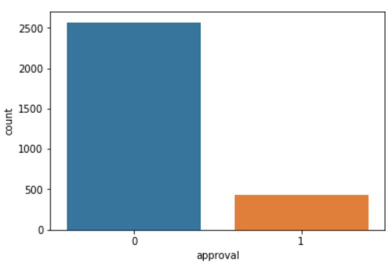
```
[[2539    0]
 [ 461    0]]
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      2539
           1       0.00      0.00      0.00       461

   micro avg       0.85      0.85      0.85      3000
   macro avg       0.42      0.50      0.46      3000
weighted avg       0.72      0.85      0.78      3000
```

➢ In this matrix we can see both the FN and TN value is ‘0’, that means the number of predicted ‘N’ is ‘0’
All the predicted value is ‘T’ (here ‘T’ is ‘0’)

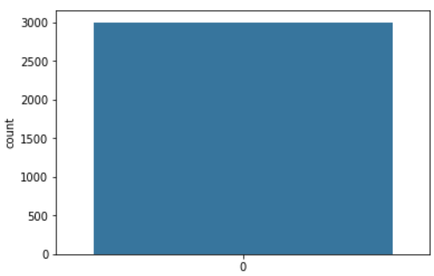
```
sb.countplot(train_y)
pl.show()
```




```
sb.countplot(test_y)
pl.show()
```




```
sb.countplot(predicted_approval)
pl.show()
```



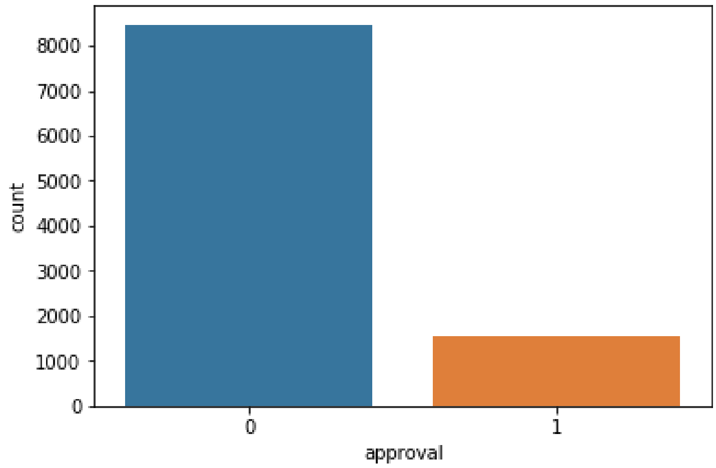
❖ In these above figure the 1st figure shows the count-plot of the **output feature** of the **training** set only
   ▪ Here we can visualize the ration of 0 & 1 in output feature which is highly imbalanced (almost 6:1)

❖ In these above figure the 2nd figure shows the count-plot of the **output feature** of the **testing** set only
   ▪ Here also we can visualize the ration of 0 & 1 in output feature which is highly imbalanced almost (almost 5:1)

- In these above figure the 3rd figure shows the count-plot of the **output feature** of the **predicted** set only
    - Here we can see all the predicted output is '0' & there is no predicted output of '1'
    - That means as the model was trained for output '0' 6 times more than for output '1', the prediction for output '0' was 100% but the prediction of output '1' was 0% because the model was almost untrained for output '1' rather than output '0'.

- Reason behind this class imbalance is, the whole dataset has class imbalance in the output feature which is shown below in the count-plot

```
sb.countplot(df['approval'])
pl.show()
```



- To remove this class imbalance, we have to do Resampling of the output features (Up-sampling or Down-sampling)

  To know which one will lead to better result we did both up-sampling and down-sampling and get these matrices after evaluation

Logistic Regression (Upsample)

```
US_model_r=LG().fit(Xtrn,Ytrn)
USpredict_r=US_model_r.predict(Xtst)
m8lU=cm(Ytst,USpredict_r)
print(m8lU)

print(acc(USpredict_r,Ytst))
print(cr(USpredict_r,Ytst))
```

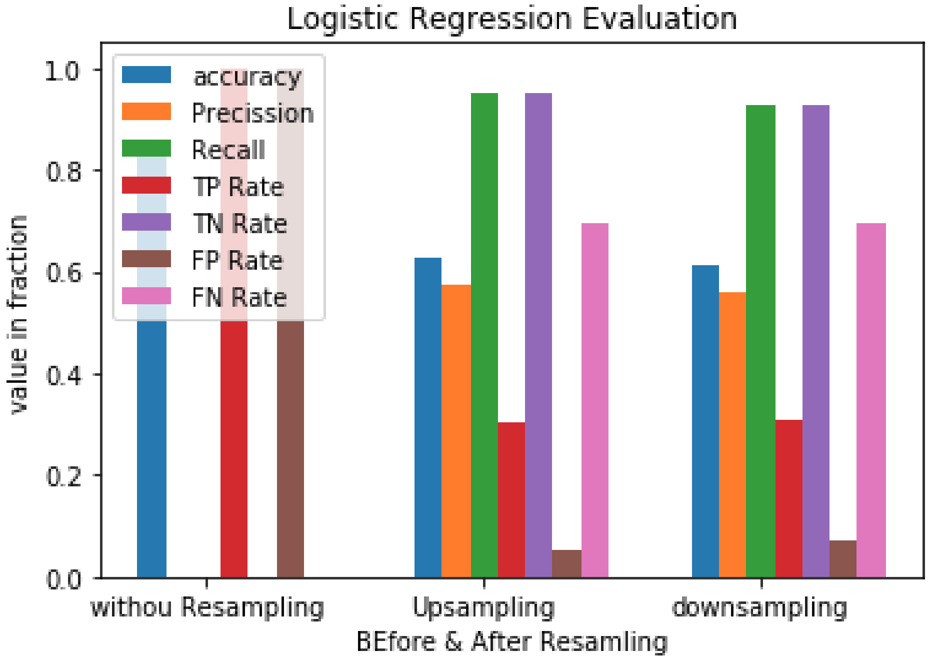
```
[[ 772 1774]
 [ 128 2406]]
0.6255905511811024
              precision    recall  f1-score   support

           0       0.30      0.86      0.45       900
           1       0.95      0.58      0.72      4180

   micro avg       0.63      0.63      0.63      5080
   macro avg       0.63      0.72      0.58      5080
weighted avg       0.83      0.63      0.67      5080
```

