**Santander Customer Transaction Prediction**

*Sahayaraj Daniel*

July 26, 2019

**Contents**

[**Chapter 1** 4](#_Toc15058921)

[**Introduction** 4](#_Toc15058922)

[**1.1** **Problem Statement** 4](#_Toc15058923)

[**1.2** **Data** 4](#_Toc15058924)

[**1.2.1 Exploratory Data Analysis:** 5](#_Toc15058925)

[**Chapter 2** 9](#_Toc15058926)

[**Methodology** 9](#_Toc15058927)

[**2.1 Pre-Processing** 9](#_Toc15058928)

[**2.1.1 Outlier Analysis** 11](#_Toc15058929)

[**2.1.2 Missing Value Analysis** 13](#_Toc15058930)

[**2.1.3 Feature Selection** 14](#_Toc15058931)

[**2.1.4 Feature Scaling** 17](#_Toc15058932)

[**2.1.5 Sampling Train Data** 17](#_Toc15058933)

[**2.2 Modelling** 18](#_Toc15058934)

[**2.2.1 Model Selection** 18](#_Toc15058935)

[**2.2.2 Logistics Regression:** 19](#_Toc15058936)

[**2.2.3 Decision Tree** 20](#_Toc15058937)

[**2.2.4 Random Forest** 21](#_Toc15058938)

[**Chapter 3** 22](#_Toc15058939)

[**Conclusion** 22](#_Toc15058940)

[**3.1 Model Evaluation** 22](#_Toc15058941)

[**3.1.1 Confusion Matrix** 22](#_Toc15058942)

[**3.1.2 ROC\_AUC Curve** 23](#_Toc15058943)

[**3.2** **Model Selection** 24](#_Toc15058944)

[**3.2.1 Logistics Regression Model** 24](#_Toc15058945)

[**3.2.2 Decision Tree Method** 25](#_Toc15058946)

[**3.2.3 Random Forest Method** 26](#_Toc15058947)

[**Appendix A – R Code** 28](#_Toc15058948)

[**Appendix B – Python Code** 34](#_Toc15058949)

# **Chapter 1**

# **Introduction**

## **Problem Statement**

Santander Bank’s data science team wants to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted. The bank is continually challenging its machine learning algorithms to make sure they can more accurately identify new ways to solve its most common challenges such as: Will a customer buy this product? Can a customer pay this loan?

The purpose of this project is to construct a prediction model using various machine learning algorithms and to document the end-to-end steps using a template. The Santander Bank Customer Transaction Prediction competition is a binary classification situation where we are trying to predict one of the two possible outcomes.

Here are

## **Data**

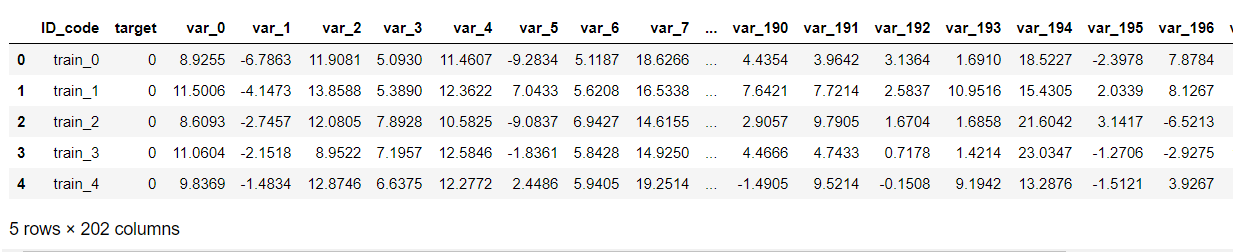
We have been provided with an anonymized dataset of train and test, containing 200 numeric feature variables, the binary target column, and a string ID\_code column being only in train data set. There are 200000 observations in each data set.

Train data set: (200000, 202)

Test data ser: (200000, 201)

**Some sample data:**

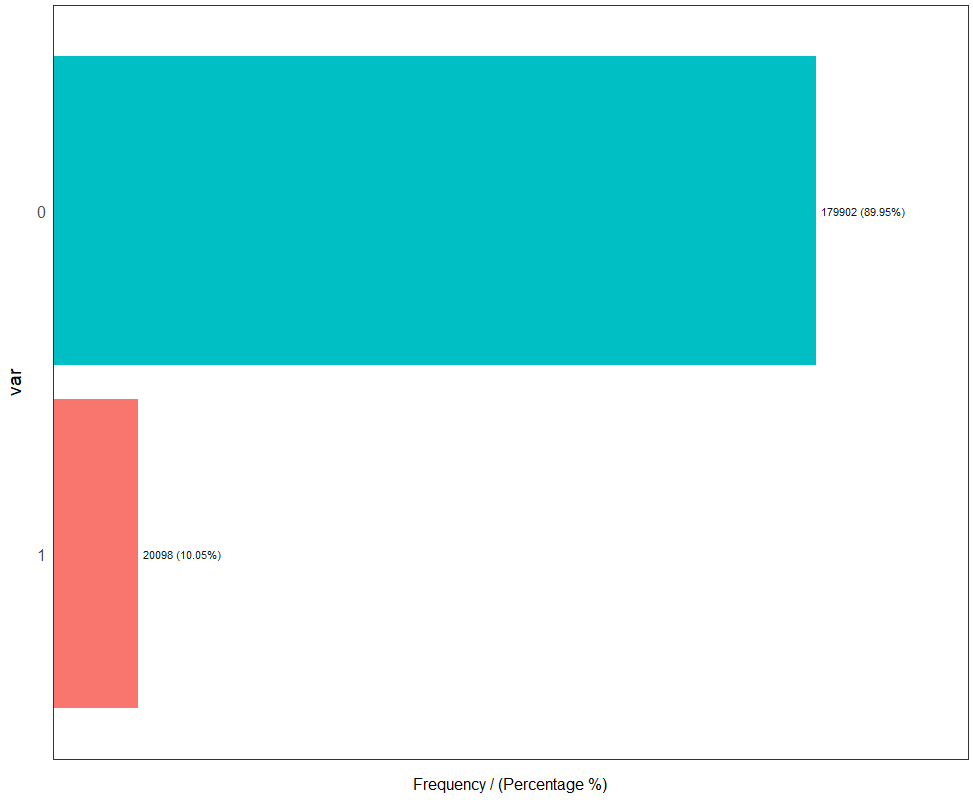
***Table1.1 Sample data (Raw Data)***



The original dataset contains an ID\_Code field, which is a primary key field of the table. There are 200 numerical fields from Var\_0, Var\_1,.. upto Var\_199. All these data are anonymized that there is no possibility to understand the business importance of the variables. All numerical variables are treated as the same variables and our aim is to identify which of these variables influences our target variable, which is a binary factor variable, with 1 being customer making a transaction in future and 0 being not.

