Stander Customer Transaction Prediction

Project Report

Background:

At Santander, mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals. Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as:

- is a customer satisfied?
- Will a customer buy this product?
- Can a customer pay this loan?

Problem Statement:

We need to predict the target variable in the test dataset with the help of building ML algorithms by using train dataset. Train data set contains target variable which has two observations **0** indicates no which means customer does not make transactiono**1** indicates yes which means customer makes transactions irrespective to amount.

Data Details:

Provided with an anonymized dataset containing 200 numeric feature variables, the binary target column, and a string ID_code column.

File Descriptions:

train.csv – the training set .

test.csv - the test set.

Problem Analysis:

This is a binary classification problem under supervised machine learning algorithm. The task is to predict the value of target column in the test set.

Introduction

Machine Learning is a study of computer algorithms that improve automatically through experience and by the use of data. It is a subfield of artificial intelligence. Machine Learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. These ML algorithms are widely used in email spam filtering, computer vision, sales forecasting … etc. where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

In addition, machine learning is closely related to computational statistics, which focuses on making predictions using computers; but not all machine learning is statistical learning. The study of mathematical optimization delivers methods, theory and application domains to field of machine learning. And also, Data mining is a related field of study, focusing on EDA through unsupervised learning.

Machine Learning approaches are traditionally divided into four broad categories, depending on the nature of the signal or feedback available to the learning system. There are,

- **Supervised Learning** is a type of train the machine learning algorithms by giving an input features and target in train dataset for to develop a machine learning model, to make predictions.
- **Unsupervised Learning** is a type of train the machine learning algorithm by giving only input features to find the target while in training and deploying the machine learning model.
- **Semi Supervised Learning** is an approach to machine learning that combines a small amount of labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised learning and supervised learning. It is a special instance of weak supervision.
- **Reinforcement Learning** is a model free train the machine learning algorithms by giving an actions and space in environment and agent objects for to make right predictions in real time.

Scikit-Learn is a python module for machine learning built top on Scipy module and is distributed under the 3-Clause BSD License. It is simple and efficient tools for predictive data analysis. It is accessible to everyone, and reusable in various contexts. It was built on Numpy, Scipy and Matplotlib.

Scikit-Learn module mainly used for the Regression, Classification, Clustering, Dimensionality Reduction, Model Selection and Preprocessing of historical data to predict the futures, and classify the data.

System Requirements:

The requirements of the CPU or laptop for to run the project. The following are important requirement to run this project. They are:

1. Laptop or CPU with above i3 8th gen processor.
2. RAM not less than 8GB
3. Python 3.8.8 64 bit
4. R 4.0.5
5. Windows 10 64bit
6. Required python Packages and R libraries are listed in `Requirement.txt` file.

Preprocessing Data:

Santander Customer Transaction Prediction dataset is downloaded from the link given in the project problem statement. It contains the csv file with name of "test.csv" and "train.csv" and it extracted to the working directory as df_train and df_test.in pyhton and train and test in R. The project folder is open with jupyter lab for python coding and with R studio for R coding and create new file as Santander customer Trasaction prediction.py and Santander customer transaction prediction.R.

After that, import a pandas module as "pd", by simply typing as "import pandas as pd" in Santander cunstomer transaction predictionpy. Then load the dataset in python by using pandas data frame as df_train and df_test. In R as train and test with a code as follows as:

Python code :-

```
#load data
df_train=pd.read_csv("train.csv")
df_test=pd.read_csv("test.csv")
```

R code :-

```
train=read.csv("train.csv")
test=read.csv("test.csv")
```

Above code will display the table format of data which is convert from json file to pandas Dataframe. Now the right time to check the null values present in dataframe or not by following syntax.

Pyhton code:-

```
df_train.info
df_test.info
df_train.isna().sum().sum()
df_test.isna().sum().sum()
```

R code:-

```
dim(train)
dim(test)
anyNA(train)
anyNA(test)
```

Above line code will display the list, with explain of dataframe of each column contains a dtype and null values count. And next line code will display the total number of null values present in each columns.

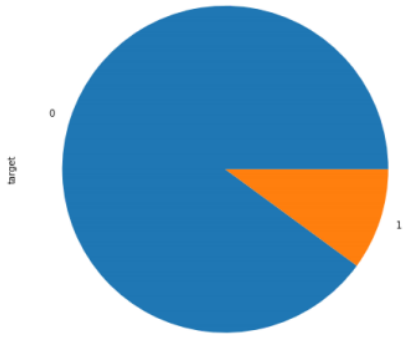
According to the project statement, in description dataset is given information as where the column target as target. So, the target column need to count the values to perform a classification or regression by following syntax.

```
df_train.target.value_counts()
```

Above the code will display the counted of values, which explains the values was represent the number of times repeated in the column. So, the limited values are repeated and repeated, then it is known to go for classification else go for regression where the target values are unique or almost unique.

The below code will display a pie chart of target class.

```
#plotting pie chart for target class
df_train['target'].value_counts().plot(kind='pie', figsize=(5,5))
```



Exploratory Data Analysis:

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations.

The most important code line in python, is that df_train.info(), df_train.describe(), df_train.shape, df_train.target.value_counts() and in R str(train), dim(str), class(train), summary(train) are the initial steps to perform the Exploratory Data Analysis.

