Project Report

On

Santander Customer Transaction Prediction Dataset

Prepared By : Sankalp Srivastava

1. Background of the Problem Statement:

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?

2. Problem Statement:

In this challenge, we need to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

3. Data Set & Attributes:

We are provided with a train and a test dataset which are in CSV format.

The train.csv file has the dimension of 200000x202 i.e. 2 lac observation and 202 features/variables among which 201 are independent features or variable and one target variable or dependent variable.

The test.csv file has the dimension of 200000x201 i.e. 2 lac observation and 201 independent features or variables.

4. Exploratory Data Analysis (EDA):

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

Various processes involves in the EDA are as follows:

4.1 Examining the Dataset:

- Right after we load the data set to R/Python environment we would have a look at how the data looks i.e. what are the types of features? How many of them are categorical and how many are continuous or numerical?
- In our given dataset:
    - ➢ 200 independent variable are continuous or numeric type and labeled as var_0, var_1, var_2 ……var_199

- 1 variable which is ID_code is representing just the observation ID_code and it will contribute nothing in our result prediction so we will drop this variable from our dataset.
- One variable which is left is our target variable, which is of categorical type in which class ‘1’ represent that the customers will make a specific transaction in the future, irrespective of the amount of money transacted & class ‘0’ represents that customers will not make a specific transaction in the future.

- By figuring out this we can be sure that none of the features are of categorical type and we need to perform the various test only on the numerical variables which somewhat shorten the list of statistical test we would like to conduct on the dataset.
- One more thing that we can say by looking at the dataset is that we cannot really interpret the result of our model in terms of actual real world feature label which the variables labeled as var_0, var_1 etc. are representing.
- By conducting this investigation over the target class, we found out that target class ‘1’ is having approx 10% of the observations and target class ‘0’ is having rest of the 90% observations. So our data set is having **imbalanced target class**. We’ll keep this issue noted.

4.2 Data Cleaning:

Now as we have an overview that how our data looks like or what kind of features are there in our dataset, we need to move on the next most important step i.e. Data Cleaning.

Quality of our data is critical in getting to final analysis. Any data which tend to be incomplete, noisy and inconsistent can affect our result. Thus Data cleaning in data mining is the process of detecting and removing corrupt or inaccurate records from a record set, table or database.

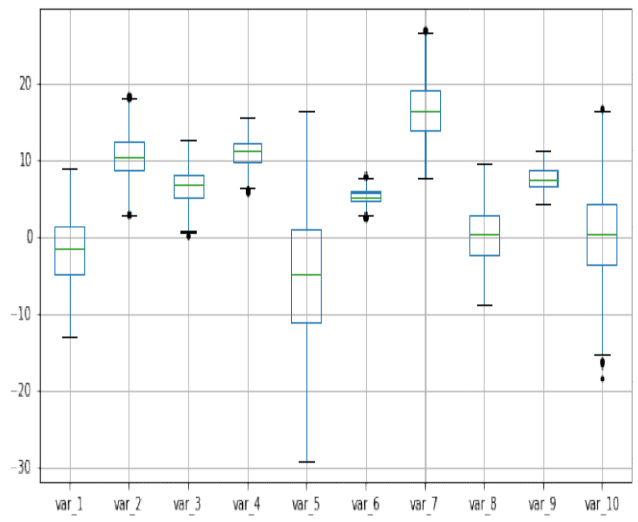
Cleaning of data involves the removal of noise from our dataset. In our dataset we would be conducting Missing Value Analysis, Outlier Analysis to remove noisy or incomplete records.

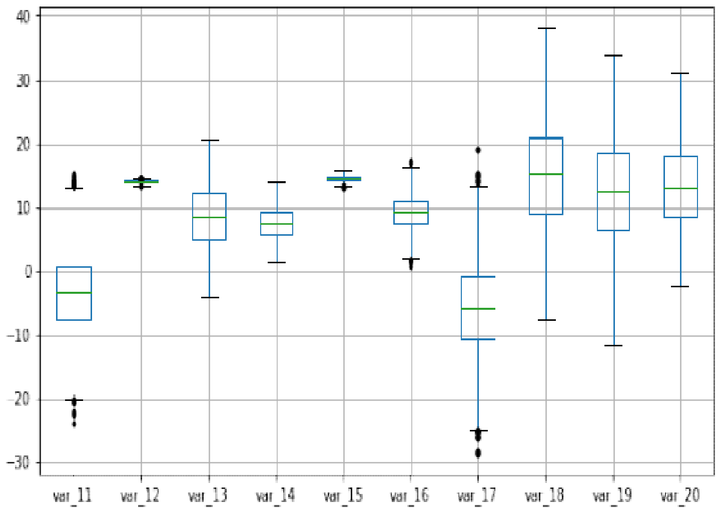
- **Missing Value Analysis:** There can be some missing values in our data set for the variables in an observation due to various reasons:
  i) Manual Data Entry Methods
  ii) Measurement Error
  iii) Optional question in Questionnaire
  iv) Equipment Error

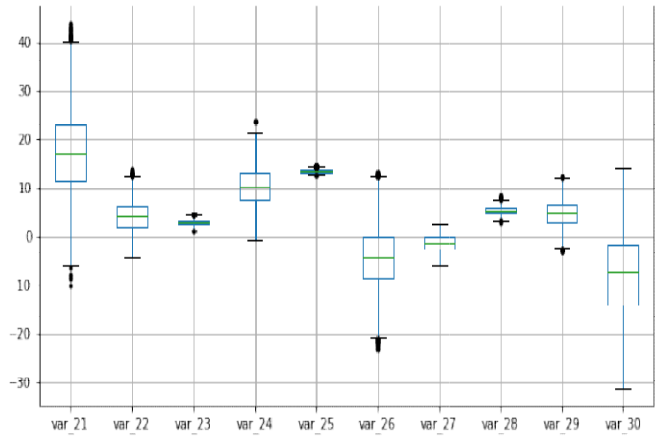
We can check for the missing values under all the variables, and if the percentage of missing value for a variable is under a threshold limit (generally 30%), we try to impute it using various statistical methods such as mean, median for numerical variable and mode for categorical variables. Another method which is very frequently used to impute the missing values is K-Nearest neighbor method.