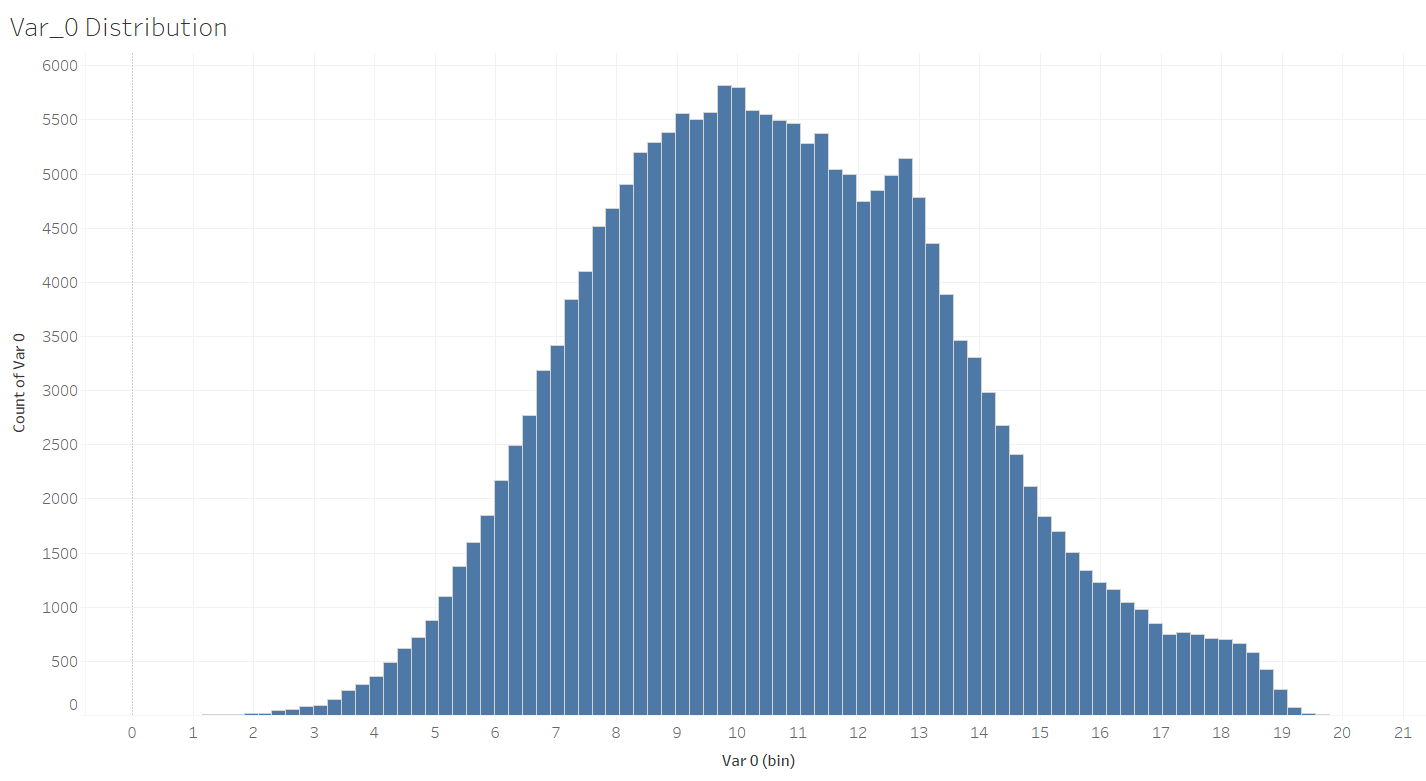
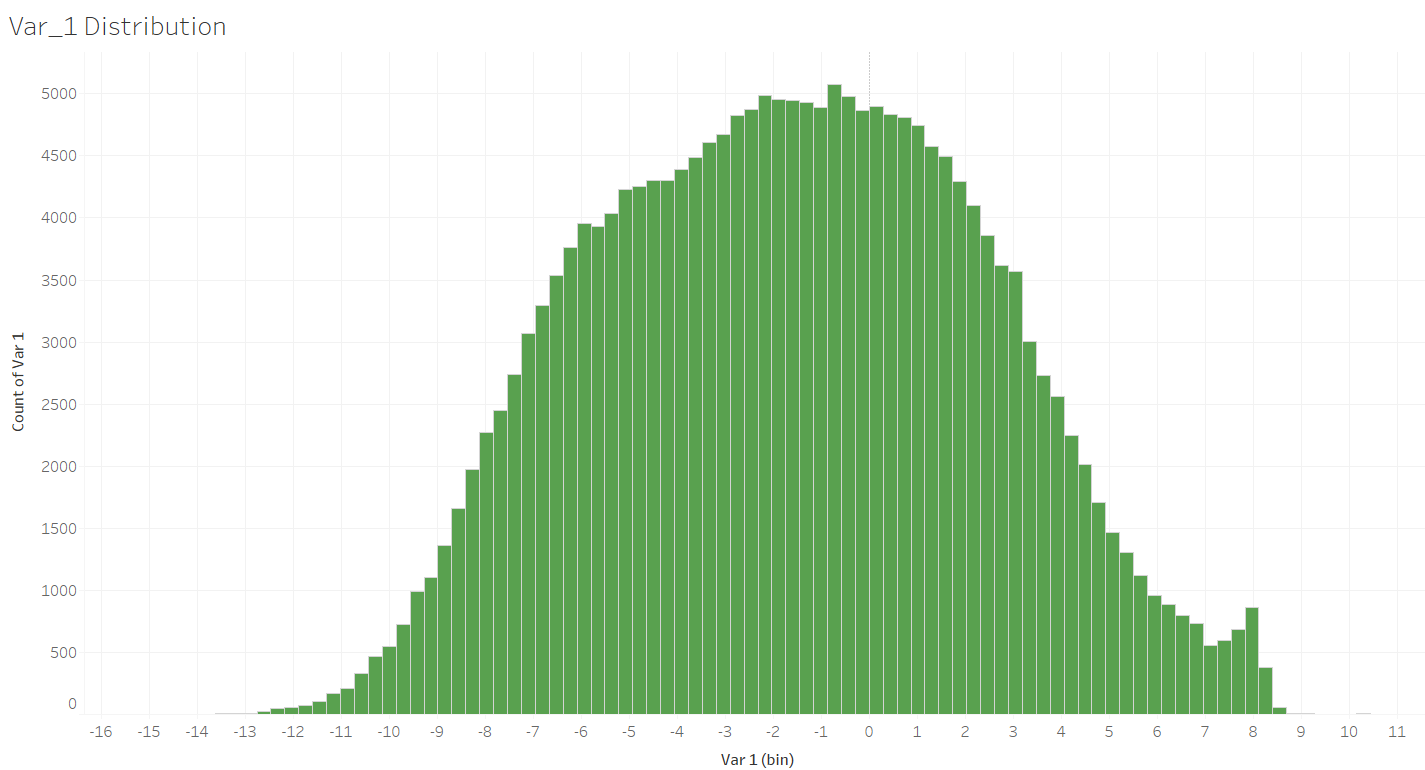
### **1.2.1 Exploratory Data Analysis:**

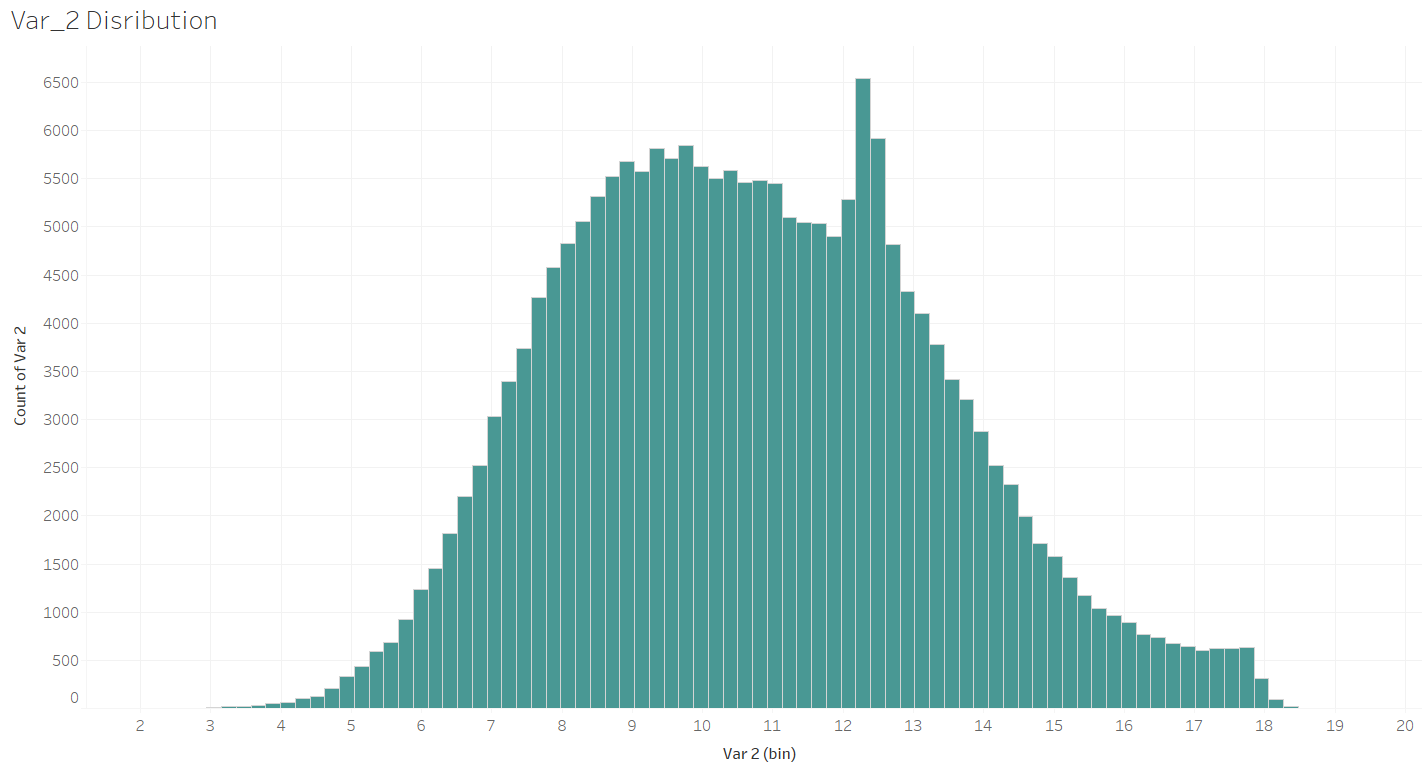
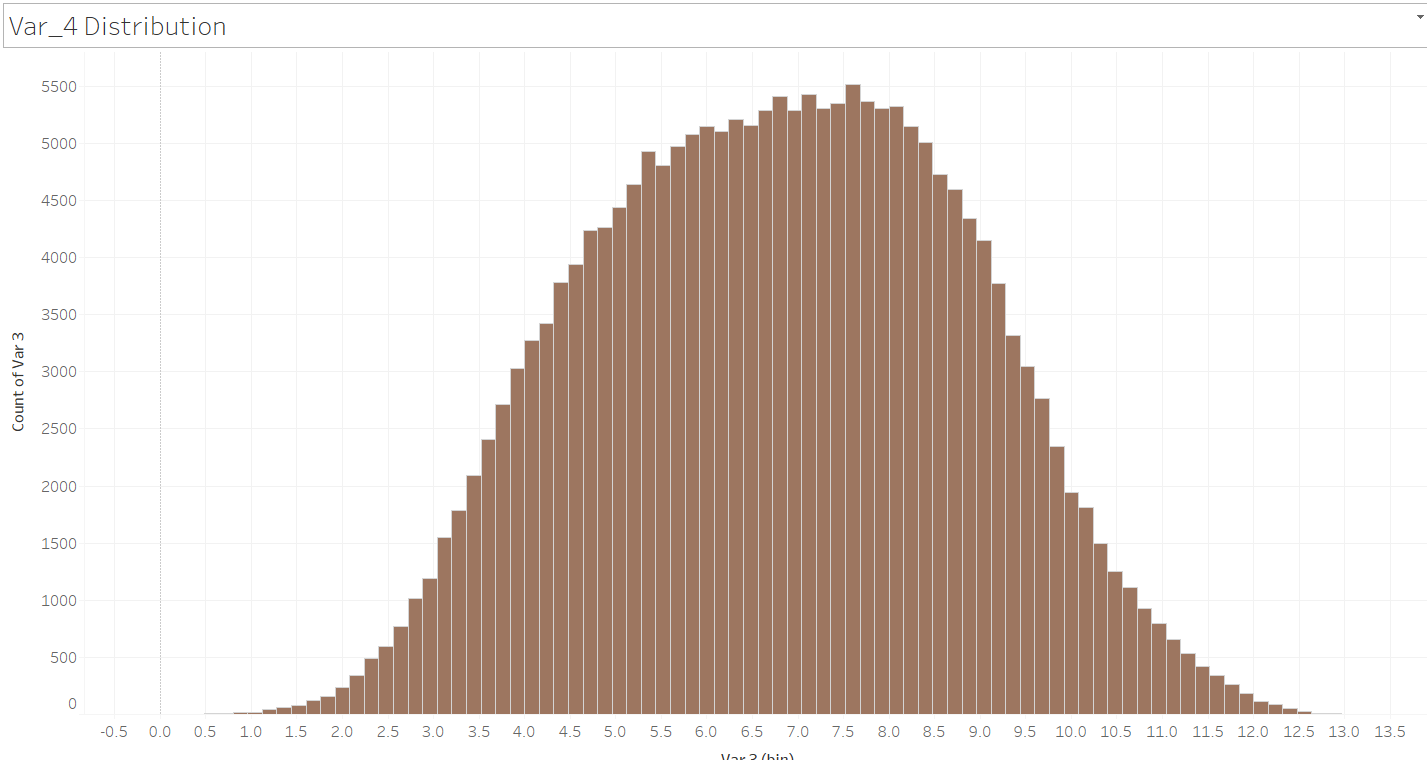
Here, we are trying to analyse the data and getting insights from them like how each variable is related to each other and to the target variable. Initially we are checking the distribution of the target variable of train data.