Chauvenet"s Criterion

Chauvenet's criterion is a way to identify outliers. The method works by creating an acceptable band of data around the mean, specifying any values that fall outside that band should be eliminated. The formula to calculate the band is:

$$\left(1-\frac{1}{2n}\right)$$

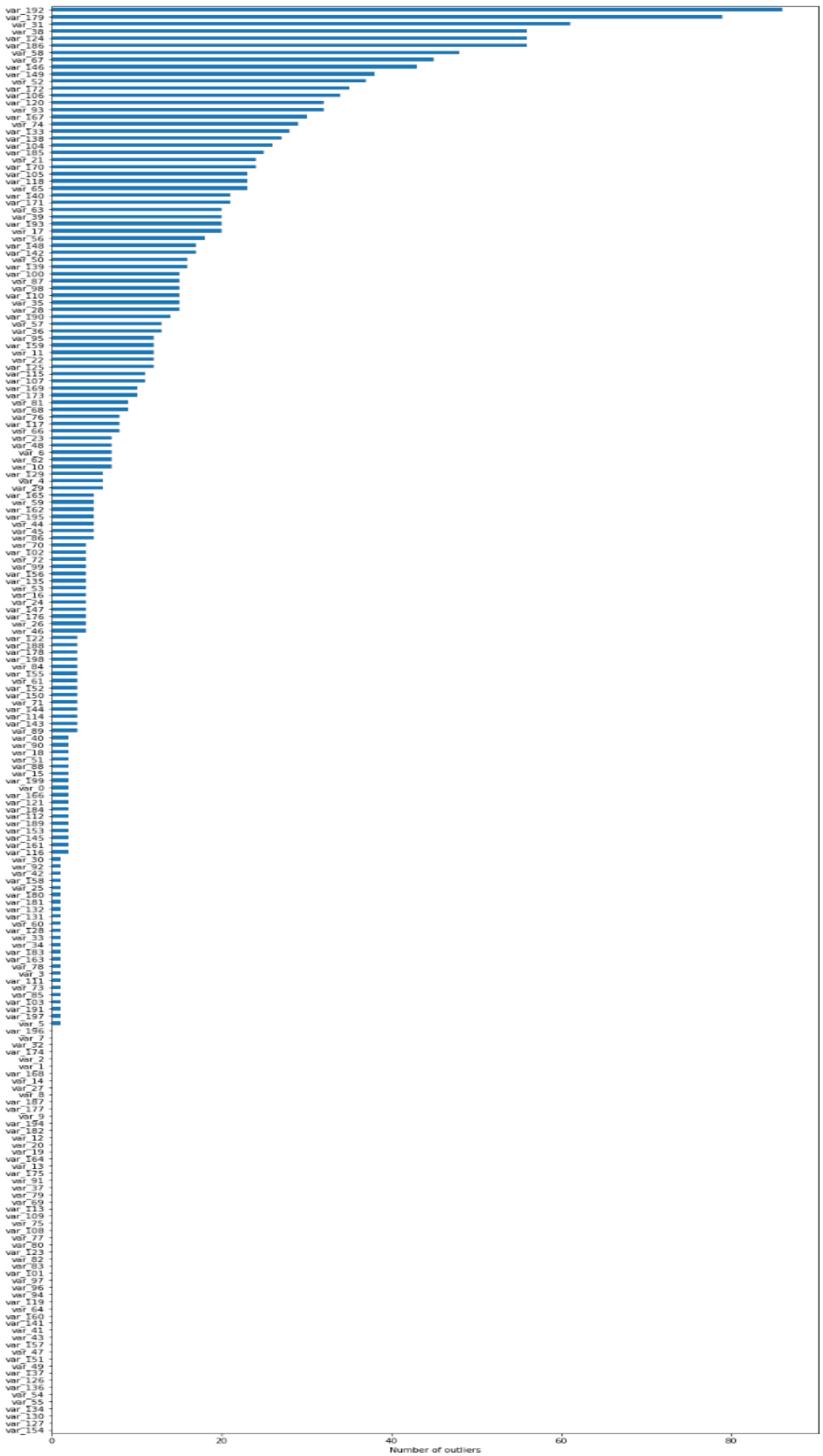
Where n is the sample size

This procedure assumes data is normally distributed.