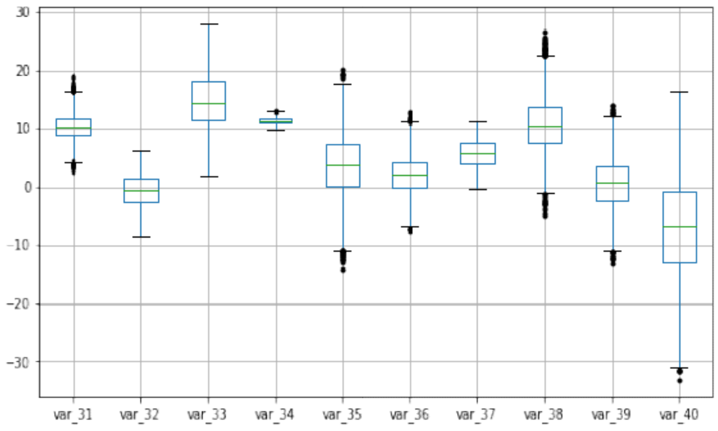
After checking for the null/missing values in our dataset we found out that there are no missing data in our given train and test datasets.

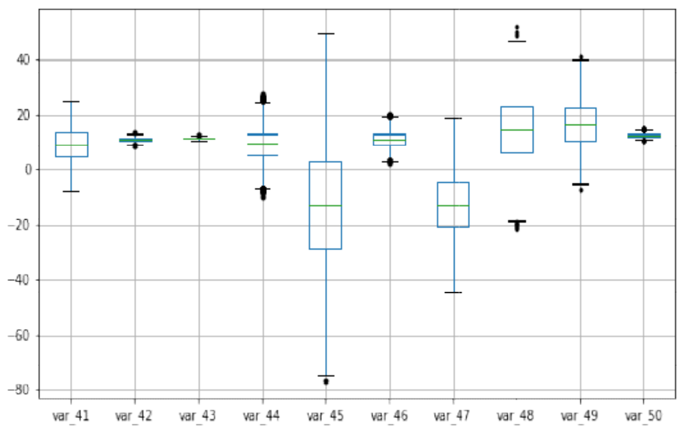
- **Outlier Analysis:** Outliers are that data points which are significantly different from rest of the data points under a variable**,** we can basically check the existence of such data points using **box plot** which has the representation of various statistical parameters i.e. min value,max value, median(50 Percentile) & Inter Quartile Range IQR (which is nothing but 75 Percentile(Q75) – 25 Percentile(Q25))

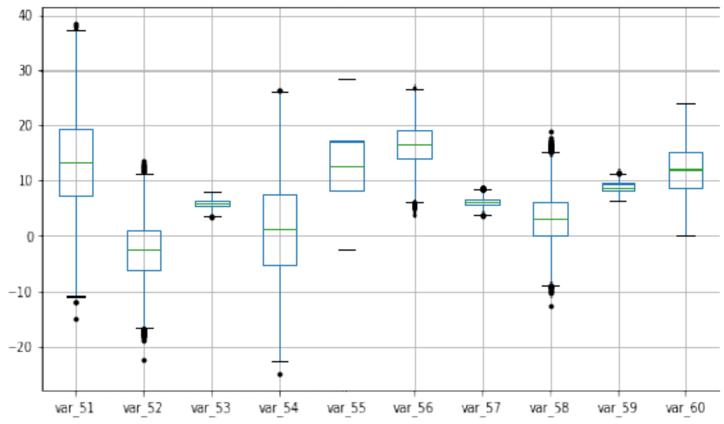












We can see from few of the box plots shown above that spread of data is not very significant and various data points outside the min-max range are not very significantly higher values. So in such case we omit the outlier removal process and will continue with the same dataset.

5. Feature Selection:

After missing value analysis and removal of Outliers from the dataset, our dataset is now free from noise and null values; we can now focus on the individual feature's characteristics and their relation with other variables.

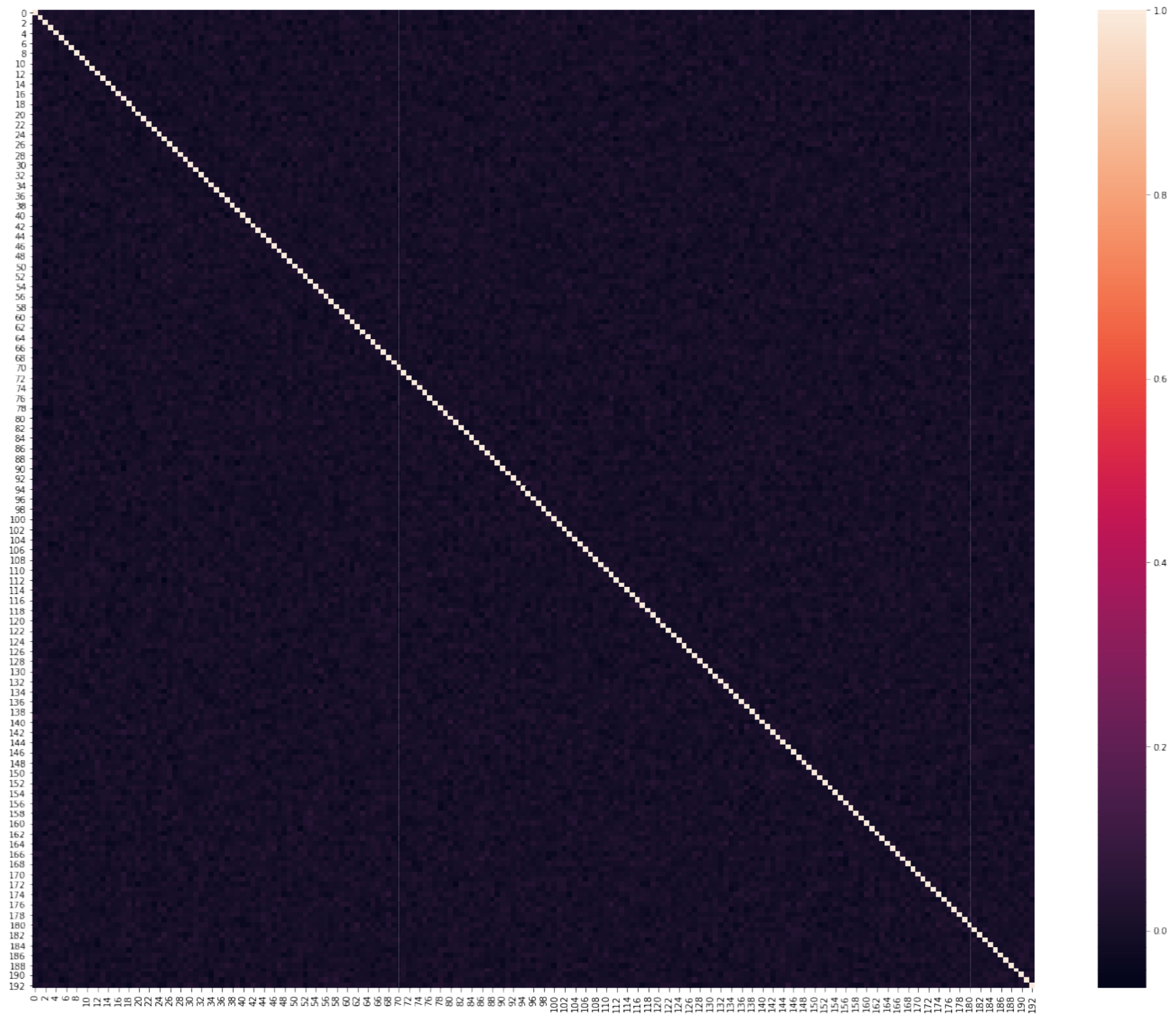
5.1 Constant feature removal:

The first step in this process is to filter out the features which are having Constant entries in their samples.

5.2 Correlation:

- As all the independent variables in our dataset are of numerical/continuous type, we would first check whether any variables are mutually correlated with each other, as this can lead to an effect known as **Multicollinearity**.

- To check the correlation between all the variables we can plot a heat map of the correlation matrix which can be created using various functions in R & Python.



<u>The above shown heat map shows that white blocks in the diagonals are having correlation as 1 which is normal as it is showing the correlation of any variable with itself. All the other areas are having black block which represents 0 or near to 0 correlations.</u>

<u>One thing to note here that in such scenario the algorithms which have the assumption of independent variables may perform well in this case like Logistic Regression, Naïve Bayes etc.</u>

6. Handling Imbalanced target Class:

- Now, the balancing of training data sets can be achieved either by increasing the frequency of minority class or decreasing the frequency of majority class in order to make both classes equal or close to each other.

- Few of the Re-sampling techniques are as follows:
  - **Random Under Sampling of Majority Class**
  - **Random Over Sampling of Minority Class**
  - **Cluster based Over Sampling**
  - **Synthetic Minority Over Sampling Technique (SMOTE) in Python**
  - **Modified Synthetic Minority Over Sampling Technique (MSMOTE) in Python**
- As our data set is quite large and it will be a cumbersome & time consuming process to run various algorithms on it on local machine, so we need to take a sample from the given dataset
- So under this scenario when we have to perform sampling anyway Random Under Sampling of Majority class seems a fair idea to me.
- So, we will be using the Random Under Sampling Technique to balance our imbalanced target class.
- Now as the number of observations in training set having target class as '1' is 16,078 and observations having class '0' is 1,43,922 so after under sampling the observation for target class '0' will reduce to 16,078 i.e. equal to the class'1' observations/samples.
- Thus now we have an overall 32,156 observations to work with for our training set.

7. MODELING:

7.1 Model Creation:-

Now as our dataset is well pre-processed & cleaned, we can proceed with the modeling part. First step is to select a model which will perform best on our dataset, and provide the best result/predictions as per the client's requirement.

We can judge our model by using various metrics like Accuracy Score, Precision, Recall, F-Score & ROC-AUC Score.

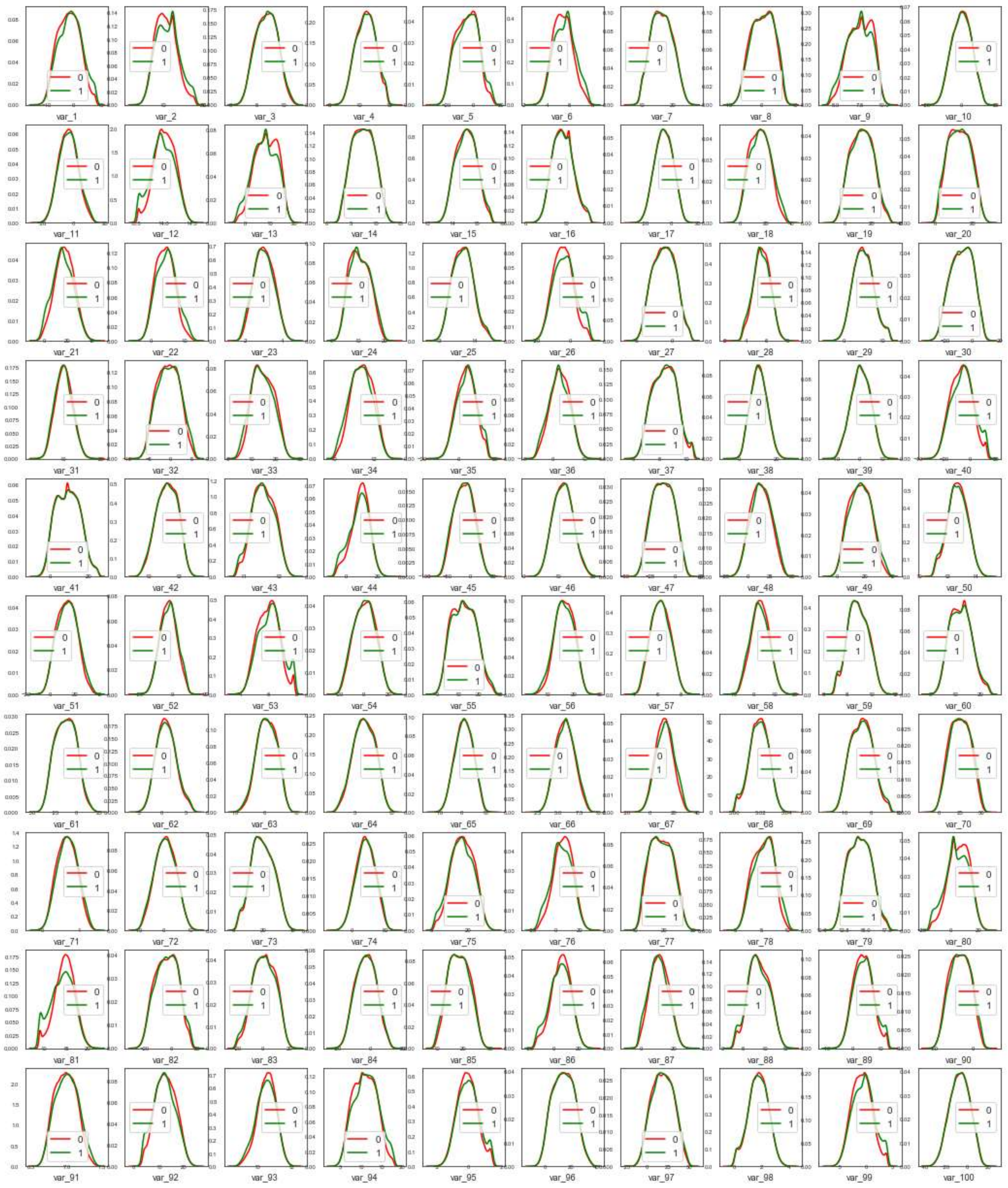
As per the problem statement we need to find a model which can predict the customers who are willing to do a specific transaction irrespective of the previous transaction, so we need to find the potential customers to whom the bank can target and approach them with the new scheme.