*Fig 1.1 Pick up Year Vs Fare amount*

It can be observed that the distribution of target variable is biased towards 0 as it occupies 89.95% of total data and the percentage of target value 1 is very less, being 10.05%. this is a serious issue in predicting the output as the model will be biased towards the majority class, i.e., 0, which has to be addressed while preparing the data.

*Fig 1.2 Distribution of few Variables*

Almost all variables are normally distributed in the train data set. First few variables and its histogram are shown above in detail. Kernel distribution of all variables are shown below.



*Fig 1.3 Distribution of first 100 predictor variables*



*Fig 1.4 Distribution of next 100 predictor variables*

# **Chapter 2**

# **Methodology**

## **2.1 Pre-Processing**

Any data modelling requires that we look at the data before we start modelling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. To start this process we will ﬁrst try and look at all the probability distributions of the variables separately for target variable categories.





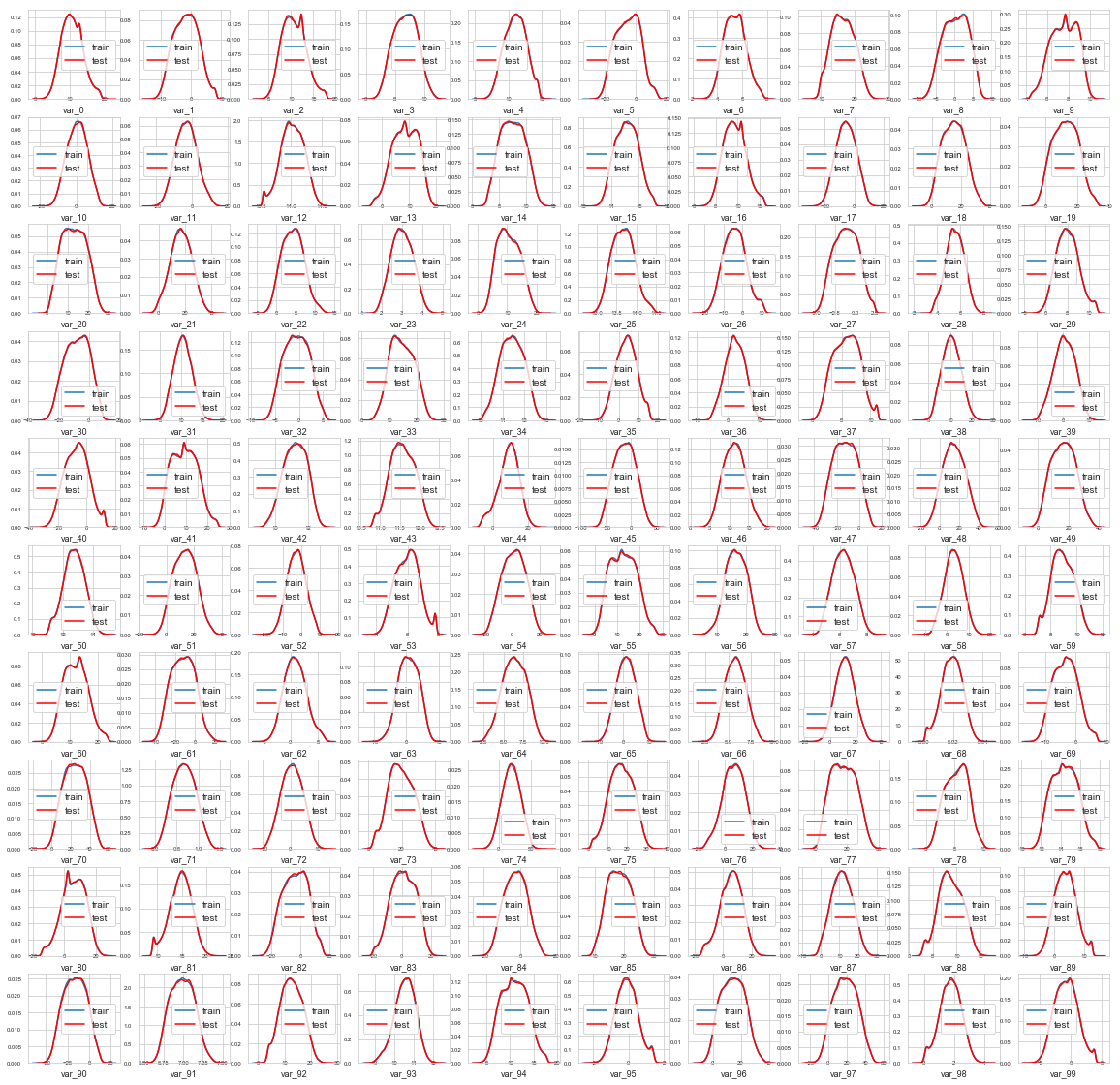
*Fig 2.1 Density Func. of continuous predictor variables for each categories of target variable (0 & 1)*

We can observe that there is a considerable number of features with significant different distribution for the two target values.

For example, var\_0, var\_1, var\_2, var\_5, var\_9, var\_13, var\_106, var\_109, var\_139 and many others.

Also some features, like var\_2, var\_13, var\_26, var\_55, var\_175, var\_184, var\_196 shows a distribution that resembles to a bivariate distribution.

We will investigate the distribution of the same features in parallel in train and test datasets and it happens that the train and test seems to be well balanced with respect with distribution of the numeric variables.