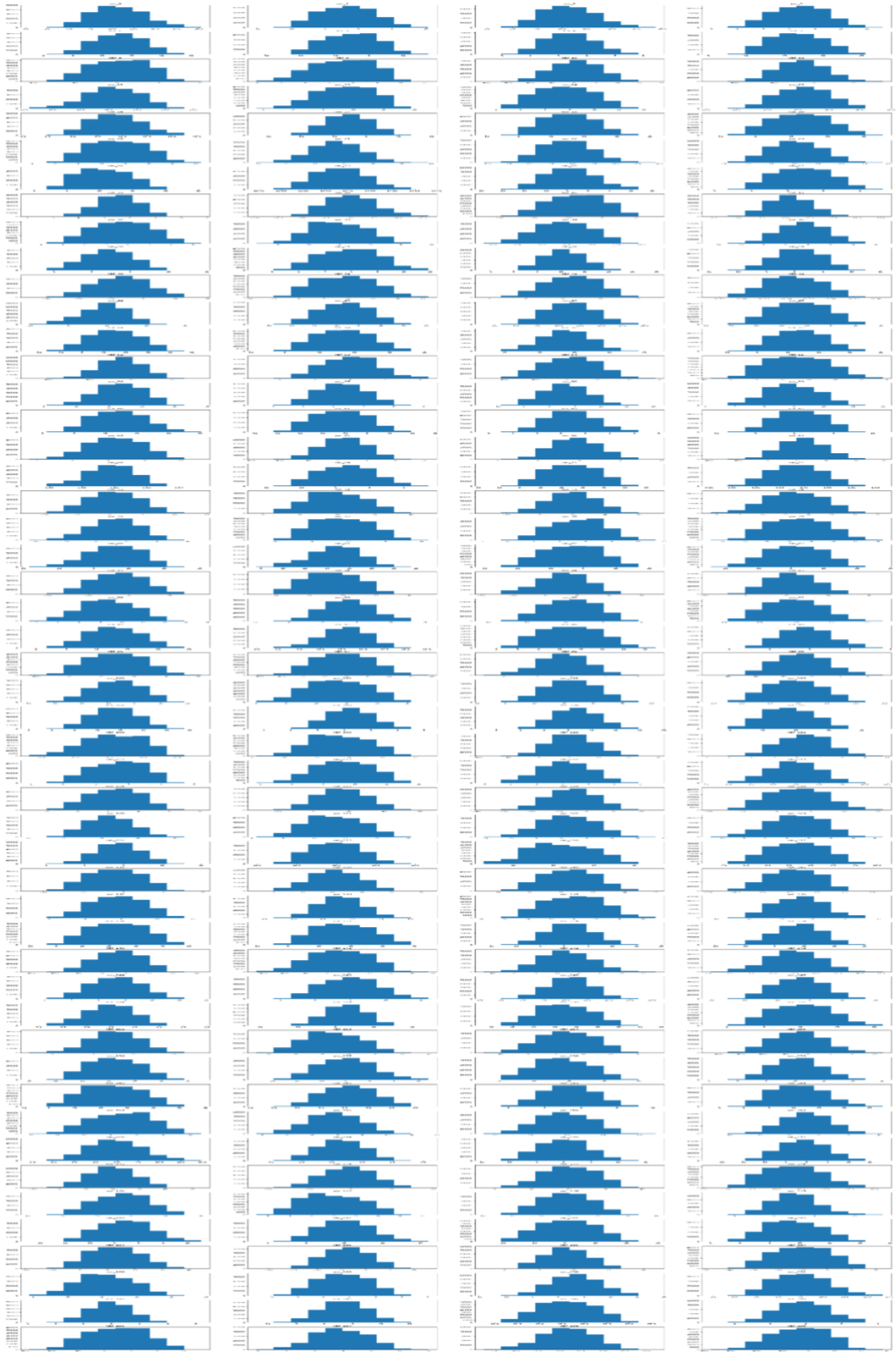
Python code:-

```
def chauvenet(array):
    mean = array.mean()           # Mean of incoming array
    stdv = array.std()            # Standard deviation
    N = len(array)                # Lenght of incoming array
    criterion = 1.0/(2*N)         # Chauvenet's criterion
    d = abs(array-mean)/stdv      # Distance of a value to mean in stdv's
    prob = erfc(d)                # Area normal dist.
    return prob < criterion       # Use boolean array outside this function
numerical_features=df_train.columns[2:]
#outliers in each variable in train data
train_outliers = dict()
for col in [col for col in numerical_features]:
    train_outliers[col] = df_train[chauvenet(df_train[col].values)].shape[0]
train_outliers = pd.Series(train_outliers)
#Plotting outliers in a barplot
train_outliers.sort_values().plot(figsize=(14, 40), kind='barh').set_xlabel('Number of outliers');
#draw histograms of numeric data in training set
print("Distribution of Columns")
plt.figure(figsize=(40,200))
for i,col in enumerate(numerical_features):
    plt.subplot(50,4,i+1)
    plt.hist(train[col])
    plt.title(col)
```

Above code is given as the bar plot as below(showing outliers in train dataset):


var_192
var_179
var_31
var_38
var_124
var_186
var_58
var_67
var_146
var_149
var_52
var_172
var_106
var_120
var_93
var_167
var_74
var_133
var_138
var_104
var_185
var_21
var_170
var_105
var_118
var_65
var_140
var_171
var_63
var_39
var_193
var_17
var_56
var_148
var_142
var_50
var_139
var_100
var_87
var_98
var_110
var_35
var_28
var_190
var_57
var_36
var_95
var_159
var_11
var_22
var_125
var_115
var_107
var_169
var_173
var_81
var_68
var_76
var_117
var_66
var_23
var_48
var_6
var_62
var_10
var_129
var_4
var_29
var_165
var_59
var_162
var_195
var_44
var_45
var_86
var_70
var_102
var_72
var_89
var_156
var_135
var_53
var_16
var_24
var_147
var_176
var_26
var_46
var_122
var_188
var_178
var_198
var_84
var_155
var_61
var_152
var_150
var_71
var_144
var_114
var_143
var_89
var_40
var_90
var_18
var_51
var_88
var_15
var_199
var_0
var_166
var_121
var_184
var_112
var_189
var_153
var_145
var_161
var_116
var_30
var_92
var_42
var_158
var_25
var_180
var_181
var_132
var_131
var_60
var_128
var_33
var_34
var_183
var_163
var_78
var_3
var_111
var_73
var_85
var_103
var_191
var_197
var_5
var_196
var_7
var_32
var_174
var_2
var_1
var_168
var_14
var_27
var_8
var_187
var_177
var_9
var_194
var_182
var_12
var_20
var_19
var_164
var_13
var_175
var_91
var_37
var_79
var_69
var_113
var_109
var_75
var_108
var_77
var_80
var_123
var_82
var_83
var_101
var_97
var_96
var_94
var_119
var_64
var_160
var_141
var_41
var_43
var_157
var_47
var_151
var_49
var_137
var_126
var_136
var_54
var_55
var_134
var_130
var_127
var_154
0
20
40
60
80
Number of outliers

Histogram to check the distribution of the independent variables:



All variables follow fairly normal distribution.