Logistic Regression (Down Sampling)

```
DS_model_r=LG().fit(Xtrn,Ytrn)
DSpredict_r=DS_model_r.predict(Xtst)

m8lD=cm(Ytst,DSpredict_r)
print(m8lD)

print(acc(Ytst,DSpredict_r))
print(cr(Ytst,DSpredict_r))
```

```
[[145 326]
 [ 32 416]]
0.6104461371055495
              precision    recall  f1-score   support

           0       0.82      0.31      0.45       471
           1       0.56      0.93      0.70       448

   micro avg       0.61      0.61      0.61       919
   macro avg       0.69      0.62      0.57       919
weighted avg       0.69      0.61      0.57       919
```

- For better experince of visualization we plot and table the accuracy, Precession, Recall and TPR, TNR, FPR, FNR with barplot below



	withou Resampling	Upsampling	downsampling
accuracy	0.845	0.623106	0.628945
Precission	0.000	0.571326	0.573793
Recall	0.000	0.948187	0.928571
TP rate	1.000	0.305988	0.343949
TN rate	0.000	0.948187	0.928571
FP rate	1.000	0.051813	0.071429
FN rate	0.000	0.694012	0.656051

- Not only for Logistic regression, we evaluated this dataset with three more model - Decision Tree model, K Neighbours Model, Naïve Bayes Model and without resampling and with resampling and the result are shown below

Decision Tree Model

Decision tree (no resample)

```python
entropy=dtc(criterion='entropy',random_state=100)
entropy.fit(train_x,train_y)
predict_y=entropy.predict(test_x)
m2=cm(test_y,predict_y)
print(m2)

print(acc(test_y,predict_y))
print(cr(test_y,predict_y))
```

```
[[2480   59]
 [  47  414]]
0.9646666666666667
              precision    recall  f1-score   support

           0       0.98      0.98      0.98      2539
           1       0.88      0.90      0.89       461

   micro avg       0.96      0.96      0.96      3000
   macro avg       0.93      0.94      0.93      3000
weighted avg       0.97      0.96      0.96      3000
```

Decision Tree (Up sampling)

```python
US_model_t=dtc(criterion='entropy',random_state=1)
US_model_t.fit(Xtrn,Ytrn)
USpredict_t=US_model_t.predict(Xtst)
m8tU=cm(Ytst,USpredict_t)
print(m8tU)

print(acc(USpredict_t,Ytst))
print(cr(USpredict_t,Ytst))
```

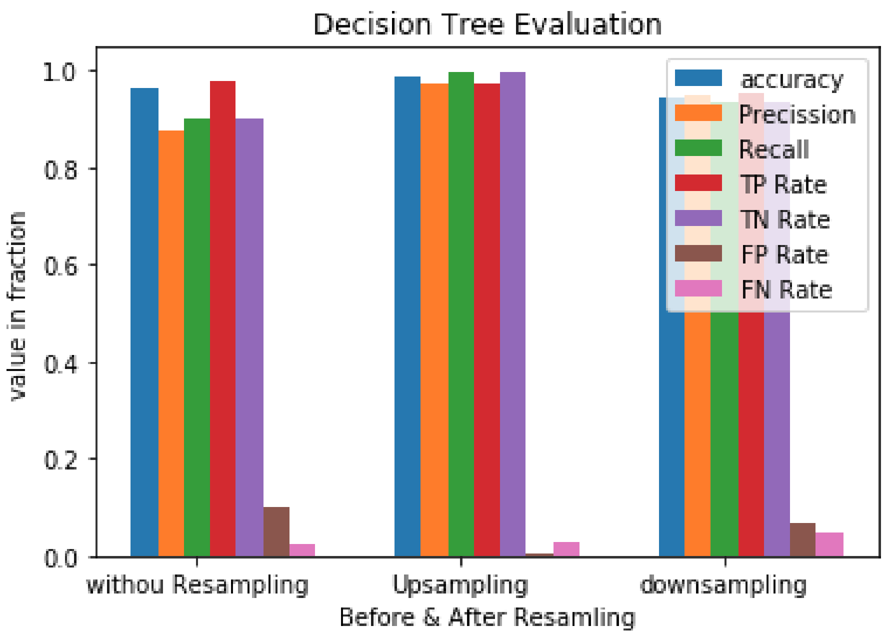
```
[[2479   67]
 [  36 2498]]
0.9797244094488189
              precision    recall  f1-score   support

           0       0.97      0.99      0.98      2515
           1       0.99      0.97      0.98      2565

   micro avg       0.98      0.98      0.98      5080
   macro avg       0.98      0.98      0.98      5080
weighted avg       0.98      0.98      0.98      5080
```

Decision Tree (Down Sampling)

```
DS_model_t=dtc(criterion='entropy',random_state=1)
DS_model_t.fit(Xtrn,Ytrn)
DSpredict_t=DS_model_t.predict(Xtst)

m8tD=cm(Ytst,DSpredict_t)
print(m8tD)

print(acc(Ytst,DSpredict_t))
print(cr(Ytst,DSpredict_t))
```

```
[[448  23]
 [ 30 418]]
0.9423286180631121
              precision    recall  f1-score   support

           0       0.94      0.95      0.94       471
           1       0.95      0.93      0.94       448

   micro avg       0.94      0.94      0.94       919
   macro avg       0.94      0.94      0.94       919
weighted avg       0.94      0.94      0.94       919
```

❖ The result is visualized by a table and plot here below



	withou Resampling	Upsampling	downsampling
accuracy	0.965333	0.987601	0.956474
Precission	0.902004	0.977734	0.945415
Recall	0.870968	0.997609	0.966518
TP rate	0.982643	0.977838	0.946921
TN rate	0.870968	0.997609	0.966518
FP rate	0.129032	0.002391	0.033482
FN rate	0.017357	0.022162	0.053079

➢ K Neighbours Model

K neighbors (no resample) ¶

```
train_x=SC().fit_transform(train_x)
test_x=SC().fit_transform(test_x)

classifier=KNC(n_neighbors=83,p=2,metric='euclidean')
classifier.fit(train_x,train_y)
y_pred=classifier.predict(test_x)
m3=cm(test_y,y_pred)
print(m3)

print(acc(test_y,y_pred))
print(cr(test_y,y_pred))
```

```
[[2534    5]
 [ 261  200]]
0.9113333333333333
              precision    recall  f1-score   support

           0       0.91      1.00      0.95      2539
           1       0.98      0.43      0.60       461

   micro avg       0.91      0.91      0.91      3000
   macro avg       0.94      0.72      0.78      3000
weighted avg       0.92      0.91      0.90      3000
```