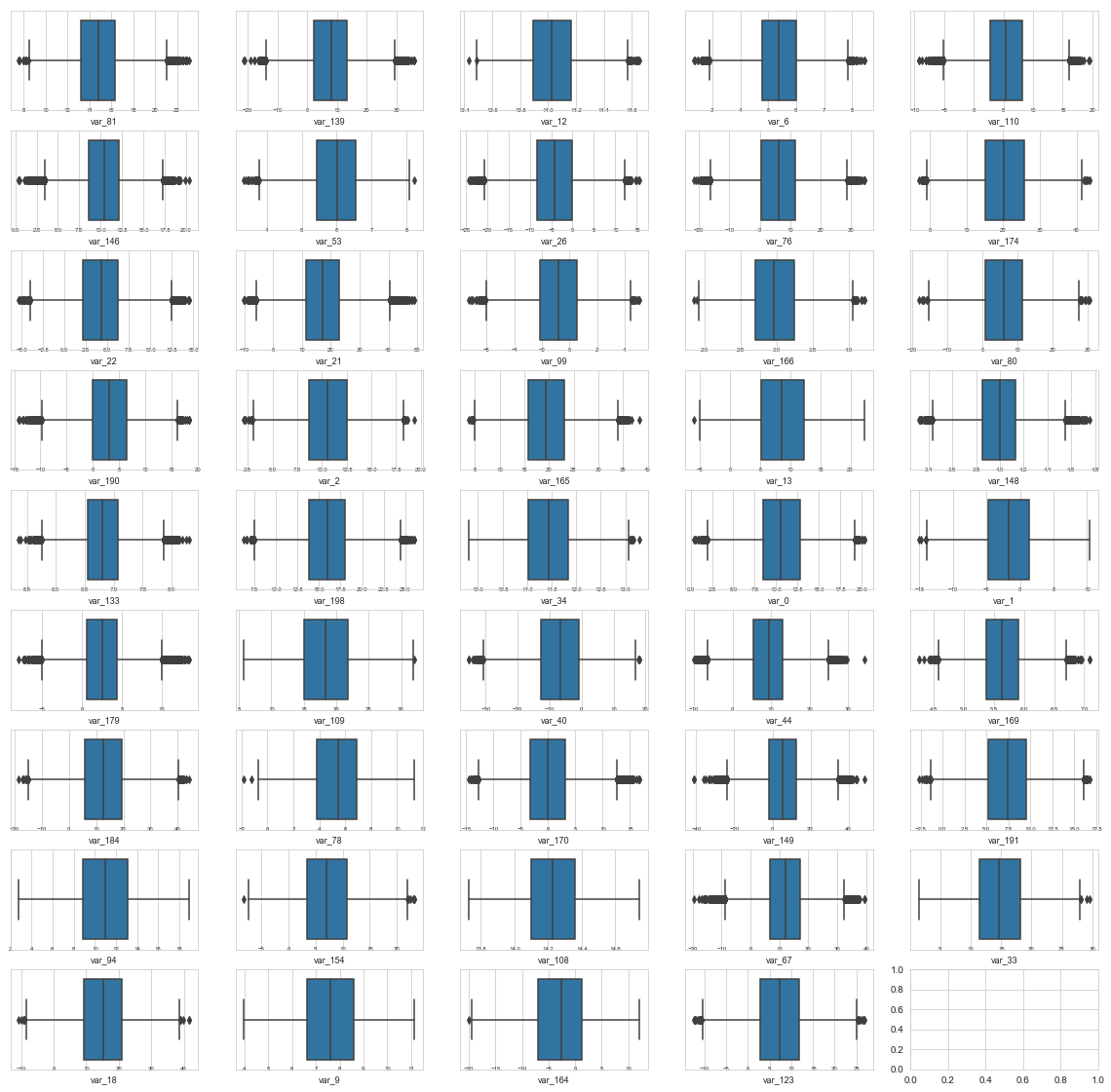


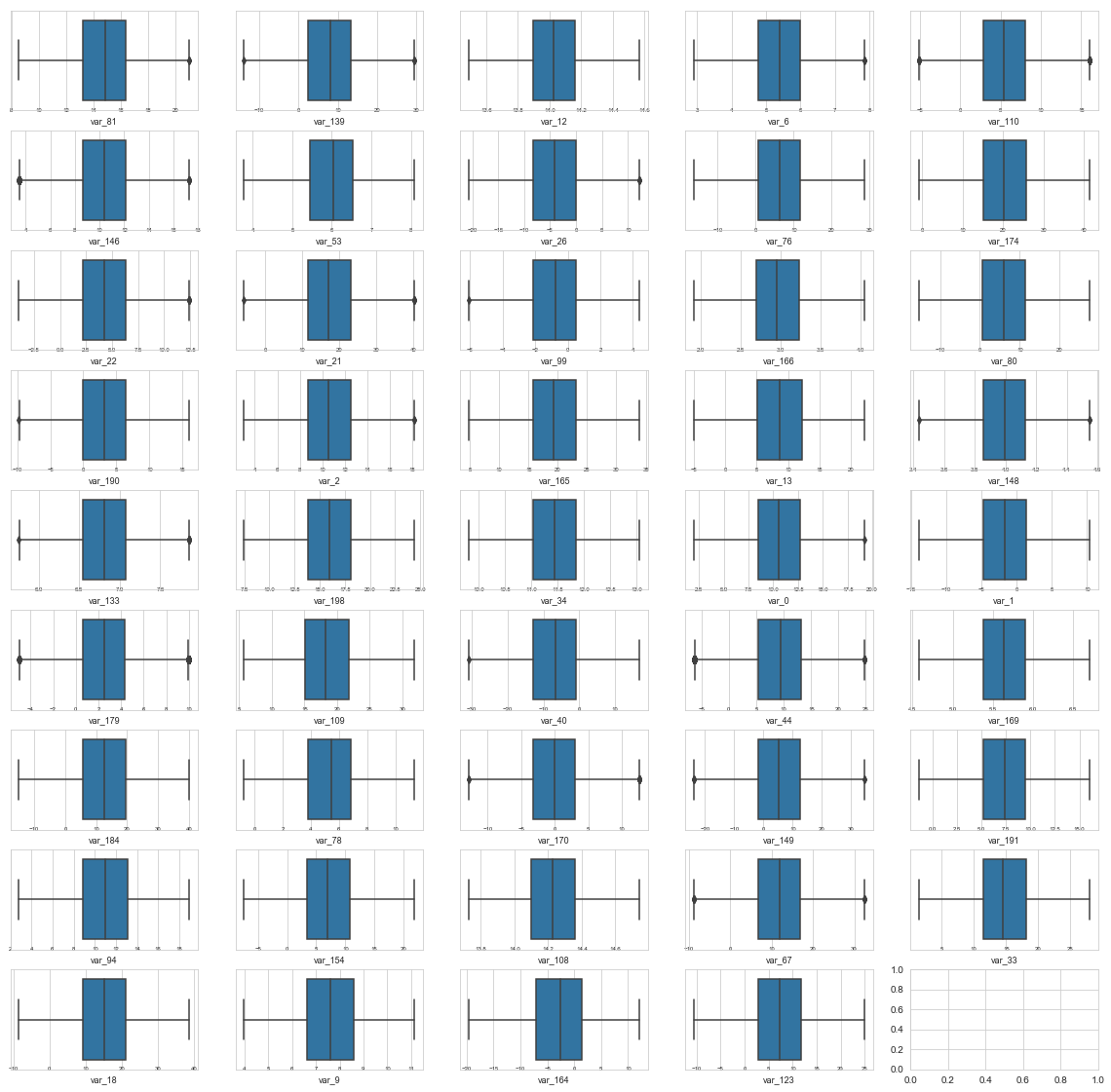
*Fig 2.2 Density Func. of test and train variables*

### **2.1.1 Outlier Analysis**

An outlier is an element of a data set that distinctly stands out from the rest of the data. In other words, outliers are those data points that lie outside the overall pattern of distribution as shown in figure below. The easiest way to detect outliers is to create a graph. Plots such as Box plots, Scatterplots and Histograms can help to detect outliers. In our case, we have used Box plots for various predictor variables. Here all the variables being only numerically described, we can find outliers in all the fields and remove them.

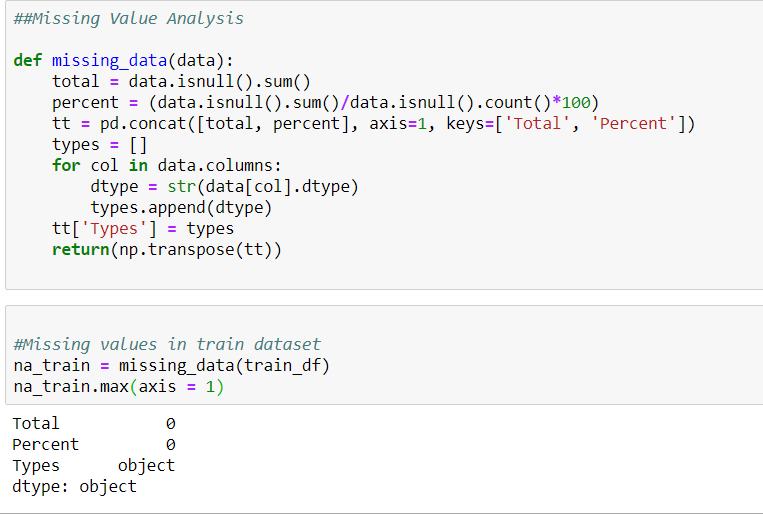
Below is the picture of Box plots of few variables from the train data set.