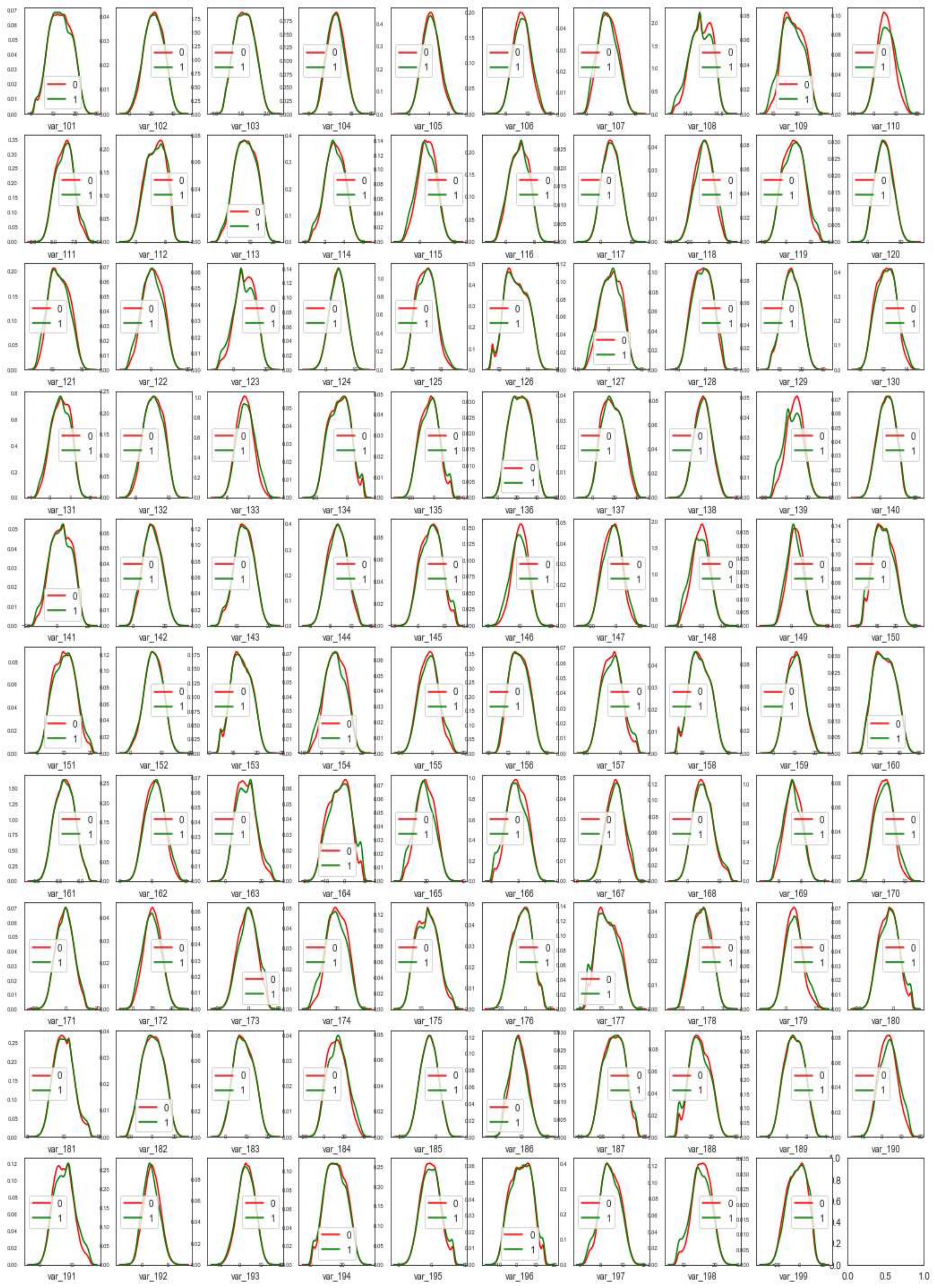
Here the target class '1' represents those potential customers; hence our model should perform well on following metrics:

- **Recall Score**: It represents the capability of predicting the target class '1' correctly. It is defined by the formula TP/(TP+FN)

- **Precision Score**: It represents the capability of predicting the negative class “0” correctly. It is defined by the formula TN/(TN+FP)
- **ROC-AUC Score**: It represents the area under the ROC Curve which is very important as it tells us how well our model is able to differentiate the two classes as it is the plot between the True Positive Rate (TPR) i.e. Sensitivity and False Positive Rate (FPR).
  Here TPR is the proportion of potential customers our model identified correctly & FPR is the proportion of customers who classified as potential customers but actually they are not.
- **False Negative Rate:** Another important metric is the False Negative Rate which represents proportion of customers who actually are potential customers but our model classified them as not potential.
  So we need to minimize it as it would make the bank to skip approaching these customers for doing the specific transaction.

In order to select the best method for the given problem statement, we first plot a distribution graph of various variable for both the target classes to have a look if there is any significant difference in their distribution which will help us to choose which method will work well on the particular dataset.