R code:-

```r
# Create a generic function for generating distribution & box plots
buildplot =function(x, type = 'dist'){
  variablenames = colnames(x)
  temp = as.numeric(1)
  for(i in seq(8, dim(x)[2], 8)){
    par(mar = c(2, 2, 2, 2))
    par(mfrow = c(4, 2))
    for (j in variablenames[temp:i]){
      if(type == 'box'){
        jpeg(filename = paste('boxplot', j, '.jpg', sep = ' '), width = 1080, height = 1080)
        boxplot(x[[j]], main = j, col = 'grey' )
        dev.off()
      }

      else{
        jpeg(filename = paste('dplot', j, '.jpg', sep = ' '), width = 1080, height = 1080)
        plot(density(x[[j]]), main = j, col = 'red' )
        dev.off()
      }
    }
    temp = i + 1
  }
}
# Distribution plot for training & test data
buildplot(x)
buildplot(df.test)
```

Feature Engineering:

Feature engineering is the process of using domain knowledge to extract features from raw data. These features can be used to improve the performance of machine learning algorithms. Feature engineering can be considered as applied machine learning itself.

Standardizing a dataset involves rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1. This can be thought of as subtracting the mean value or centering the data. This can be thought of as subtracting the mean value or centering the data. Standardize of a dataset is done using the scikit-learn object StandardScaler.

Pipeline of transforms with a final estimator. Sequentially apply a list of transforms and a final estimator. Intermediate steps of the pipeline must be 'transforms', that is, they must implement fit and transform methods. The final estimator only needs to implement fit. The transformers in the pipeline can be cached using memory argument. The main parameter of a pipeline we'll be working on is '**steps**'.

In R we are doing scaling to bring all features into a single scale.

Python code:-

```python
steps = [("Standarad Scalar", StandardScaler(with_mean=True)),
             ("PCA", PCA(n_components=10))
            ]

    pipline = Pipeline(steps=steps)
    pXtrain = pipline.fit_transform(X_train)
```

R code:-

```r
training[,2:201] = scale(training[,2:201])
df.test[,1:200] = scale(training[,1:200])
```

Model Selection:

Model selection is the process of selecting one final machine learning model from among a collection of candidate machine learning models for a training dataset. It can be applied both across

different types of models (e.g. logistic regression, SVM, RandomForest, etc.) and across models of the same type configured with different model hyper parameters (e.g. different kernels in an SVM).

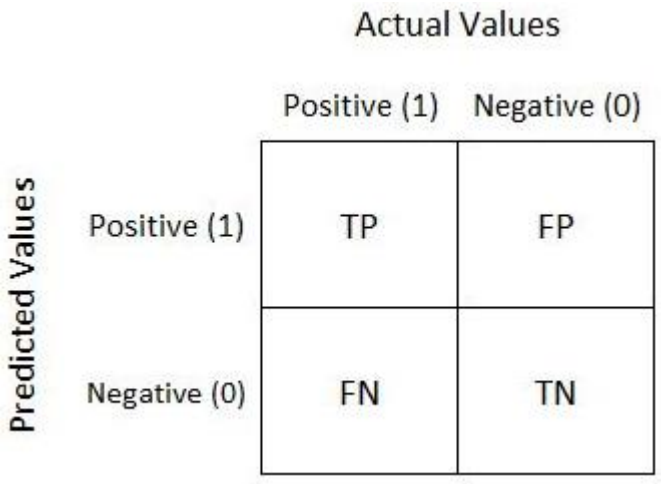
Logistic Regression, SVC, GaussianNB, RandomForest Classifier Machine Learning alogrithms are used to train and test the data in python, whereas, Logistic Regression, RandomForest, GaussianNB to make a Machine Learning Model in this project.

Metrics:

This is a classification problem and we need to understand confusion matrix for getting ecaluation metrics.

It is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.

It extremely useful for measuring Recall, Precision, Specificty, Accuracy and most importantly **AUC-ROC curve.**



True Positive: You predicted positive and it's true.

True Negative: You predicted negative and it's true.

False Positive:(Type 1 Error) You predicted positive and it's false.

False Negative: (Type 2 Error) You predicted negative and it's false.

Based on confusion matrix we have following evaluation metrics-

Recall- Out of all the positive classes, how much we predicted correctly. It should be high as possible.

Precision- Out of all the classes, how much we predicted correctly. It should be high as possible.