*Fig 2.2 Outlier analysis – Box Plots showing outliers of few variables*

*Fig 2.3 Outlier analysis – Box Plots after removing Outliers*

### **2.1.2 Missing Value Analysis**

Missing value analysis helps address several concerns caused by incomplete data. If cases with missing values are systematically different from cases without missing values, the results can be misleading. Also, missing data may reduce the precision of calculated statistics because there is less information than originally planned. We checked for missing values in all the variables of the data set and there is missing value found for fare amount and passenger count. However, there is no missing values found in the data set provided to us.

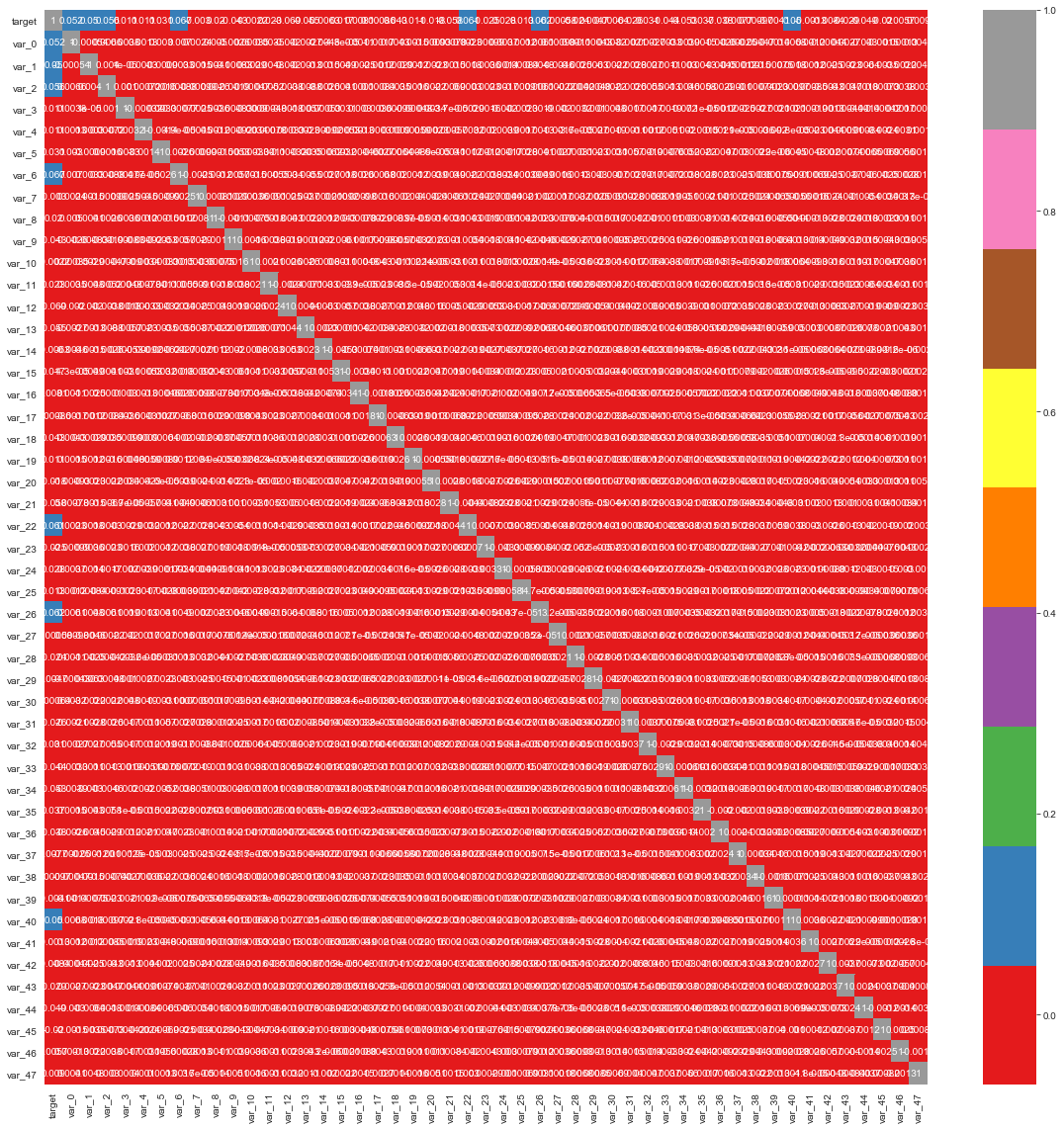


*Fig 2.4 Missing Value Analysis – No Values in train dataset*

### **2.1.3 Feature Selection**

Before performing any type of modelling we need to assess the importance of each predictor variable in our analysis. There is a possibility that many variables in our analysis are not important at all to the problem of class prediction. There are several methods of doing that and we have used a very simple method of finding the correlation between predictor variables through a correlation plot.

We performed a correlation plot for first 50 variables as calculating for entire dataset took lot of time. We could almost see no correlation among all the predictor variables and correlation between predictor and target variable is also very negligible.



*Fig 2.5 Correlation Heat map for first 50 predictor variables*

Since we couldn’t see much distinct results for all predictor variables from correlation plot, we cannot take proper decision on choosing the appropriate variables. Hence following two methods for feature selection here. The first one being, SelectKBest method, from which we will be choosing the first 50 variable that has Maximum Variable importance values.

|  |  |  |
| --- | --- | --- |
| **Column1** | **Variable** | **Score** |
| 81 | var\_81 | 1318.140513 |
| 139 | var\_139 | 1103.60377 |
| 12 | var\_12 | 970.428385 |
| 6 | var\_6 | 894.575796 |
| 110 | var\_110 | 829.682225 |
| 146 | var\_146 | 813.395373 |
| 53 | var\_53 | 807.112785 |
| 26 | var\_26 | 782.346677 |
| 76 | var\_76 | 769.697695 |
| 174 | var\_174 | 763.510862 |
| 22 | var\_22 | 736.156951 |
| 21 | var\_21 | 686.386893 |
| 99 | var\_99 | 683.663943 |
| 166 | var\_166 | 669.773502 |
| 80 | var\_80 | 665.974095 |
| 190 | var\_190 | 628.567666 |
| 2 | var\_2 | 626.24774 |
| 165 | var\_165 | 623.18056 |
| 13 | var\_13 | 610.293549 |
| 148 | var\_148 | 607.075107 |
| 133 | var\_133 | 596.858557 |
| 198 | var\_198 | 563.369262 |
| 34 | var\_34 | 556.832091 |
| 0 | var\_0 | 550.439149 |
| 1 | var\_1 | 508.158855 |
| 115 | var\_115 | 504.756482 |
| 179 | var\_179 | 501.28363 |
| 109 | var\_109 | 499.76182 |
| 40 | var\_40 | 491.85098 |
| 44 | var\_44 | 482.126986 |
| 169 | var\_169 | 469.260774 |
| 184 | var\_184 | 467.963435 |
| 78 | var\_78 | 466.590844 |
| 170 | var\_170 | 461.334646 |
| 149 | var\_149 | 448.819517 |
| 191 | var\_191 | 444.922865 |
| 94 | var\_94 | 429.575996 |
| 92 | var\_92 | 429.566692 |
| 154 | var\_154 | 426.054825 |
| 108 | var\_108 | 402.052129 |
| 67 | var\_67 | 399.936251 |
| 33 | var\_33 | 393.869751 |
| 18 | var\_18 | 378.797707 |
| 192 | var\_192 | 368.040838 |
| 9 | var\_9 | 367.131193 |
| 122 | var\_122 | 361.236029 |
| 173 | var\_173 | 353.794998 |
| 164 | var\_164 | 336.708468 |
| 118 | var\_118 | 326.279435 |
| 123 | var\_123 | 325.203042 |

*Tab 2.1 First 50 feature selected Variables*

We have employed another method, Extra tree classifier, to select first 50 variables just to cross verify if the same set of features are being selected. Then comparing both the list and selecting the common features. Likewise, there are 44 features selected and we will be using these 45 predictors to develop our model to predict the target variables.

### **2.1.4 Feature Scaling**

Since all the variables are numeric and there are many values to be taken into account for model creation, it would be convenient if we scale the values to same level. Here we are using Minimum – Maximum Scaling method to our data to scale the ranges of our numerical predictors same.