0
1
var_1
var_2
var_3
var_4
var_5
var_6
var_7
var_8
var_9
var_10
var_11
var_12
var_13
var_14
var_15
var_16
var_17
var_18
var_19
var_20
var_21
var_22
var_23
var_24
var_25
var_26
var_27
var_28
var_29
var_30
var_31
var_32
var_33
var_34
var_35
var_36
var_37
var_38
var_39
var_40
var_41
var_42
var_43
var_44
var_45
var_46
var_47
var_48
var_49
var_50
var_51
var_52
var_53
var_54
var_55
var_56
var_57
var_58
var_59
var_60
var_61
var_62
var_63
var_64
var_65
var_66
var_67
var_68
var_69
var_70
var_71
var_72
var_73
var_74
var_75
var_76
var_77
var_78
var_79
var_80
var_81
var_82
var_83
var_84
var_85
var_86
var_87
var_88
var_89
var_90
var_91
var_92
var_93
var_94
var_95
var_96
var_97
var_98
var_99
var_100


var_101
var_102
var_103
var_104
var_105
var_106
var_107
var_108
var_109
var_110
var_111
var_112
var_113
var_114
var_115
var_116
var_117
var_118
var_119
var_120
var_121
var_122
var_123
var_124
var_125
var_126
var_127
var_128
var_129
var_130
var_131
var_132
var_133
var_134
var_135
var_136
var_137
var_138
var_139
var_140
var_141
var_142
var_143
var_144
var_145
var_146
var_147
var_148
var_149
var_150
var_151
var_152
var_153
var_154
var_155
var_156
var_157
var_158
var_159
var_160
var_161
var_162
var_163
var_164
var_165
var_166
var_167
var_168
var_169
var_170
var_171
var_172
var_173
var_174
var_175
var_176
var_177
var_178
var_179
var_180
var_181
var_182
var_183
var_184
var_185
var_186
var_187
var_188
var_189
var_190
var_191
var_192
var_193
var_194
var_195
var_196
var_197
var_198
var_199
0
1

As it can be clearly seen from the distribution that for both the target classes the distribution of variable parameters are somewhat similar, in such scenario tree based algorithms can perform well.

However, we'll start with the most basic model used for classification i.e. Logistic Regression.

a) LOGISTIC REGRESSION:

It is a linear model which uses the maximum likelihood function to generate the probability of our output variable i.e. 'Target' class.

Logistic Regression which is used to generate the probability of the target class is

$$p(X) = e^{\beta0+\beta1X}/1 + e^{\beta0+\beta1X}.$$

Now if we re-arrange the above equation:

$$p(X)(1+ e^{\beta0+\beta1X}) = e^{\beta0+\beta1X}$$

$$p(X)/(1-p(X)) = e^{\beta0+\beta1X}$$

The fraction in left side of the equation is called as the odds and it can take any value between 0 to infinity. 0 indicate very low probability whereas infinity indicates very high probability.

Now if we take log of both sides of the equation:

$$\log(p(X)/(1-p(X))) = \beta0+\beta1X$$

Thus log of the odds is a linear function of the weighted avg of variables in our dataset. The log of the odds is called as **logit or log-odds.**

In a logistic regression model, increasing in *X* by one unit changes the log odds by β_1 (4.4), or equivalently it multiplies the odds by e^{β_1}

Now we will apply our logistic regression model to our balanced dataset and then check its performance based on the above mentioned classification metrics.

The Performance of the logistic Regression Model on our balanced data set can be tabled as below:

Model	**Accuracy**	**Precision**	**Recall**	**ROC-AUC Score**	**FPR**	**FNR**	**Python/ R**
Logistic Regression	77.79%	77.83%	77.51%	77.67%	22.17%	22.48%	R
Logistic Regression	77.76%	28.17%	77.73%	77.75%	22.23%	22.26%	Python

b) Decision Tree Algorithm:

This Algorithm involves *stratifying* or *segmenting* the predictor space in to a number of simple regions. In order to make a prediction for a given observation, we typically use the mean or the mode of the training observations in the region to which it belongs. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as *decision tree* methods.

In classification tree, we predict that each observation belongs to the *most commonly occurring class* of training observations in the region to which it belongs. In interpreting the results of a classification tree, we are often interested not only in the class prediction corresponding to a particular terminal node region, but also in the *class proportions* among the training observations that fall into that region.

The classification in the decision tree algorithm is based on the node purity measures, which can be computed by mainly two methods:

1. ***Gini index:*** It is basically a measure of total variance across kth class & is defined as: $G = \sum p_{mk}(1-p_{mk})$, where p_{mk} represents the proportion of the mth region that belongs to the kth class.
   It can be seen that all if proportion is very high (i.e.1) or very low (i.e. 0) G will be very small. Thus the small value of G indicates the node purity & we can be more certain that the particular observation belongs to any particular class.

2. ***Cross-Entropy***: It is an alternative to the Gini Index, and is quite similar to it.
   It can be defined as $D = - \sum p_{mk} \log p_{mk}$, it also provides low values if proportion of any particular class is very high or very low.

Now let's see how well Decision tree perform on our given data set:

Model	Accuracy	Precision	Recall	ROC-AUC Score	FPR	FNR	Python/ R
Decision Tree	62.37%	62.31%	62.93%	62.63%	37.68%	37.06%	R
Decision Tree	58.72%	13.54%	57.46%	58.16%	41.14%	42.54%	Python

c) Random Forest Classifier:

As we have seen that a single tree is not providing us the desired result, now let us create a huge number of such trees and then predict our final result based on the majority vote method as while building a random forest, at each split in the tree, the algorithm is *not even allowed to consider* a majority of the available predictors. This may sound crazy, but it has a clever rationale. Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors. Then in the collection of bagged trees, most or all of the trees will use this strong predictor in the top split. Consequently, all of the bagged trees will look quite similar to each other. Hence the predictions from the bagged trees will be highly correlated. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities. In particular, this means that bagging will not lead to a substantial reduction in variance over a single tree in this setting.

Random forests overcome this problem by forcing each split to consider only a subset 'm' of the total predictors 'p'. Therefore, on average $(p-m)/p$ of the splits will not even consider the strong predictor, and so other predictors will have more of a chance. We can think of this process as *de-correlating* the trees, thereby making the average of the resulting trees less variable and hence more reliable.

Now let's try to fit the Random Forest Algorithms on our training set and see the Result:

Model	Accuracy	Precision	Recall	ROC-AUC Score	FPR	FNR	Python/R
Random Forest	74.00%	73.31%	80.09%	76.70%	26.68%	19.90%	R (with 500 Trees)
Random Forest	74.00%	25.02%	79.03%	76.23%	26.56%	20.97%	Python(with 2000 trees

d) Naïve Bayes Algorithm:

Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified, is independent of each other.