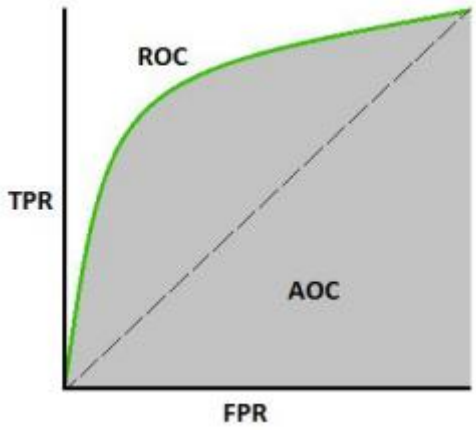
F-measure- It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

Another important measure is AUC-ROC Score.

It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as AUROC (Area Under the Receiver Operating Characteristics).

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability.

It tells how much model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.



An excellent model has AUC near to the 1 which means it has good measure of separability. A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity.

Random-Over Sampling:

Over-Sampling increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample.

- Unlike under sampling this method leads to no information loss.
- Outperforms under sampling

Before Machine learning model build up, data was split into Train and Test data. In this project, test data size has been taken as 25% of whole data.

Python code:-

```
124    Xtrain, Xtest, ytrain, ytest = train_test_split(pXtrain, y_train, test_size=0.25, random_state=42 )
```

R code:-

```
100  # create a list of 75% of the rows in the train dataset we can use for training
101  validation_index = createDataPartition(training$target, p=0.75, list=FALSE)
```

Logistic Regression:

Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist. In regression analysis, logistic regression (or logit regression) is estimating the parameters of a logistic model (a form of binary regression). Mathematically, a binary logistic model has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled "0" and "1".

An explanation of logistic regression can begin with an explanation of the standard logistic function. The logistic function is a sigmoid function, which takes any real input t, and outputs a value

between zero and one. For the logit, this is interpreted as taking input log-odds and having output probability. The standard logistic function $\sigma : R \rightarrow (0, 1)$ is defined as follows:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

In this project, the Scikit-learn module is used to build a model in python for classification of target variable as yes or no. The following is a R and python code for Logistic Regression Model.

R code:-

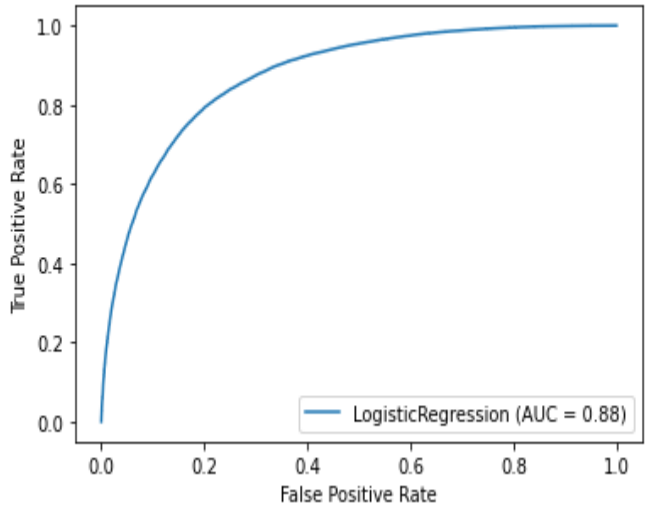
```r
print("Logistic Regression:- ")
#Logistic Regression
#Setting seed
set.seed(2)
#fitting model
fit.glm = glm(target ~., data = training, family = binomial(link ="logit" ))

# print(fit.glm)
predictions.glm = predict(fit.glm, validation)
predictions.glm = as.factor(ifelse(predictions.glm < 0.5, 'zero', 'one'))
#calculate confusion matrix
cm = ConfusionMatrix(predictions.glm, validation$target)
glm = list()
#calculate Scores
glm[['accuracy']] = Accuracy(predictions.glm, validation$target) * 100
glm[['fnr']] = (cm['one','zero'] * 100)/(cm['one','zero']+cm['one', 'one'])
#plot ROc Curve
glm[['auc']] = roc.curve(predictions.glm, validation$target)$auc
glm[['ConfusionMatrix']] = cm
#print scores and Confusion matrix
print(glm)
```

Python code:-

```python
print("Logistic Regression:- ")
#Logistic Regression
lr = LogisticRegression()
lr.fit(Xtrain, ytrain)

lr_ypred = lr.predict(Xtest)

#print Classification report
print(classification_report(ytest, lr_ypred)+"\n\n")
#print Accuracy
print("AUC Score:- %.3f \n\n" % roc_auc_score(ytest, lr_ypred))

#print confusion matrix
cnf_mtx(ytest, lr_ypred, "LR-Confusion Matrix.png")
#plot ROC Curve
plot_roc_curve(lr, Xtest, ytest)
#axis labels
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
#show plot
plt.show()
```

Above code given output as below:

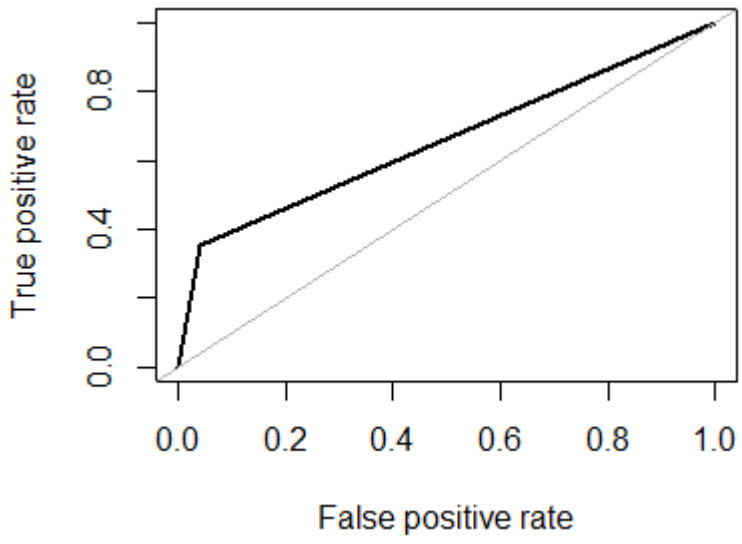
Python-ROC Curve:-



Python-Classification Report:-

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

           0       0.80      0.79      0.79     45100
           1       0.79      0.80      0.80     44851

    accuracy                           0.79     89951
   macro avg       0.79      0.79      0.79     89951
weighted avg       0.79      0.79      0.79     89951


AUC Score:- 0.795


----confusion matrix-----
[[35565  9535]
 [ 8926 35925]]
```

R-ROC curve:-



R-Classification Report :-

```
$accuracy
[1] 90.0118

$auc
[1] 0.8211249

$fnr
[1] 99.08439

$confusionMatrix
        y_pred
y_true   zero   one
  zero 44959    16
  one   4978    46
```

RandomForest Classifier:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multiple of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean/average prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.587–588 random forests generally outperform decision trees, but their accuracy is lower than gradient boosted trees. However, data characteristics can affect their performance.

The first algorithm for random decision forests was created in 1995 by Tin Kam Ho using the random subspace method, which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg.

An extension of the algorithm was developed by Leo Breiman and Adele Cutler, who registered "Random Forests" as a trademark in 2006 (as of 2019, owned by Minitab, Inc.). The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho and later independently by Amit and Geman in order to construct a collection of decision trees with controlled variance.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

In this project, the Scikit-learn module is used to build a model in python and randomForest library in R for classification of target variable as o and 1. The following are python and R code for RandomForest Classifier Model.

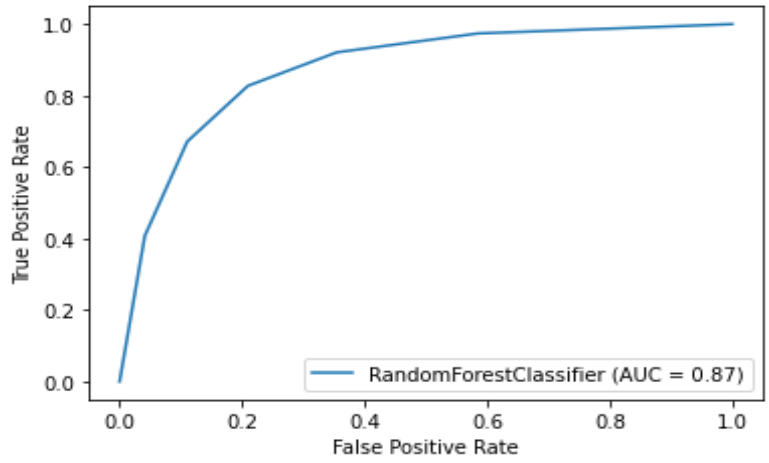
Python code:-

```
    print("RandomForest Classifier:- ")
    #RandomForest Classifer
    rnclf = RandomForestClassifier(n_estimators=5)
    rnclf.fit(Xtrain, ytrain)
    rnclf_ypred = rnclf.predict(Xtest)

    #Print Classification Reprt
    print(classification_report(ytest, rnclf_ypred)+"\n\n")
    #Print Accuracy
    print("AUC Score:- %.3f \n\n" % roc_auc_score(ytest, rnclf_ypred))

    #Print Confusion Matrix
    cnf_mtx(ytest, rnclf_ypred, "Rnclf-ConfusionMatrix.png")
    #plot Roc Curve
    plot_roc_curve(rnclf, Xtest, ytest)
    # axis labels
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    #show plot
    plt.show()
```

R code:-

```
print("RandomForest:- ")
#RandomForesst
#setting seed
set.seed(2)
#fit the model
fit.rf = randomForest(target ~., data = training, ntree = 20, mtry = 10)
# print(fit.xgb)
predictions.rf = predict(fit.rf, validation)
#calculate confusion matrix
cm = ConfusionMatrix(predictions.rf, validation$target)
rf = list()
#calculate scores
rf[['accuracy']] = Accuracy(predictions.rf, validation$target) * 100
#plot Roc Curve
rf[['auc']] = roc.curve(predictions.rf, validation$target)$auc
rf[['fnr']] = (cm['one','zero'] * 100)/(cm['one','zero']+cm['one', 'one'])
rf[['ConfusionMatrix']] = cm
#print scores and confusion matrix
print(rf)
```

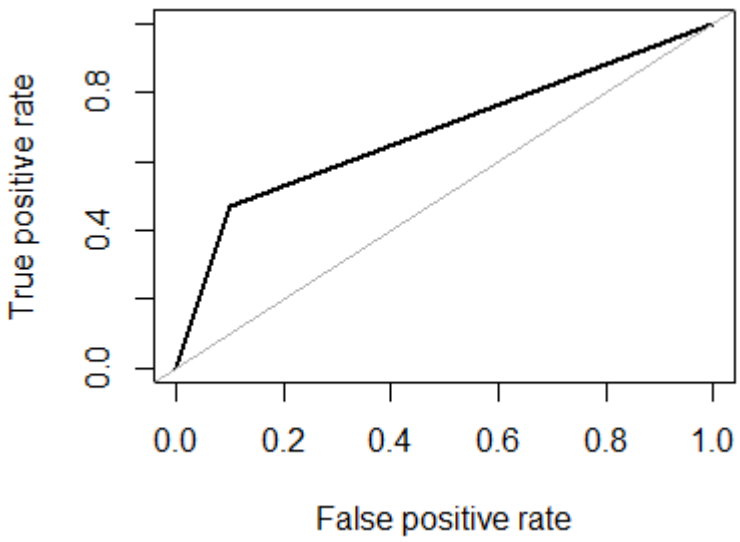
Python-ROC curve:-



Python-Classification Report