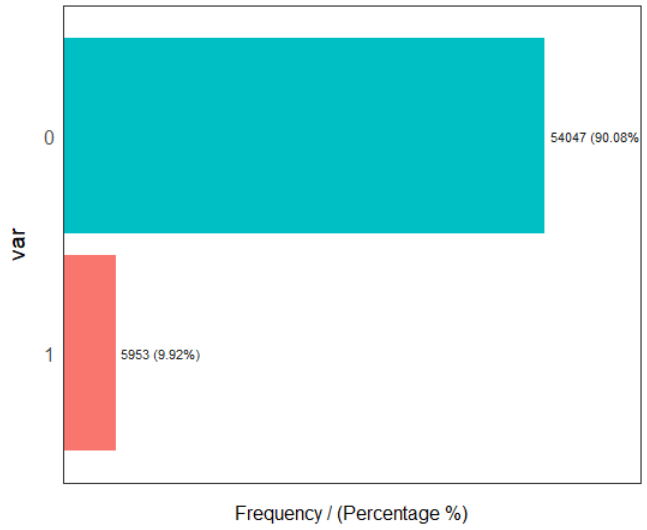
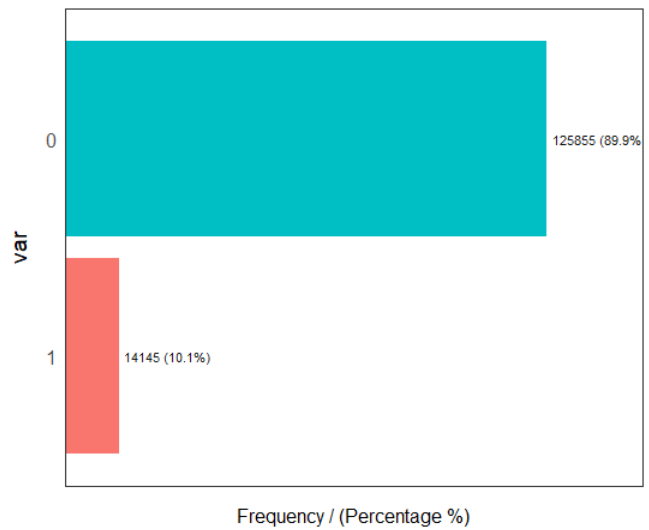
### **2.1.5 Sampling Train Data**

As we have seen above, the target variable is distributed unevenly, and we need to take extra care to sample the data as the results will be biased by the majority class. We are following stratified sampling technique to sample our dataset in order to retain the percentage of target class categories.

However, since the distribution of target variables remains biased, it will not give proper results. So we are performing under-sampling/over-sampling in addition to the stratified sampling. The distribution of down-sampled data, majority class 0 is being down-sampled, is shown below.

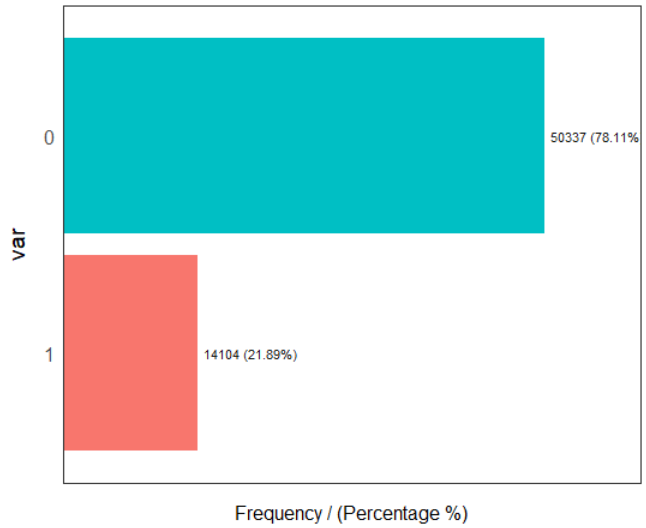
Undersampling:- it means taking the less number of majority class (In our case taking less number of Normal transactions so that our new data will be balanced

Oversampling: it means using replicating the data of minority class (fraud class) so that we can have a balanced data



*(a)Train data (b)Test data*

*Fig 2.6 Distribution of Stratified sample data*



*Fig 2.7 Down-Sampled train data distribution*

## **2.2 Modelling**

### **2.2.1 Model Selection**

Selecting a model for data set purely depends on the nature of the dependant variable or the variable that must be predicted. The dependent variable can fall in either of the four categories:

1. Nominal
2. Ordinal
3. Interval
4. Ratio

If the dependent variable(in our case target) is Nominal the only predictive analysis that we can perform is Classiﬁcation, and if the dependent variable is Interval or Ratio the normal method is to do a Regression analysis, or classiﬁcation after binning; and if the dependent variable is Ordinal, then both classiﬁcation and regression can be done.

In our case, the variable to be predicted being a binary factor variable, we are going for Classification techniques. You always start your model building from the simplest to more complex. Here we are going to use three regression techniques, namely Logistics Regression, Decision Tree algorithm, Random Forest.

### **2.2.2 Logistics Regression:**

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

summary(logit\_model)

Call:

glm(formula = target ~ ., family = "binomial", data = train)

Deviance Residuals:

Min 1Q Median 3Q Max

-2.3345 -0.7507 -0.4934 0.7545 3.1495

Coefficients:

Estimate Std. Error z value Pr(>|z|)

(Intercept) 21.014926 0.898237 23.40 <2e-16 \*\*\*

var\_81 -0.109216 0.004623 -23.62 <2e-16 \*\*\*

var\_139 -0.031155 0.001407 -22.14 <2e-16 \*\*\*

var\_6 0.237205 0.012524 18.94 <2e-16 \*\*\*

var\_12 -1.159826 0.056690 -20.46 <2e-16 \*\*\*

var\_53 0.290027 0.014130 20.53 <2e-16 \*\*\*

var\_76 -0.025879 0.001366 -18.95 <2e-16 \*\*\*

var\_146 -0.083772 0.004275 -19.60 <2e-16 \*\*\*

var\_110 0.053089 0.002812 18.88 <2e-16 \*\*\*

var\_22 0.072527 0.003776 19.21 <2e-16 \*\*\*

var\_26 0.032684 0.001808 18.08 <2e-16 \*\*\*

var\_174 -0.028411 0.001515 -18.76 <2e-16 \*\*\*

var\_99 0.101237 0.005805 17.44 <2e-16 \*\*\*

var\_21 -0.023802 0.001336 -17.81 <2e-16 \*\*\*

var\_80 -0.023768 0.001445 -16.45 <2e-16 \*\*\*

var\_190 0.040648 0.002384 17.05 <2e-16 \*\*\*

var\_166 -0.513642 0.029319 -17.52 <2e-16 \*\*\*

var\_13 -0.038648 0.002347 -16.47 <2e-16 \*\*\*

var\_34 -0.318286 0.020274 -15.70 <2e-16 \*\*\*

var\_165 -0.035828 0.002169 -16.52 <2e-16 \*\*\*

var\_2 0.067161 0.004086 16.44 <2e-16 \*\*\*

var\_198 -0.056118 0.003570 -15.72 <2e-16 \*\*\*

var\_133 0.473090 0.028815 16.42 <2e-16 \*\*\*

var\_148 -0.890062 0.054239 -16.41 <2e-16 \*\*\*

var\_0 0.054081 0.003530 15.32 <2e-16 \*\*\*

var\_44 -0.028437 0.001812 -15.70 <2e-16 \*\*\*

var\_40 0.019358 0.001310 14.77 <2e-16 \*\*\*

var\_179 0.058902 0.003847 15.31 <2e-16 \*\*\*

var\_1 0.038192 0.002702 14.14 <2e-16 \*\*\*

var\_184 0.017807 0.001167 15.26 <2e-16 \*\*\*

var\_78 0.084117 0.005507 15.27 <2e-16 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 60903 on 51847 degrees of freedom

Residual deviance: 50392 on 51817 degrees of freedom

AIC: 50454

Number of Fisher Scoring iterations: 5

From the output above, the coefficients table shows the beta coefficient estimates and their significance levels. Columns are:

Estimate: the intercept (b0) and the beta coefficient estimates associated to each predictor variable

Std.Error: the standard error of the coefficient estimates. This represents the accuracy of the coefficients. The larger the standard error, the less confident we are about the estimate.

z value: the z-statistic, which is the coefficient estimate (column 2) divided by the standard error of the estimate (column 3)

Pr(>|z|): The p-value corresponding to the z-statistic. The smaller the p-value, the more significant the estimate is.

This model provides the result in terms of probability which we need to convert to binary format, either 1 or 0, using a threshold value, which generally be 0.5 in most cases.

### **2.2.3 Decision Tree**

Decision tree is a type of supervised learning algorithm that can be used in both regression and classification problems. It works for both categorical and continuous input and output variables. Since our target field is categorical, we can apply decision trees for classification method.

We have used C5.0 algorithm for our classification decision tree method. This node uses the C5.0 algorithm to build either a decision tree or a rule set. A C5.0 model works by splitting the sample based on the field that provides the maximum information gain. Each subsample defined by the first split is then split again, usually based on a different field, and the process repeats until the subsamples cannot be split any further. Finally, the lowest-level splits are re-examined, and those that do not contribute significantly to the value of the model are removed or pruned.