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$$P(A|B) = P(B|A) \times P(A) / P(B)$$

where A and B are events.

- Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as **evidence**.
- P(A) is the **priori** of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance(here, it is event B).
- P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

Now, with regards to our dataset, we can apply Bayes' theorem in following way:

$$P(y|X) = P(X|y) \times P(y) / P(X)$$

where, y is class variable and X is a dependent feature vector (of size *n*) where:

$$X = (x_1,x_2,x_3\ldots\ldots\ldots x_n)$$

We can use the Naïve bayes algorithm for our data set as the assumption of independence of features in naïve bayes is almost true in the given dataset as the correlation between the variables is extremely low.

Now let us fit the Naïve Bayes Classifier Algorithm to our training set and see the performance metrics:

Model	**Accuracy**	**Precision**	**Recall**	**ROC-AUC Score %**	**FPR**	**FNR**	**Python/ R**
Naïve Bayes	80.86%	80.87%	80.79%	80.84%	19.12%	19.20%	R
Naïve Bayes	80.81% We	31.98%	79.02%	80.49%	19.11%	19.91%	Python

can see here that Naïve Bayes algorithm has performed better than the other models used so far for the given dataset.

It provides the least False Negative Rate & Highest Recall & ROC-AUC Score when tested against our validation set.

7.2 Model Selection :

As our main aim for this problem statement was to identify the potential customer who will perform a specific transaction irrespective of the previous transaction, so we have performed various algorithms in their basic mode i.e. without hyper parameter tuning and select our final model based on some important metrics like Recall, ROC-AUC Score (Area Under the ROC Curve), Precision, False Positive & False Negative Rate.

From all the four basic models that we have fitted on our training set we can conclude that the Naïve Bayes Algorithm performed better than all the other model and takes very less time compared to the other model like Random Forest, Logistic Regression etc.

The better performance of Naïve bayes might be due to the reason that this algorithm is basically based on the assumptions that all the predictor variables are independent of each other, and in this given dataset all the features are having very low correlation among themselves.

As the model is providing higher ROC-AUC Score which is basically a indicator that how better the particular classifier classifies the target classes & Recall indicates that how well the model predicted target class '1' which was our main focus.

So by correctly classifying the potential customer i.e. target class '1' bank can approach them with various future prospects and schemes and the identified customers will be more likely to make that specific transaction based on their feature which was analyzed by our model.

<u>Instruction to Run the Codes</u>

Code written in R:

```
rm(list = ls())

#Setting the working Directory first
setwd("C:/Users/acer/Desktop/edWisor Project1")

#re-check the working directory
getwd()

#Loading important Libraries
library(caret) # For various perfomance parameters
library(caTools) # FOr splitting the datasett
library(dplyr) # For balancing the target class of dataset
library(corrplot) #For plotting the correlation heatmap
library(ggplot2) #For visualization
library(Matrix)
library(pROC) #For calculating the ROC-AUC Score
library(C50) #For using Decision Tree Algorithm
library(randomForest) # For using Random Forest Algorithm
library(e1071) # for using Naive Bayes Algorithm

#Load the train & test dataset into the R environment.
df_train = read.csv("train.csv", header = TRUE)
df_test = read.csv("test.csv",header = TRUE)


###################### Exploratory Data Analysis (EDA) ##########################

# 1. EXAMINING THE DATASET
```

```
View(df_train)
View(df_test)

# After viewing both the train and test data:

# 1. Both the dataset are having 200000 observations. The train set has 202 whereas test set has
201 features/variables/predictors

# 2. Note that the feature "ID_code" in both the dataset is just representing index of
observations and it will not contribute
#    anything in our model so we'll just drop this variable

df_train = df_train[,-1]
df_test = df_test[,-1]

# See the statistical parameters of the rest of the predictors/variables

summary(df_train)
summary(df_test)

str(df_train)
str(df_test)
#After viewing the dataset summary we can conclude all the indipendent variables are numeric
in nature and are having numerical values in a narrow range.



#Let's check the distribution of target class in the train dataset
table(df_train$target)
#Visualization
hist(df_train$target,col = "green", main = paste("Imbalanced Target Class Distribution"))


#   We can clearly see from the plot that the target class is imbalanced and majority and minority
classes have 1,79,902 & 20,098
#   observations i.e. having a ratio of approx 9:1
#  This imbalance in target class will create a bias towards the Majority class in the prediction so
we need
#  to keep this in mind before training our model.
```