K neighbors (Up sampling)

```
Xtrn=SC().fit_transform(Xtrn)
Xtst=SC().fit_transform(Xtst)

US_model_n=KNC(n_neighbors=19,p=2,metric='euclidean')
US_model_n.fit(Xtrn,Ytrn)
US_predict_n=US_model_n.predict(Xtst)
m8nU=cf(Ytst,US_predict_n)
print(m8nU)

print(acc(Ytst,US_predict_n))
print(cr(Ytst,US_predict_n))
```

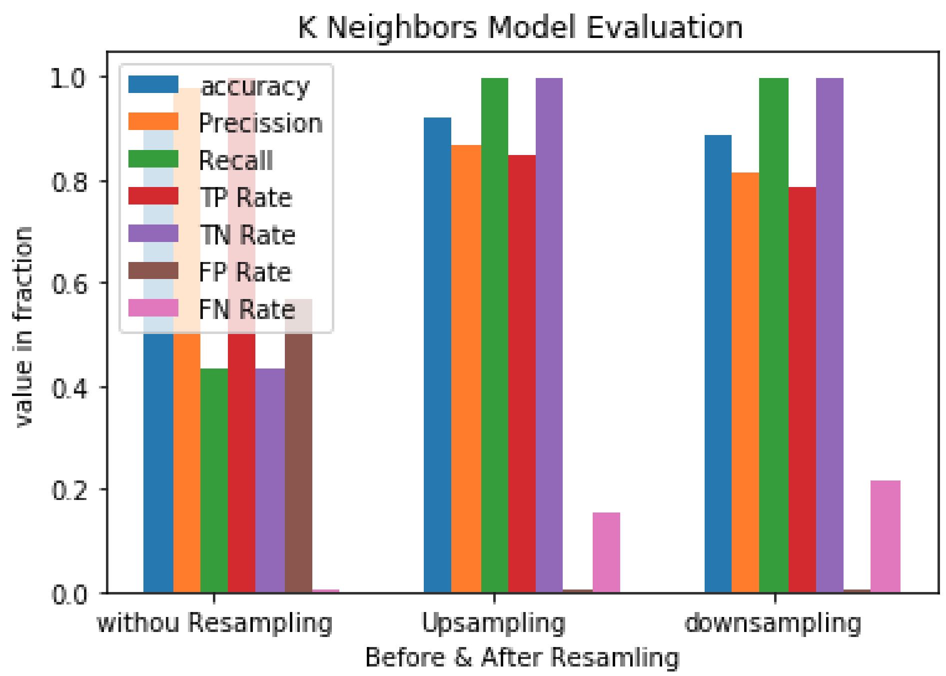
```
[[2237  309]
 [  13 2521]]
0.9366141732283465
              precision    recall  f1-score   support

           0       0.99      0.88      0.93      2546
           1       0.89      0.99      0.94      2534

   micro avg       0.94      0.94      0.94      5080
   macro avg       0.94      0.94      0.94      5080
weighted avg       0.94      0.94      0.94      5080
```

K neighbors (Down Sampling)

```
Xtrn=SC().fit_transform(Xtrn)
Xtst=SC().fit_transform(Xtst)

DS_model_n=KNC(n_neighbors=19,p=2,metric='euclidean')
DS_model_n.fit(Xtrn,Ytrn)
DS_predict_n=DS_model_n.predict(Xtst)
m8nD=cf(Ytst,DS_predict_n)
print(m8nD)

print(acc(Ytst,DS_predict_n))
print(cr(Ytst,DS_predict_n))
```

```
[[2237  309]
 [  13 2521]]
0.9366141732283465
              precision    recall  f1-score   support

           0       0.99      0.88      0.93      2546
           1       0.89      0.99      0.94      2534

   micro avg       0.94      0.94      0.94      5080
   macro avg       0.94      0.94      0.94      5080
weighted avg       0.94      0.94      0.94      5080
```

- The result is visualized by a table and plot here below



	withou Resampling	Upsampling	downsampling
accuracy	0.943000	0.926786	0.922742
Precission	0.924855	0.894426	0.877756
Recall	0.688172	0.965723	0.977679
TP rate	0.989744	0.888802	0.870488
TN rate	0.688172	0.965723	0.977679
FP rate	0.311828	0.034277	0.022321
FN rate	0.010256	0.111198	0.129512

➢ <u>Naïve Bayes Model</u>

Naive Bayes (no resample)

```
model_b=GB().fit(train_x,train_y)
predict_b=model_b.predict(test_x)
m4=cm(test_y,predict_b)
print(m4)

print(acc(test_y,predict_b))
print(cr(test_y,predict_b))
```

```
[[2505   34]
 [ 139  322]]
0.9423333333333334
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      2539
           1       0.90      0.70      0.79       461

   micro avg       0.94      0.94      0.94      3000
   macro avg       0.93      0.84      0.88      3000
weighted avg       0.94      0.94      0.94      3000
```

Naive Bayes (Up sampling)

```
from sklearn.naive_bayes import GaussianNB as GB
US_model_b=GB().fit(Xtrn,Ytrn)
US_predict_b=US_model_b.predict(Xtst)
m8bU=cm(Ytst,US_predict_b)
print(m8bU)

print(acc(Ytst,US_predict_b))
print(cr(Ytst,US_predict_b))
```