Note: The C5.0 node can predict only a categorical target. When analysing data with categorical (nominal or ordinal) fields, the node is more likely to group categories together than versions of C5.0 prior to release 11.0.

C5.0 can produce two kinds of models. A decision tree is a straightforward description of the splits found by the algorithm. Each terminal (or "leaf") node describes a subset of the training data, and each case in the training data belongs to exactly one terminal node in the tree. In other words, exactly one prediction is possible for any particular data record presented to a decision tree.

In contrast, a rule set is a set of rules that tries to make predictions for individual records. Rule sets are derived from decision trees and, in a way, represent a simplified or distilled version of the information found in the decision tree. Rule sets can often retain most of the important information from a full decision tree but with a less complex model.

*Text documents: C5.0 Rules for decision trees*

### **2.2.4 Random Forest**

Random forest is a tree-based algorithm which involves building several trees (decision trees), then combining their output to improve generalization ability of the model. The method of combining trees is known as an ensemble method. For our modelling we have used 200 decision trees in our random forest algorithm.

# **Chapter 3**

# **Conclusion**

## **3.1 Model Evaluation**

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models, of which we are using confusion matrix and ROC-AUC curve.

### **3.1.1 Confusion Matrix**

A confusion matrix is an N X N matrix, where N is the number of classes being predicted. For the problem in hand, we have N=2, and hence we get a 2 X 2 matrix. This classification (or prediction) produces four outcomes – true positive, true negative, false positive and false negative.

* True positive (TP): correct positive prediction
* False positive (FP): incorrect positive prediction
* True negative (TN): correct negative prediction
* False negative (FN): incorrect negative prediction

Below are the few definitions we would be using to evaluate a model with the help of a confusion matrix :

* Accuracy : the proportion of the total number of predictions that were correct.

**ACC = (TP+TN)/(TP+TN+FP+FN)**

* Positive Predictive Value or Precision : the proportion of positive cases that were correctly identified of all positively predicted outcomes.

**Precision = TP/(TP+FP)**

* Negative Predictive Value : the proportion of negative cases that were correctly identified.

**FPR = FP/(FP+TN)**

* Sensitivity or Recall : the proportion of actual positive cases which are correctly identified.

**Recall or TPR = TP/(TP+FN)**

* Specificity : the proportion of actual negative cases which are correctly identified.

**FPR = FP/(FP+TN)**

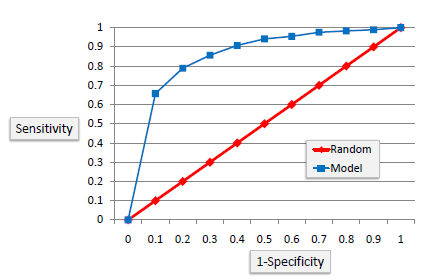
* F-Score: F-Score is the weighted average between Precision and Recall. A higher value of F-score indicate that the model is stronger in classifying correctly.

**F1 = 2\*((Precision\*TPR)/(Precision+TPR))**

### **3.1.2 ROC\_AUC Curve**

ROC Chart

The ROC chart is similar to the gain or lift charts in that they provide a means of comparison between classification models. The ROC chart shows false positive rate (1-specificity) on X-axis, the probability of target=1 when its true value is 0, against true positive rate (sensitivity) on Y-axis, the probability of target=1 when its true value is 1. Ideally, the curve will climb quickly toward the top-left meaning the model correctly predicted the cases. The diagonal red line is for a random model (ROC101).



*Fig 3.1 ROC Curve*

Area Under the Curve (AUC)

Area under ROC curve is often used as a measure of quality of the classification models. A random classifier has an area under the curve of 0.5, while AUC for a perfect classifier is equal to 1. In practice, most of the classification models have an AUC between 0.5 and 1.

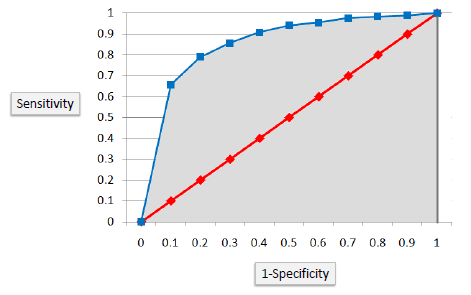


Fig 3.2 Area under ROC Curve

## **Model Selection**

We cannot use the accuracy from the confusion metrics as the accuracy is biased for the 0 target value for a biased data, Hence we are performing our model evaluation on the over/under-sampled data and using ROC Curve and AUC value.

### **3.2.1 Logistics Regression Model**

Output without over sampling data:

Accuracy : 90.60636261528484

Sensitivity or Recall/TPR : 0.09789422922359749

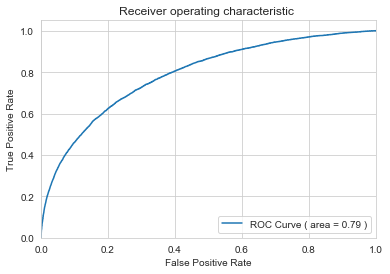
Specificity or TNR : 0.9942798989331836

Fall-Out or FPR : 0.005720101066816395

Miss rate or False Negative rate : 0.9021057707764025

Precision : 0.651336898395722

F1 Score: 0.17020681945220792



Output with Over-sampled data:

Accuracy : 75.25968556990455

Sensitivity or Recall/TPR : 0.48589275687815836

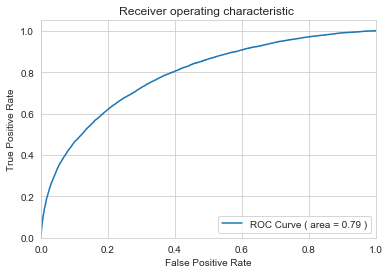
Specificity or TNR : 0.8859489051094891

Fall-Out or FPR : 0.11405109489051095

Miss rate or False Negative rate : 0.5141072431218416

Precision : 0.6805268848913791

F1 Score: 0.5669710495065723



### **3.2.2 Decision Tree Method**

Output without over sampling data:

Accuracy : 84.00012655624634

Sensitivity or Recall/TPR : 0.23340298987301078

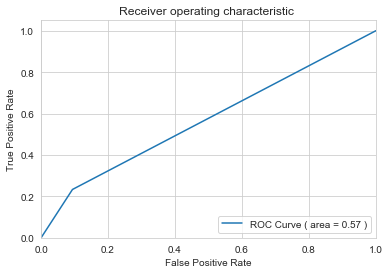
Specificity or TNR : 0.9062149073554183

Fall-Out or FPR : 0.0937850926445817

Miss rate or False Negative rate : 0.7665970101269892

Precision : 0.2136236574959541

F1 Score: 0.2230757412813028



Output with Over-sampled data:

Accuracy : 90.85602657682949

Sensitivity or Recall/TPR : 0.9659250421111735

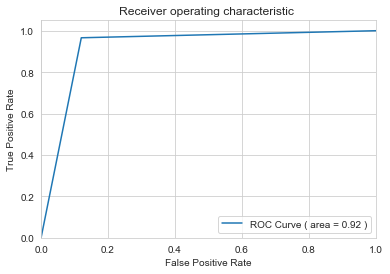
Specificity or TNR : 0.8798778775968557

Fall-Out or FPR : 0.12012212240314431

Miss rate or False Negative rate : 0.0340749578888265

Precision : 0.800820459107969

F1 Score: 0.8756581354287623



### **3.2.3 Random Forest Method**

Output without over sampling data:

Accuracy : 90.22985778241818

Sensitivity or Recall/TPR : 0.008198038900498313

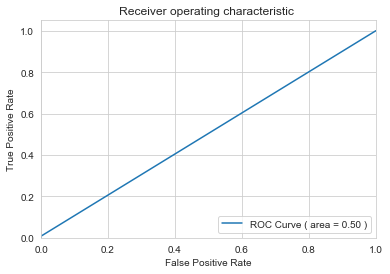
Specificity or TNR : 0.999894722066255

Fall-Out or FPR : 0.00010527793374508703

Miss rate or False Negative rate : 0.9918019610995017

Precision : 0.8947368421052632

F1 Score: 0.01624721248805352



Output with Over-sampled data:

Accuracy : 98.40679393599102

Sensitivity or Recall/TPR : 0.9546953958450309

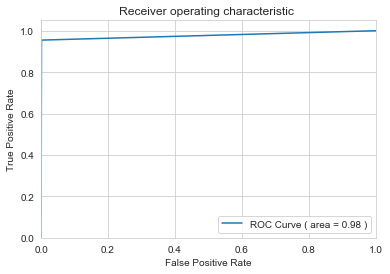
Specificity or TNR : 0.9987542111173499

Fall-Out or FPR : 0.0012457888826501965

Miss rate or False Negative rate : 0.04530460415496912

Precision : 0.9973969790291831

F1 Score: 0.975579143656315



It can be seen that, although accuracy is high for normal data, it is quite misleading as the model is biased towards negative cases. We can consider the upsampled data results and observe that the F-Value is quite high for the models compared to the ones without up-sampling.