```
RandomForest Classifier:-
              precision    recall  f1-score   support

           0       0.82      0.79      0.81     45100
           1       0.80      0.83      0.81     44851

    accuracy                           0.81     89951
   macro avg       0.81      0.81      0.81     89951
weighted avg       0.81      0.81      0.81     89951


AUC Score:- 0.809


----confusion matrix-----
[[35788  9312]
 [ 7831 37020]]
```

R-ROC curve



R-Classification Report:-

```
$accuracy
[1] 91.25383

$fnr
[1] 81.92675

$auc
[1] 0.8475568

$confusionMatrix
        y_pred
y_true    one  zero
  zero    257 44718
  one     908  4116
```

SVC:

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. Developed at AT&T Bell Laboratories by Vladimir Vapnik with colleagues (Boser et al., 1992, Guyon et al., 1993, Vapnik et al., 1997), SVMs are one of the most robust prediction methods, being based on statistical learning frameworks or VC theory proposed by Vapnik and Chervonenkis (1974) and Vapnik (1982, 1995). Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

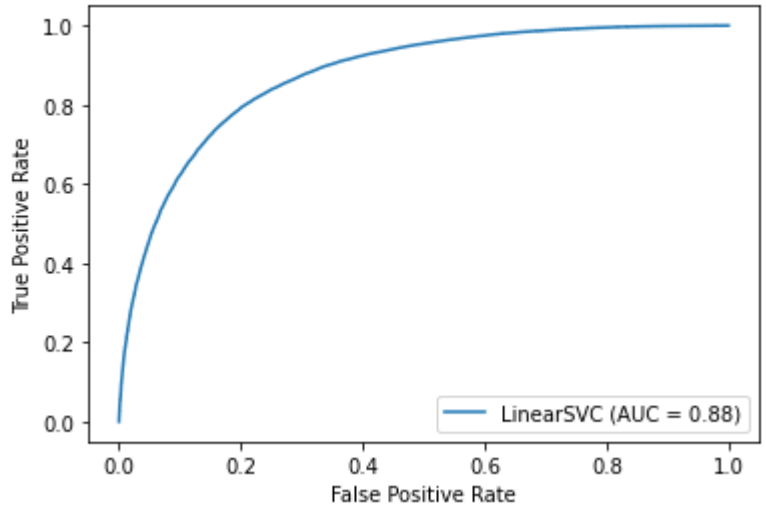
$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i - b)\right)\right] + \lambda\|\mathbf{w}\|^2.$$

In this project, the Scikit-learn module is used to build a model in python for classification of target as 0 and 1. The following is a python code for SVC Model.

Python code:-

```
    print("LinearSVC Classifier:- ")
    #Linear SVC
    svc = LinearSVC()
    svc.fit(Xtrain, ytrain)
    svc_ypred = svc.predict(Xtest)

    #print classification report
    print(classification_report(ytest, svc_ypred)+"\n\n")
    #print Accuracy
    print("AUC Score:- %.3f \n\n" % roc_auc_score(ytest, svc_ypred))

    #print confusion matrix
    cnf_mtx(ytest, svc_ypred, "L SVC-ConfusionMatrix.png")
    #plot ROC curve
    plot_roc_curve(svc, Xtest, ytest)
    #axis labels
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    #show plot
    plt.show()
```

Python-ROC curve:-



Python-Classification Report:-

```
LinearSVC Classifier:-
              precision    recall  f1-score   support

           0       0.80      0.79      0.79     45100
           1       0.79      0.81      0.80     44851

    accuracy                           0.80     89951
   macro avg       0.80      0.80      0.80     89951
weighted avg       0.80      0.80      0.80     89951