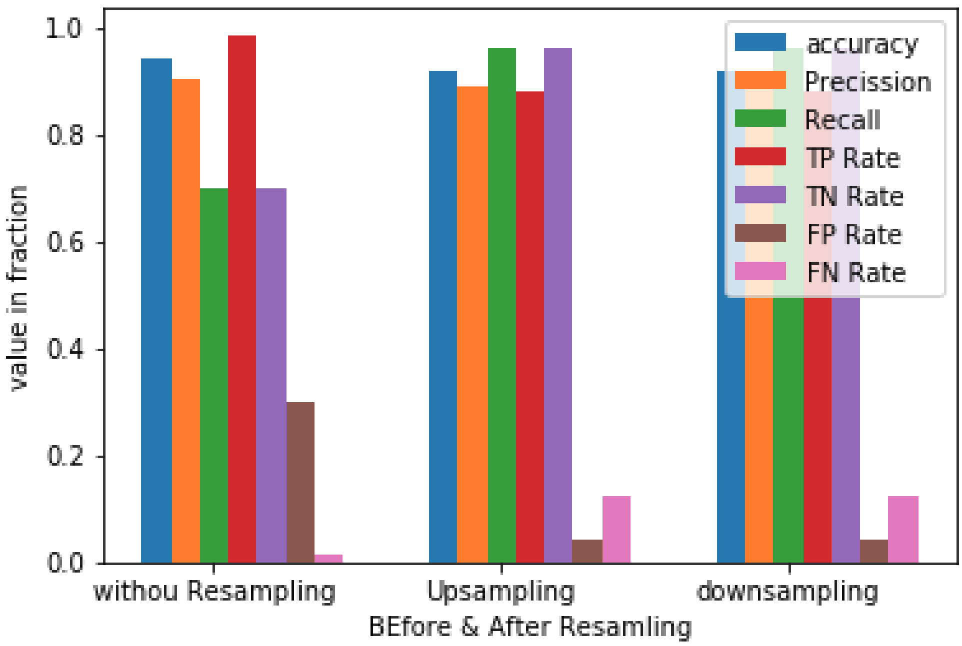
```
[[2239  307]
 [ 101 2433]]
0.9196850393700787
              precision    recall  f1-score   support

           0       0.96      0.88      0.92      2546
           1       0.89      0.96      0.92      2534

   micro avg       0.92      0.92      0.92      5080
   macro avg       0.92      0.92      0.92      5080
weighted avg       0.92      0.92      0.92      5080
```

Naive Bayes (Down Sampling)

```
DS_model_b=GB().fit(Xtrn,Ytrn)
DS_predict_b=DS_model_b.predict(Xtst)
m8bD=cm(Ytst,DS_predict_b)
print(m8bD)

print(acc(Ytst,DS_predict_b))
print(cr(Ytst,DS_predict_b))
```

```
[[414  57]
 [ 18 430]]
0.9183895538628944
              precision    recall  f1-score   support

           0       0.96      0.88      0.92       471
           1       0.88      0.96      0.92       448

   micro avg       0.92      0.92      0.92       919
   macro avg       0.92      0.92      0.92       919
weighted avg       0.92      0.92      0.92       919
```

❖ The result is visualized by a table and plot here below



	withou Resampling	Upsampling	downsampling
accuracy	0.943000	0.926786	0.922742
Precission	0.924855	0.894426	0.877756
Recall	0.688172	0.965723	0.977679
TP rate	0.989744	0.888802	0.870488
TN rate	0.688172	0.965723	0.977679
FP rate	0.311828	0.034277	0.022321
FN rate	0.010256	0.111198	0.129512

Here we can see for all the cases the upsampled dataset are giving us better result in terms of accuracy, precession, recall and TPR, TNR and the lesser FPR and FNR

So we can say our data need to be upsampled. So further process are done on upsampled sataset.

- Now we will apply the wrapper method to reduce 'Curse of Dimensionality' by selecting 'K Best features'
    - As we know that idle value of K is 5 for larger dataset we would take K=5 initially and apply all the four models (Logistic Regression Model, Decision Tree Model, K Neighbours Model, Naïve Bayes Model) on the dataset.

➢ **<u>For K=5</u>**

Logisctic Regression

```
model_l5=LG().fit(Xtrn,Ytrn)
predict_l5=model_l5.predict(Xtst)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\lin
to 'lbfgs' in 0.22. Specify a solver to silence this w
  FutureWarning)
```

```
m5L=cm(Ytst,predict_l5)
print(m5L)

print(acc(Ytst,predict_l5))
MA5=cr(Ytst,predict_l5)
print(MA5)
```

```
[[   0 2546]
 [   0 2534]]
0.4988188976377953
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      2546
           1       0.50      1.00      0.67      2534

   micro avg       0.50      0.50      0.50      5080
   macro avg       0.25      0.50      0.33      5080
weighted avg       0.25      0.50      0.33      5080
```

Decision Tree

```
model_t5=dtc(criterion='entropy',random_state=1)
model_t5.fit(Xtrn,Ytrn)
predict_t5=model_t5.predict(Xtst)
m5t=cm(Ytst,predict_t5)
print(m5t)

print(acc(Ytst,predict_t5))
print(cr(Ytst,predict_t5))
```

```
[[2485   61]
 [   8 2526]]
0.9864173228346457
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      2546
           1       0.98      1.00      0.99      2534

   micro avg       0.99      0.99      0.99      5080
   macro avg       0.99      0.99      0.99      5080
weighted avg       0.99      0.99      0.99      5080
```

K neighbors

```
Xtrn=SC().fit_transform(Xtrn)
Xtst=SC().fit_transform(Xtst)

model_n5=KNC(n_neighbors=83,p=2,metric='euclidean')
model_n5.fit(Xtrn,Ytrn)
predict_n5=model_n5.predict(Xtst)
m5n=cm(Ytst,predict_n5)
print(m5n)

print(acc(Ytst,predict_n5))
print(cr(Ytst,predict_n5))
```