2.DATA CLEANING

```
##  Missing Value Analysis
#Check for the missing value
sum(is.na(df_train))
sum(is.na(df_test))

#result shows that no missing values are there in both train and test dataset


### creating dependent and independent arrays
X = df_train[,-1]
y = df_train[,1]


####### 3. FEATURE SELECTION

### Removal of constant features

names(df_train[,sapply(df_train, function(v) var(v, na.rm = TRUE)== 0)])

df_train[,sapply(df_train, function(v) var(v, na.rm = TRUE)== 0)]



### Checking for the Correlation between all the variable as all are of numeric type with the help
of a heatmap

corrplot(cor(X))
corrplot(cor(df_test))
# we can see from the Correlation plot (heatmap) that there are no correlated features in the
dataset
```

```
#########   Spliting the dataset in train and validation set #############


set.seed(121)

split = sample.split(df_train$target,SplitRatio = 0.8)

training_Set = subset(df_train, split==TRUE)
Validation_set = subset(df_train,split==FALSE)

print(dim(training_Set))
print(dim(Validation_set))

X_train = training_Set[-1]
y_train = training_Set[1]

X_val = Validation_set[-1]
y_val = Validation_set[1]


######## Handling the Imbalanced target class #############


#    Now as our dataset is huge i.e. it has 2lacs observation we can perform undersampling
#    method to balance as well as sample our data set as it will take huge amount
#    of time and computaion power to perform operations on 2Lacs observations, therefore we
will use the Random Under
#    Sampling technique to sample as well as balance our dataset

#Sampling the training dataset

Majority_class = subset(training_Set,training_Set$target==0)

Minority_class = subset(training_Set,training_Set$target==1)
```

```
dim(Majority_class)[1]

#Undersampling the Majority Class
set.seed(123)
training_Set_Und_samp<- training_Set %>%
  group_by(target) %>%
  sample_n(dim(Minority_class)[1])


dim(training_Set_Und_samp)

table(training_Set_Und_samp$target)


X_train_res = training_Set_Und_samp[-1]
y_train_res = training_Set_Und_samp[1]





##############################      Model Creation   ################

#Let's first define a function that returns the variaous important performance metrics


Performance_metrics <- function(y_true,y_pred) {
  cm = as.matrix(table(y_true,y_pred))
  Accuracy = sum(diag(cm))/sum(cm)
  Precision_m = cm[1,1]/(cm[1,1]+cm[1,2])
  Recall_m = cm[2,2]/(cm[2,2]+cm[2,1])
  ROC_AUC_Score = roc(y_actual,as.numeric(y_pred))
  auc_score = auc(ROC_AUC_Score)
  FNR = cm[2,1]/(cm[2,1]+cm[2,2])
  FPR = cm[1,2]/(cm[1,2]+cm[1,1])

  paste0("Accuracy: ",Accuracy, ",  Precision: ",Precision_m,",  Recall : ",Recall_m,",  ROC-AUC
Score: ", auc_score,  ",  FPR :", FPR, ",  FNR: ", FNR)
}

#################### 1. LOGISTIC REGRESSION  #################
# We will start the creation of model with our basic linear model Logistic Regression


clf_LR = glm(target~.,data=training_Set_Und_samp, family = "binomial")

summary(clf_LR)
prob_pred = predict(clf_LR, type = "response",newdata = Validation_set[-1])

y_pred_LR = ifelse(prob_pred > 0.5,"1","0")

y_actual = Matrix::t(y_val)

#Confusion Matrix
cm_LR = as.matrix(table(y_actual,y_pred_LR))
cm_LR

#Evaluating Performance Metrics
Performance_metrics(y_actual,y_pred_LR)

# We get an accuracy of 77.79 % & a recall score of 77.51% with the Logistic regression model
# Let's evaluate some more important meterics for our result




################## 2.Decision Tree Algorithms ################

# As our Logistic regression model doesn't provide that great result let's try decision tree method
now


training_Set_Und_samp$target = as.factor(training_Set_Und_samp$target)
```

```
clf_DT = C5.0(x = training_Set_Und_samp[-1],y = training_Set_Und_samp$target)

y_pred_DT = predict(clf_DT, type = "class", newdata = X_val)

cm_DT = as.matrix(table(y_actual,y_pred_DT))
cm_DT

Performance_metrics(y_actual,y_pred_DT)


#Showing the tree classification
plot(clf_DT)
text(clf_DT)




############### 3. Random Forest Algorithm #######################


clf_RF = randomForest(target ~.,data = training_Set_Und_samp, importance = TRUE,ntree = 500)

y_pred_RF = predict(clf_RF, X_val, type = "class")

cm_RF = as.matrix(table(y_actual,y_pred_RF))
cm_RF

Performance_metrics(y_actual,y_pred_RF)



#Showing the classification
plot(clf_RF)
text(clf_RF)
```

```
################## 4. Naive Bayes Algorithm ################


clf_NB = naiveBayes(target~.,data = training_Set_Und_samp)

y_pred_NB = predict(clf_NB, type = "class", newdata = X_val)

cm_NB = as.matrix(table(y_actual,y_pred_NB))
cm_NB

Performance_metrics(y_actual,y_pred_NB)







################################# Model Selection ###################################



#Based on the performance metrics of various models we will select the Naive Bayes as along
with providing
# great accuracy it also has the higher ROC-AUC Score and high Recall value.

# Now let's predict the test dataset results and store the file


y_pred_test = predict(clf_NB,newdata = df_test,type = "class")


# Adding this predicted vector of target class into the df_test dataframe and store it in the
working directory
```

```
# location by the name of Submission_R.csv

df_test$predicted_target_Class = y_pred_test

write.csv(df_test,"C:/Users/acer/Desktop/edWisor
Project1/Submission_R.csv",row.names=FALSE)
```

Code Written in Python:

#**importing basic libraries**

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import os
from sklearn.feature_selection import VarianceThreshold
```

#importing various algorithms

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
```

#importing libraries for Performance Metrics

```
from sklearn.metrics import
accuracy_score,confusion_matrix,roc_auc_score,roc_curve,precision_score,recall_score,f1_scor
e,classification_report
from sklearn.model_selection import cross_val_score
```

#for handling Imbalance Dataset

```
from imblearn.under_sampling import RandomUnderSampler
```

#**Setting the working directory**

```
os.chdir("C://Users/acer/Desktop/edWisor Project1")
```