AUC Score:- 0.796


----confusion matrix-----
[[35507  9593]
 [ 8736 36115]]
```

GaussianNB:

In statistics, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features (see Bayes classifier). They are among the simplest Bayesian network models, but coupled with kernel density estimation, they can achieve higher accuracy levels.

Naïve Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

In the statistics and computer science literature, naive Bayes models are known under a variety of names, including simple Bayes and independence Bayes. All these names reference the use of Bayes' theorem in the classifier's decision rule, but naïve Bayes is not (necessarily) a Bayesian method.

$$p(x = v \mid C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}}\, e^{-\frac{(v-\mu_k)^2}{2\sigma_k^2}}$$

In this project, the Scikit-learn module is used to build a model in python for classification of target as 0 and 1. The following are python and R code for GaussianNB Model.

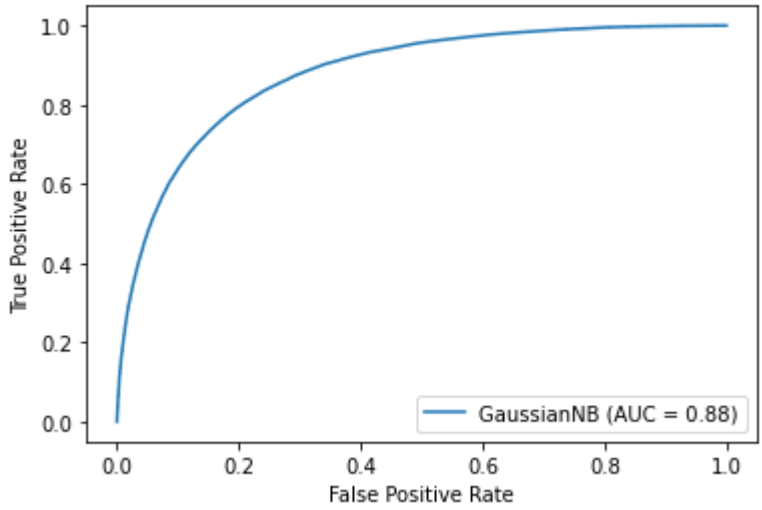
Python code:-

```
    print("Gaussian NB:- ")
    #GaussianNB
    gnb = GaussianNB()
    gnb.fit(Xtrain, ytrain)
    gnb_ypred = gnb.predict(Xtest)

    #print classification report
    print(classification_report(ytest, gnb_ypred)+"\n\n")
    #print accuracy score
    print("AUC Score:- %.3f \n\n" % roc_auc_score(ytest, gnb_ypred))

    #print confusion matrix
    cnf_mtx(ytest, gnb_ypred, "GaussionNB-ConfusionMatrix.png")
    #plot ROC curve
    plot_roc_curve(gnb, Xtest, ytest)
    #axis labels
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    #show plot
    plt.show()
```

R code:-

```r
print("GaussianNaiveBayes:-")
#GaussianNB
#setiing seed
set.seed(2)
#fit the model
fit.naivebayes = naiveBayes(target ~., data = training)

# print(fit.naivebayes)
predictions.naivebayes = predict(fit.naivebayes, validation)
#calculate confusion matrix
cm = ConfusionMatrix(predictions.naivebayes, validation$target)
naivebayes = list()
#calculate Scores
naivebayes[['accuracy']] = Accuracy(predictions.naivebayes, validation$target) * 100
#plot ROC curve
naivebayes[['auc']] = roc.curve(predictions.naivebayes, validation$target)$auc
naivebayes[['fnr']] = (cm['one','zero'] * 100)/(cm['one','zero']+cm['one', 'one'])
naivebayes[['ConfusionMatrix']] = cm
#print scores and confusion matrix
print(naivebayes)
```

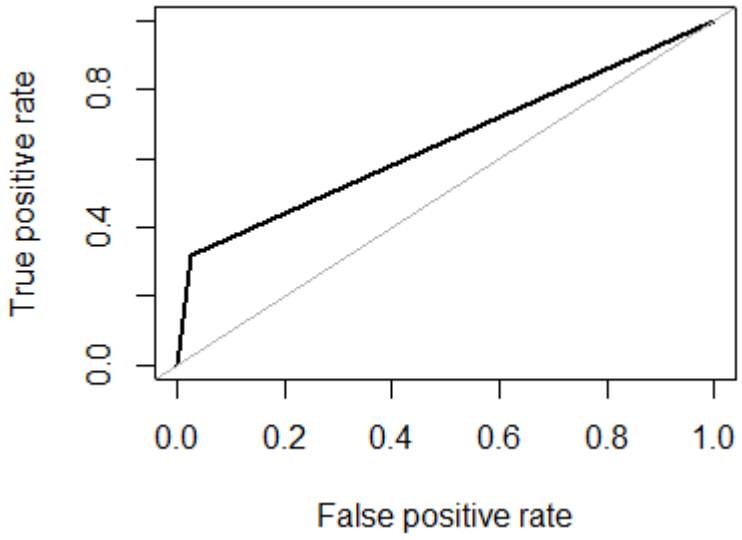
Python-ROC curve



Python-Classification Report:-

```
Gaussian NB:-
              precision    recall  f1-score   support

           0       0.81      0.78      0.79     45100
           1       0.79      0.81      0.80     44851

    accuracy                           0.80     89951
   macro avg       0.80      0.80      0.80     89951
weighted avg       0.80      0.80      0.80     89951


AUC Score:- 0.798


----confusion matrix-----
[[35204  9896]
 [ 8300 36551]]
```

R-ROC curve



R-Classification Report:-

```
$accuracy
[1] 92.16784

$auc
[1] 0.8245057

$fnr
[1] 63.47532

$confusionMatrix
        y_pred
y_true   zero   one
  zero 44248   727
  one   3189  1835
```

Conclusion:

Finally, the accuracy score of random forest and GaussianNB in python is almost equal and high and upto 80.9% and 80.0% . When coming to R the accuracy score GaussianNB is high and upto 92.16%. So GaussainNB is best to predict the target variable. Moreover F1-Score, Precision, Recall, FNR are best score when compare to other machine learning models.