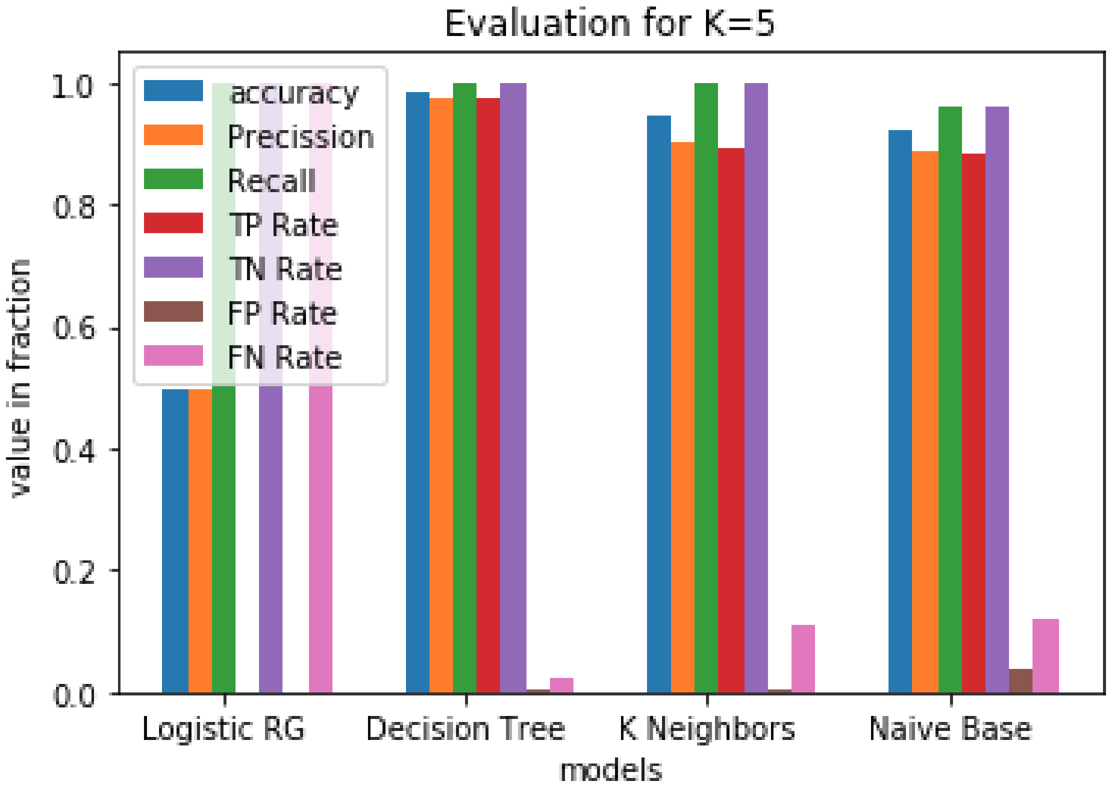
```
[[2270  276]
 [   4 2530]]
0.9448818897637795
              precision    recall  f1-score   support

           0       1.00      0.89      0.94      2546
           1       0.90      1.00      0.95      2534

   micro avg       0.94      0.94      0.94      5080
   macro avg       0.95      0.95      0.94      5080
weighted avg       0.95      0.94      0.94      5080
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB as GB
model_b5=GB().fit(Xtrn,Ytrn)
predict_b5=model_b5.predict(Xtst)
m5b=cm(Ytst,predict_b5)
print(m5b)

print(acc(Ytst,predict_b5))
print(cr(Ytst,predict_b5))
```

```
[[2243  303]
 [ 101 2433]]
0.9204724409448819
              precision    recall  f1-score   support

           0       0.96      0.88      0.92      2546
           1       0.89      0.96      0.92      2534

   micro avg       0.92      0.92      0.92      5080
   macro avg       0.92      0.92      0.92      5080
weighted avg       0.92      0.92      0.92      5080
```

- Here we will evaluate the values of the matrices and accuracy, recall, precession, TPR, TNR, FPR & HNR in table form and visualize in bar plot for better understanding



	Logistic RG	Decision Tree	K Neighbors	Naive Base
accuracy	0.4938	0.989963	0.946861	0.927180
Precission	0.4938	0.982339	0.905176	0.895087
Recall	1.0000	0.997609	0.996811	0.965723
TP rate	0.0000	0.982504	0.898134	0.889580
TN rate	1.0000	0.997609	0.996811	0.965723
FP rate	0.0000	0.002391	0.003189	0.034277
FN rate	1.0000	0.017496	0.101866	0.110420

➢ For K=6

Logisctic Regression

```
model_l6=LG().fit(Xtrn,Ytrn)
predict_l6=model_l6.predict(Xtst)
m6l=cm(Ytst,predict_l6)
print(m6l)

print(acc(Ytst,predict_l6))
print(cr(Ytst,predict_l6))
```

```
[[ 772 1774]
 [ 128 2406]]
0.6255905511811024
              precision    recall  f1-score   support

           0       0.86      0.30      0.45      2546
           1       0.58      0.95      0.72      2534

   micro avg       0.63      0.63      0.63      5080
   macro avg       0.72      0.63      0.58      5080
weighted avg       0.72      0.63      0.58      5080
```

Decision Tree

```
model_t6=dtc(criterion='entropy',random_state=1)
model_t6.fit(Xtrn,Ytrn)
predict_t6=model_t6.predict(Xtst)
m6t=cm(Ytst,predict_t6)
print(m6t)

print(acc(Ytst,predict_t6))
print(cr(Ytst,predict_t6))
```

```
[[2478   68]
 [   6 2528]]
0.9854330708661417
              precision    recall  f1-score   support

           0       1.00      0.97      0.99      2546
           1       0.97      1.00      0.99      2534

   micro avg       0.99      0.99      0.99      5080
   macro avg       0.99      0.99      0.99      5080
weighted avg       0.99      0.99      0.99      5080
```

K neighbors

```
Xtrn=SC().fit_transform(Xtrn)
Xtst=SC().fit_transform(Xtst)
model_n6=KNC(n_neighbors=83,p=2,metric='euclidean')
model_n6.fit(Xtrn,Ytrn)
predict_n6=model_n6.predict(Xtst)
m6n=cm(Ytst,predict_n6)
print(m6n)

print(acc(Ytst,predict_n6))
print(cr(Ytst,predict_n6))
```