#**Cross-Checking the set directory**

```
os.getcwd()
```

#Load the train and test Dataset and look at the top five observations of each

```
    df_train = pd.read_csv("train.csv")
    df_train.head()
```

```
  df_test = pd.read_csv("test.csv")
  df_test.head()

##### Doing Exploratory Data Analysis
## First Examining the data:
df_train.info()
df_test.info()
df_train.dtypes.value_counts()
df_test.dtypes.value_counts()

#drop the ID_code variable from train and test dataset
df_train.drop(labels='ID_code',axis=1, inplace =True)
df_train.head()
df_test.drop(labels='ID_code',axis =1, inplace=True)
df_test.shape

# Checking for the target class distribution of train.csv dataset
df_train['target'].value_counts()

#plot a histogram to see the distribution of target class visualy
plt.hist(df_train['target'])

### Data Cleaning
## Missing Value Analysis
df_train[df_train.isnull().any(axis=1)]
df_test[df_train.isnull().any(axis=1)]

### See the spread of data in train and test dataset with the help of a box plot

#break the number of feature into chunks of 10 features:
# putting all the df colname in a list
dfcols_train = list(df_train.columns)
dfcols_test = list(df_test.columns)
# exculdig target and index columns
variables_train = dfcols_train[1:]
variables_test = dfcols_test[0:]
# splitting the list every n elements:
n = 10
```

```
chunks_train = [variables_train[x:x + n] for x in range(0, len(variables_train), n)]
chunks_test =  [variables_test[x:x + n] for x in range(0, len(variables_test), n)]

# displaying a boxplot of train dataset for every n columns:
for i in chunks_train:
   plt.show(df_train.boxplot(column = i, sym='k.', figsize=(10,5)))

for i in chunks_test:
   plt.show(df_test.boxplot(column = i, sym='k.', figsize=(10,5)))

##Feature Selection
# First we create independent & dependent variable dataframes and check their shapes
X = df_train.drop(labels='target',axis =1)
y = df_train['target']
X.shape, y.shape

#Apply filter to remove constant features from train and test dataset
const_filter_train = VarianceThreshold(threshold=0)
const_filter_train.fit(X)
const_filter_train.get_support().sum()

const_filter_test = VarianceThreshold(threshold = 0)
const_filter_test.fit(df_test)
const_filter_test.get_support().sum()

#Check for the correlation among predictors with the help of a heat map

plt.figure(figsize=(30,20))
sns.heatmap(X.corr())

sns.heatmap(df_test.corr())

## Now Let's break our model into train and validation set and check their shape
X_train,X_val,y_train,y_val = train_test_split(X,y, test_size = 0.2, random_state = 29)
X_train.shape, X_val.shape, y_train.shape, y_val.shape

## Handling Imbalanced Target Class
```

```
#check the original class distribution
y.value_counts()

#check the distribution of target class in training set
y_train.value_counts()

#Perform undersampling operation of majority target class to balance the target class
distribution
#creating the object of undersampling
UndSamp = RandomUnderSampler(sampling_strategy= 'auto',random_state=35)

#fit that to our training set & Storing the resampled observations in X_train_res & y_train_res
X_train_res,y_train_res = UndSamp.fit_sample(X_train,y_train)

#recheck the distribution of y_train after undersampling
y_train_res.value_counts()

##MODEL CREATION

#first create a function to generate the plot that show the distribution of various features
w.r.t. the target classes

def plot_feature_distribution(target_class_1, target_class_2, label1, label2, features):
   i = 0
   sns.set_style('white')
   plt.figure()
   fig, ax = plt.subplots(10,10,figsize=(18,22))

   for feature in features:
      i += 1
      plt.subplot(10,10,i)
      sns.distplot(target_class_1[feature], hist=False,label=label1, color='red')
      sns.distplot(target_class_2[feature], hist=False,label=label2, color='green')
      plt.xlabel(feature, fontsize=9)
      locs, labels = plt.xticks()
      plt.tick_params(axis='x', which='major', labelsize=6, pad=-6)
      plt.tick_params(axis='y', which='major', labelsize=6)
   plt.show()
```

#Now create a dataframe having resampled observations

```
df_training_res = pd.concat(objs=(X_train_res,y_train_res),axis=1)
df_training_res.head()
```

#Specify the parameters for generating the plot

```
tar_c1 = df_training_res.loc[df_training_res['target']== 0]
tar_c2 = df_training_res.loc[df_training_res['target']== 1]
features_set1 = df_training_res.columns.values[0:100]
```

#then plot the curve using the function created above

```
plot_feature_distribution(tar_c1,tar_c2,'0','1',features_set1)
```

#Again specifying rest of the features

```
 features_set2 = df_training_res.columns.values[100:200]
```

#again plot the distribution curve

```
  plot_feature_distribution(tar_c1,tar_c2,'0','1',features_set2)
```

#Now as we proceed to create our model let's create a function which provides us the various metrics to check the performance of our model on the validation set

```
def performance_metrics(y_test,y_pred,Classifier_Name):
    cm = confusion_matrix(y_test,y_pred)
    accuracy = accuracy_score(y_test,y_pred)
    TPR = round(cm[1,1]/(cm[1,1]+cm[1,0])*100,2)
    TNR = round(cm[0,0]/(cm[0,0]+cm[0,1])*100,2)
    FPR = round(cm[0,1]/(cm[0,0]+cm[0,1])*100,2)
    FNR = round(cm[1,0]/(cm[1,0]+cm[1,1])*100,2)
    roc_auc = roc_auc_score(y_test,y_pred)
    fpr,tpr,threshold = roc_curve(y_test,y_pred)
    print( "Accuracy : ", accuracy)
    print("Confusion Matrix: ", cm)
    print("Precision: ",precision_score(y_test,y_pred))
    print("Recall: ", recall_score(y_test,y_pred))
    print("F1-Score: ", f1_score(y_test,y_pred))
    print("TPR or Sensitivity : ", TPR, "TNR or Specificity: ", TNR,"FPR: ",FPR, "FNR: ",FNR)
    print("ROC-AUC Score: ",roc_auc)
    plt.plot(fpr,tpr)
    plt.plot(tpr,tpr)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.title(label="ROC_AUC Curve for " +str(Classifier_Name))
    plt.show()
```

#Create a classifier using Logistic Regression

```
clf_LR = LogisticRegression(C=0.01,random_state = 35,solver= 'lbfgs',max_iter=5000)
clf_LR.fit(X_train_res,y_train_res)
y_pred_LR = clf_LR.predict(X_val)
```

#run the function of performance metrics and see the performance of model

```
performance_metrics(y_val,y_pred_LR,"Logistic Regression")
```

#Decision Tree Classifier

```
clf_DT = DecisionTreeClassifier()
clf_DT.fit(X_train_res,y_train_res)
y_pred_DT = clf_DT.predict(X_val)
```

#run the function of performance metrics and see the performance of model

```
Performance_metrics(y_val,y_pred_DT,"Decision Tree")
```

#Random Forest Classifier

```
clf_RF = RandomForestClassifier(n_estimators=2000)
clf_RF.fit(X_train_res,y_train_res)
y_pred_RF = clf_RF.predict(X_val)
```

#run the function of performance metrics and see the performance of model

```
Performance_metrics(y_val,y_pred_DT,"Random Forest")
```

#Naive Bayes Classifier

```
clf_NB = GaussianNB()
clf_NB.fit(X_train_res,y_train_res)
y_pred_NB = clf_NB.predict(X_val)
```

#run the function of performance metrics and see the performance of model

```
performance_metrics(y_val,y_pred_NB,"Naive Bayes Classifier")
```

#Apply k-fold cross validation on Naïve Bayes classifier

```
acc_10_cv_NB = cross_val_score(clf_NB,X_train_res,y_train_res,cv=10)
acc_10_cv_NB
```

#taking mean of the 10 accuracies

```
avg_acuracy_NB = np.mean(acc_10_cv_NB)*100
avg_acuracy_NB
```

# Predicting the classes for the test dataset given using Naive bayes classifier

```
 y_pred = clf_NB.predict(df_test)
```

#Creating a column in the df_test dataset by the name "predicted target class"

```
df_test['predicted target class'] = y_pred
df_test.head()
```

#Storing the test dataset df_test into the working directory location by the file name "Submission.csv"

```
  df_test.to_csv(path_or_buf= "C://Users/acer/Desktop/edWisor Project1/Submission.csv")
```