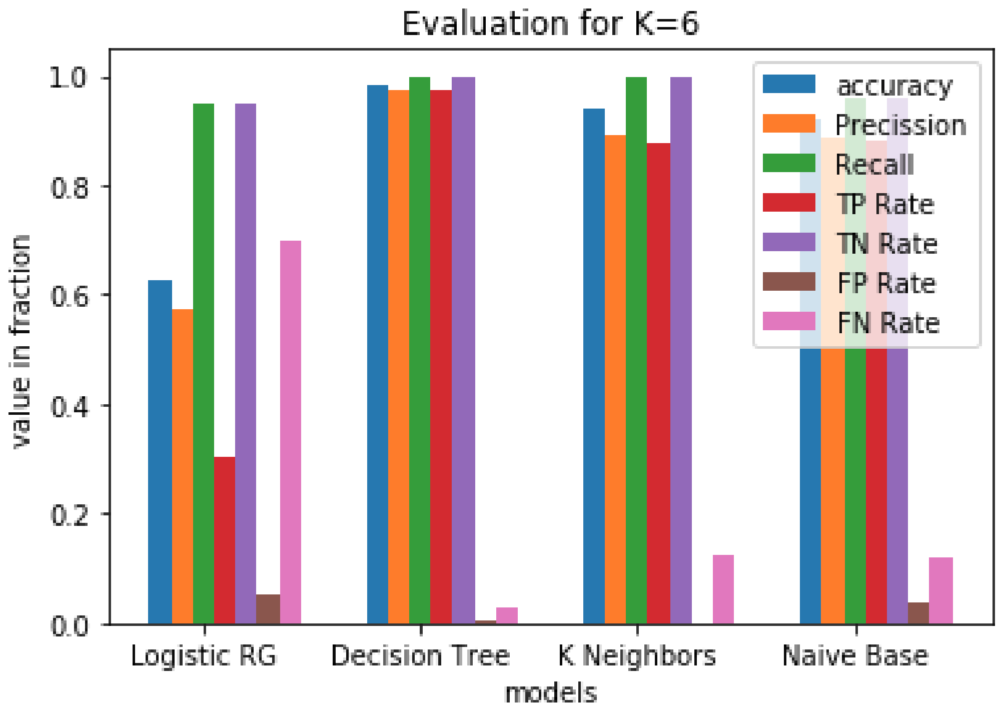
```
[[2235  311]
 [   2 2532]]
0.9383858267716535
              precision    recall  f1-score   support

           0       1.00      0.88      0.93      2546
           1       0.89      1.00      0.94      2534

   micro avg       0.94      0.94      0.94      5080
   macro avg       0.94      0.94      0.94      5080
weighted avg       0.94      0.94      0.94      5080
```

Naive Bayes

```
from sklearn.naive_bayes import GaussianNB as GB
model_b6=GB().fit(Xtrn,Ytrn)
predict_b6=model_b6.predict(Xtst)
m6b=cm(Ytst,predict_b6)
print(m6b)

print(acc(Ytst,predict_b6))
print(cr(Ytst,predict_b6))
```

```
[[2241  305]
 [ 101 2433]]
0.9200787401574804
              precision    recall  f1-score   support

           0       0.96      0.88      0.92      2546
           1       0.89      0.96      0.92      2534

   micro avg       0.92      0.92      0.92      5080
   macro avg       0.92      0.92      0.92      5080
weighted avg       0.92      0.92      0.92      5080
```

- Here we will evaluate the values of the matrices and accuracy, recall, precession, TPR, TNR, FPR & HNR in table form and visualize in bar plot for better understanding



	Logistic RG	Decision Tree	K Neighbors	Naive Base
accuracy	0.623106	0.988782	0.935839	0.926983
Precission	0.571326	0.980031	0.890519	0.894756
Recall	0.948187	0.997609	0.992029	0.965723
TP rate	0.305988	0.980171	0.881026	0.889191
TN rate	0.948187	0.997609	0.992029	0.965723
FP rate	0.051813	0.002391	0.007971	0.034277
FN rate	0.694012	0.019829	0.118974	0.110809

➢ For K=7

Logisctic Regression

```
model_l7=LG().fit(Xtrn,Ytrn)
predict_l7=model_l7.predict(Xtst)
m7l=cm(Ytst,predict_l7)
print(m7l)

print(acc(Ytst,predict_l7))
print(cr(Ytst,predict_l7))
```

```
[[ 772 1774]
 [ 128 2406]]
0.6255905511811024
              precision    recall  f1-score   support

           0       0.86      0.30      0.45      2546
           1       0.58      0.95      0.72      2534

   micro avg       0.63      0.63      0.63      5080
   macro avg       0.72      0.63      0.58      5080
weighted avg       0.72      0.63      0.58      5080
```

Decision Tree

```
model_t7=dtc(criterion='entropy',random_state=1)
model_t7.fit(Xtrn,Ytrn)
predict_t7=model_t7.predict(Xtst)
m7t=cm(Ytst,predict_t7)
print(m7t)

print(acc(Ytst,predict_t7))
print(cr(Ytst,predict_t7))
```

```
[[2481   65]
 [   6 2528]]
0.9860236220472441
              precision    recall  f1-score   support

           0       1.00      0.97      0.99      2546
           1       0.97      1.00      0.99      2534

   micro avg       0.99      0.99      0.99      5080
   macro avg       0.99      0.99      0.99      5080
weighted avg       0.99      0.99      0.99      5080
```

K neighbors

```python
Xtrn=SC().fit_transform(Xtrn)
Xtst=SC().fit_transform(Xtst)

model_n7=KNC(n_neighbors=83,p=2,metric='euclidean')
model_n7.fit(Xtrn,Ytrn)
predict_n7=model_n7.predict(Xtst)
m7n=cm(Ytst,predict_n7)
print(m7n)

print(acc(Ytst,predict_n7))
print(cr(Ytst,predict_n7))
```

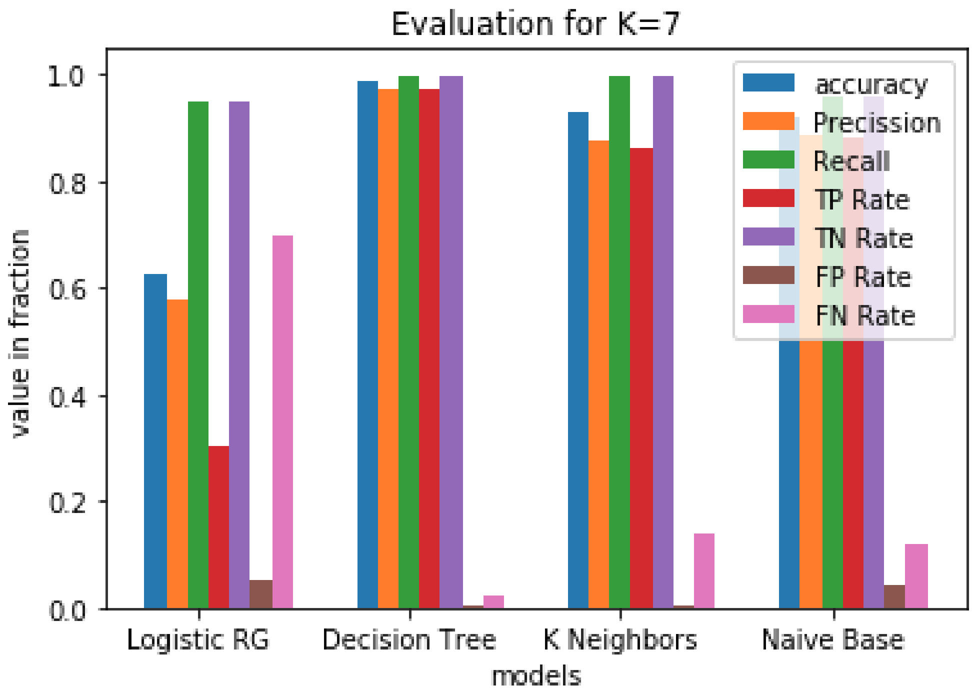
```
[[2193  353]
 [   4 2530]]
0.9297244094488188
              precision    recall  f1-score   support

           0       1.00      0.86      0.92      2546
           1       0.88      1.00      0.93      2534

   micro avg       0.93      0.93      0.93      5080
   macro avg       0.94      0.93      0.93      5080
weighted avg       0.94      0.93      0.93      5080
```

Naive Bayes

```python
from sklearn.naive_bayes import GaussianNB as GB
model_b7=GB().fit(Xtrn,Ytrn)
predict_b7=model_b7.predict(Xtst)
m7b=cm(Ytst,predict_b7)
print(m7b)

print(acc(Ytst,predict_b7))
print(cr(Ytst,predict_b7))
```

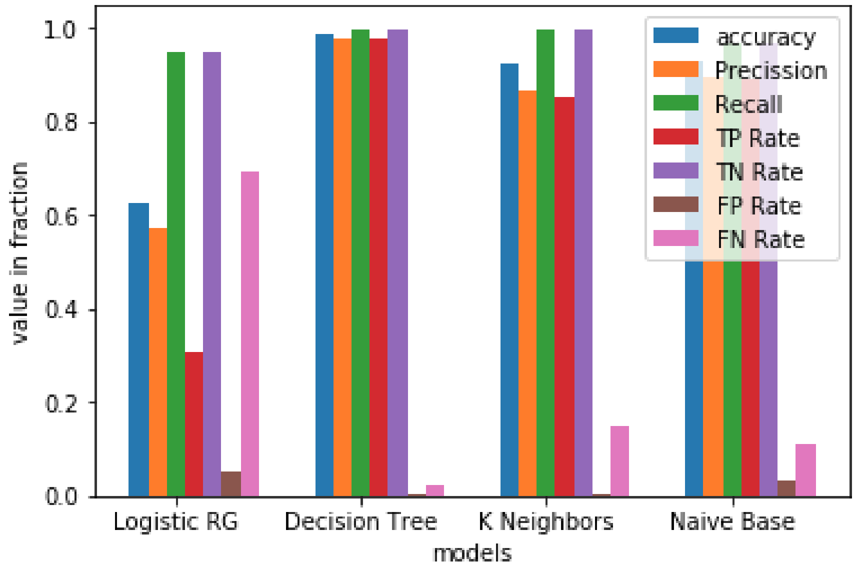
```
[[2241  305]
 [ 102 2432]]
0.9198818897637795
              precision    recall  f1-score   support

           0       0.96      0.88      0.92      2546
           1       0.89      0.96      0.92      2534

   micro avg       0.92      0.92      0.92      5080
   macro avg       0.92      0.92      0.92      5080
weighted avg       0.92      0.92      0.92      5080
```

- Here we will evaluate the values of the matrices and accuracy, recall, precession, TPR, TNR, FPR & HNR in table form and visualize in bar plot for better understanding



	Logistic RG	Decision Tree	K Neighbors	Naive Base
accuracy	0.623106	0.989766	0.924818	0.926786
Precission	0.571326	0.981954	0.870687	0.894426
Recall	0.948187	0.997609	0.995616	0.965723
TP rate	0.305988	0.982115	0.855754	0.888802
TN rate	0.948187	0.997609	0.995616	0.965723
FP rate	0.051813	0.002391	0.004384	0.034277
FN rate	0.694012	0.017885	0.144246	0.111198

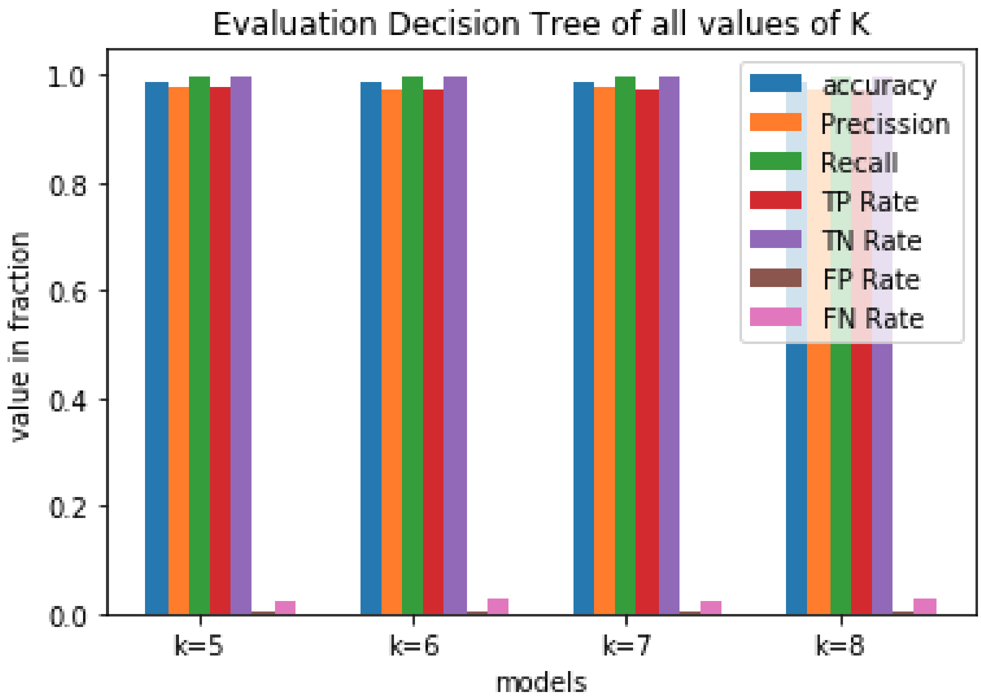
➢ For K=8 (All features)



	Logistic RG	Decision Tree	K Neighbors	Naive Base
accuracy	0.623106	0.987601	0.921866	0.926786
Precission	0.571326	0.977734	0.866667	0.894426
Recall	0.948187	0.997609	0.994819	0.965723
TP rate	0.305988	0.977838	0.850700	0.888802
TN rate	0.948187	0.997609	0.994819	0.965723
FP rate	0.051813	0.002391	0.005181	0.034277
FN rate	0.694012	0.022162	0.149300	0.111198

- Here we can see for any value of K the ‘Decision Tree Model” is giving the best result better accuracy and Precision and Recall and maximum value of TPR and TNR value more than 2 other models and FPR and FNR are less remarkably in the ‘Decision Tree Model’.
- So We decide Decision tree as our final model but to decide which K value is giving the best result we have to evaluate for which value of K the Decision tree is giving the best results.

➢ The evaluation of all the decision tree for all K values are shown down here by a table and as well as by a plot too



	k=5	k=6	k=7	All Features
accuracy	0.989963	0.988782	0.989766	0.987601
Precission	0.982339	0.980031	0.981954	0.977734
Recall	0.997609	0.997609	0.997609	0.997609
TP rate	0.982504	0.980171	0.982115	0.977838
TN rate	0.997609	0.997609	0.997609	0.997609
FP rate	0.002391	0.002391	0.002391	0.002391
FN rate	0.017496	0.019829	0.017885	0.022162

- Here it is almost impossible to determine the best value of K from the bar plot because the difference is really very less
- So we take help of the table here K=5 and k=7 have best Accuracy, precession and recall and lowest FP and FN rate
- As Recall and TP rate is a little bigger in k=5 than in k=7 and FN rate is a little less in k=5 than k=7 that’s why We take ‘5’ as the final value of K

Ensemble Learning: -

The model which has been created are supposed to be weak learner as those work as standalone models. Where may be biasness or some kind of problem in in a model. By Enable learning we can take multiple model (same type or different type) to create a strong model which is supposed to give better result than a standalone model. Because in this case the drawback of a model is usually recovered by another models which are being ensemble together for creating a stronger model.

We have 3 ways for ensemble learning-

1. Bagging
2. Boosting
3. Voting

Here we will apply Bagging and voting to see if any performance improvement is possible or not.
As we got best accuracy for k value 5, here also we will apply bagging and voting, taking k as 5

Bagging (Random Forest) [k=5]

```
seed=1
num_trees=100
max_features=3
Kfold=model_selection.KFold(n_splits=10,random_state=seed)
model=RFC(n_estimators=num_trees,max_features=max_features)
results=model_selection.cross_val_score(model,x,y,cv=Kfold)
print(results.mean())
```

```
0.9861786605168443
```

Voting [for k=5]

```
estimators=[]
model1=LG()
estimators.append(('logistic',model1))
model2=DTC()
estimators.append(('tree',model2))
model3=KNC()
estimators.append(('cneifgbor',model3))
model4=GB()
estimators.append(('bayes',model4))


ensemble=VC(estimators)
results=model_selection.cross_val_score(ensemble,x,y,cv=Kfold)
print(results.mean())
```

```
0.8227512271866029
```

Here we can see Bagging method is giving us almost same accuracy as the standalone decision tree gave us which was our best accuracy.

So here we can conclude, though our standalone models are supposed to be weak models, in our case they are proven as strong as the ensemble learning model.

FUTURE SCOPE OF IMPROVEMENT

- This model could be implemented in banking software to determine status of a credit card application.
- The whole model could be implemented in each banks website as an auto chat-bot which will ask the user to input required information and tell whether his/her application for credit card would be approved or not.
    - The training dataset would be taken from individual banks to implement into each banks software or website auto chat-bot
- This model could be used in android or ios banking app also where user can give input according to the requirements of our model and our model will predict if the application of credit card would be approved or not.
- As the model will gain more experience with time as the model will be used to predict data, after times the performance of this model will be increased automatically.

Certificate

This is to certify that Ms Naiwrita Chowdhury of Guru Nanak Institute of Technology, registration number: 171430110146 (2017-2018), has successfully completed a project on Credit ‘Card Approval Prediction by Machine Learning’ using Python under the guidance of Mr Kaushik Ghosh.

Mr Kaushik Ghosh

Globsyn Finishing School

Certificate

This is to certify that Mr Pritimoy Sarkar of Guru Nanak Institute of Technology, registration number: 171430110066 of 2017-2018, has successfully completed a project on Credit ‘Card Approval Prediction by Machine Learning’ using Python under the guidance of Mr Kaushik Ghosh.

Mr Kaushik Ghosh

Globsyn Finishing School

Certificate

This is to certify that Mr Sandipan Sau of Guru Nanak Institute of Technology, registration number: 171430110081 (2017-2018), has successfully completed a project on Credit 'Card Approval Prediction by Machine Learning' using Python under the guidance of Mr Kaushik Ghosh.

Mr Kaushik Ghosh

Globsyn Finishing School

Certificate

This is to certify that Ms Snighdha Mazumdar of Guru Nanak Institute of Technology, registration number: 171430110095 (2017-2018), has successfully completed a project on Credit 'Card Approval Prediction by Machine Learning' using Python under the guidance of Mr Kaushik Ghosh.

Mr Kaushik Ghosh

Globsyn Finishing School

Certificate

This is to certify that Mr Snehandu Chanda of Guru Nanak Institute of Technology, registration number: 171430110094 (2017-2018), has successfully completed a project on Credit ‘Card Approval Prediction by Machine Learning’ using Python under the guidance of Mr Kaushik Ghosh.

Mr Kaushik Ghosh

Globsyn Finishing School

Certificate

This is to certify that Ms Triyasha Kundu of Guru Nanak Institute of Technology, registration number: 171430110121 (2017-2018), has successfully completed a project on Credit 'Card Approval Prediction by Machine Learning' using Python under the guidance of Mr Kaushik Ghosh.

Mr Kaushik Ghosh