Moreover, Random forest shows relatively a higher F-value or the AUC value and we can chose that model to predict the customer transaction in future

# **Appendix A – R Code**

rm(list = ls())

#install required packages

library(dplyr)

library(funModeling)

library(ggplot2)

library(corrgram)

#install.packages('caret', dependencies = TRUE)

library(caret)

library(randomForest)

library(tidyr)

#library(Boruta)

#install.packages("Hmisc")

library(Hmisc)

#install.packages("corrplot")

library(corrplot)

#install.packages("pryr")

library(pryr)

#instal.packages("c50")

library(C50)

#install.packages("PRROC")

#install.packages("pROC")

library(PRROC)

library(pROC)

#Set Working directory

setwd("E:/EDW/Projects/Santander Prediction")

#getwd()

#Load data

train\_df <- read.csv("train.csv", header = T)

test\_df <- read.csv("test.csv", header = T)

train\_df$ID\_code <- as.character(train\_df$ID\_code)

train\_df$target <- as.character(train\_df$target)

train\_df$target <- as.factor(train\_df$target)

#EDA - Train data

str(train\_df)

glimpse(train\_df) #to view data of all columns

df\_1 <- df\_status(train\_df) #gives no. of zeros(& %value), na values(& % value), infinity values, type and no of unique entries in each column

freq(train\_df$target) #percentage of target variable

train\_summary <- profiling\_num(train\_df) #Summary of train data

# row.names(train\_summary) <- train\_summary$variable

# train\_summary <- train\_summary[-1]

# train\_summary <- data.frame(t(train\_summary))

#EDA - Test Data

glimpse(test\_df) #to view data of all columns

df\_2 <- df\_status(test\_df) #gives no. of zeros(& %value), na values(& % value), infinity values, type and no of unique entries in each column

test\_summary <- profiling\_num(test\_df) #Summary of train data

row.names(test\_summary) <- test\_summary$variable

test\_summary <- test\_summary[-1]

test\_summary <- data.frame(t(test\_summary))

###############################################################################################################

#get the variables that have missing values

var\_with\_na <- subset(df\_1, df\_1$q\_na > 0)

print("Missing Value details:")

var\_with\_na["variable"]

###############################################################################################################

#Visualizations:

#Density plots of individual features

ggplot(gather(train\_df[,3:30]), aes(value)) + geom\_freqpoly() + facet\_wrap(~key, scales = 'free\_x')

ggplot(gather(train\_df[,31:60]), aes(value)) + geom\_freqpoly() + facet\_wrap(~key, scales = 'free\_x')

ggplot(gather(train\_df[,61:90]), aes(value)) + geom\_freqpoly() + facet\_wrap(~key, scales = 'free\_x')

ggplot(gather(train\_df[,91:120]), aes(value)) + geom\_freqpoly() + facet\_wrap(~key, scales = 'free\_x')

ggplot(gather(train\_df[,121:150]), aes(value)) + geom\_freqpoly() + facet\_wrap(~key, scales = 'free\_x')

ggplot(gather(train\_df[,151:180]), aes(value)) + geom\_freqpoly() + facet\_wrap(~key, scales = 'free\_x')

ggplot(gather(train\_df[,181:202]), aes(value)) + geom\_freqpoly() + facet\_wrap(~key, scales = 'free\_x')

#Distribution of mean,standard deviation and ranges:

ggplot(train\_summary, aes(x=train\_summary$variable, y=train\_summary$std\_dev)) +geom\_point()

ggplot(train\_summary, aes(x=train\_summary$variable, y=train\_summary$mean)) +geom\_point(colour = 'red')

ggplot(train\_summary, aes(x=train\_summary$variable, y=train\_summary$range\_98)) +geom\_point()

##########################################################################################################

#Numerical data:

numeric\_data = train\_df[,-c(1,2)]

cnames = colnames(numeric\_data)

### Correlation Plot

corr\_data <- cor(numeric\_data)

palette = colorRampPalette(c("green", "white", "red")) (20)

heatmap(x = corr\_data, col = palette, symm = TRUE)

# corr\_data\_1 <- rcorr(as.matrix(numeric\_data))

# corrplot(corr\_data\_1)

############################################################################################################

##Feature Selection - Correlation using Logistic regression

numeric\_data = train\_df[,-1]

#numeric\_data$target = as.numeric(numeric\_data$target)

fit\_log = glm(target~., numeric\_data, family = binomial)

v\_imp <- varImp(fit\_log)

v\_imp$Variable <- row.names(v\_imp)

v\_imp <- v\_imp[ , c(2,1)]

v\_imp <- v\_imp[order(-v\_imp$Overall), ]

row.names(v\_imp) <- NULL

feature <- v\_imp$Variable[1:30]

# ##Feature Selection -Random Forest Classifier

# # numeric\_data$target <- as.character(numeric\_data$target)

# # numeric\_data$target <- as.factor(numeric\_data$target)

# # rf\_model <- randomForest(target~., data = numeric\_data)

#

#New train data with only selected variables

train\_df\_1 <- cbind(train\_df[,c(1,2)], train\_df[,feature])

############################################################################################################

# ## BoxPlots - Distribution and Outlier Check

numeric\_index = sapply(train\_df\_1,is.numeric) #selecting only numeric

numeric\_data = train\_df\_1[,numeric\_index]

cnames = colnames(numeric\_data)

ln = length(cnames)

for (i in 1:6)

{

assign(paste0("gn",i), ggplot(aes\_string(y = (cnames[i]), x = "target"), data = subset(train\_df\_1))+

stat\_boxplot(geom = "errorbar", width = 0.5) +

geom\_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,

outlier.size=1, notch=FALSE) +

theme(legend.position="bottom")+

labs(y=cnames[i],x="target")+

ggtitle(paste("Box plot of target for",cnames[i])))

}

gridExtra::grid.arrange(gn1,gn2,gn3,ncol=3)

gridExtra::grid.arrange(gn4,gn5,gn6,ncol=3)

gridExtra::grid.arrange(gn22,gn23,gn24,ncol=3)

gridExtra::grid.arrange(gn25,gn26,gn27,ncol=3)

gridExtra::grid.arrange(gn28,gn29,gn30,ncol=3)

#Outlier:

# # #loop to remove outliers from all variables in feature selected dataset

for(i in (3:length(cnames))){

print(cnames[i-2])

val = train\_df\_1[,i][train\_df\_1[,i] %in% boxplot.stats(train\_df\_1[,i])$out]

print(length(val))

train\_df\_1 = train\_df\_1[which(!train\_df\_1[,i] %in% val),]

}

# for(i in cnames){

# q = quantile(train\_df\_1[,i], c(0.25,0.75), names = FALSE)

# iqr = q[2] - q[1]

# mini = q[1] - (iqr\*1.5)

# maxi = q[2] + (iqr\*1.5)

# print(i)

# print(mini)

# print(maxi)

#

# }

nrow(train\_df\_1) #193026 ## after removal of

###############################################################################################################

#Correlation Plot after removing outliers:

numeric\_index = sapply(train\_df\_1,is.numeric) #selecting only numeric

numeric\_data = train\_df\_1[,numeric\_index]

corr\_data\_1 <- cor(numeric\_data)

palette = colorRampPalette(c("green", "white", "red")) (20)

heatmap(x = corr\_data\_1, col = palette, symm = TRUE)

###############################################################################################################

#Feature Scaling - MinMax Method

for(i in cnames){

print(i)

train\_df\_1[,i] = (train\_df\_1[,i] - min(train\_df\_1[,i]))/

(max(train\_df\_1[,i] - min(train\_df\_1[,i])))

}

#########################################################################################################

train\_df\_1$target <- as.character(train\_df\_1$target)

train\_df\_1$target <- as.factor(train\_df\_1$target)

##Simple Random Sampling

#random\_sampled = train\_df[sample(nrow(train\_df), 100000, replace = F), ]

#Divide data into train and test using stratified sampling method

set.seed(1234)

train.index = createDataPartition(train\_df\_1$target, p = .70, list = FALSE)

train = train\_df\_1[ train.index,]

test = train\_df\_1[-train.index,]

#Plot of Stratified sample Train and test target variables

freq(train$target) #percentage of target variable

freq(test$target) #percentage of target variable

#######################################################################################################

##\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Models\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*##

train = train[, -1]

test = test[, -1]

#Logistic Regression

logit\_model = glm(target ~ ., data = train, family = "binomial")

#summary of the model

summary(logit\_model)

#predict using logistic regression

logit\_Predictions = predict(logit\_model, newdata = test, type = "response")

#convert prob

logit\_Predictions = ifelse(logit\_Predictions > 0.5, 1, 0)

#Function to calculate metrics

metrics <- function(x\_act, x\_pred){

ConfMatrix = table(x\_act, x\_pred)

TN = ConfMatrix[1]

FN = ConfMatrix[3]

TP = ConfMatrix[4]

FP = ConfMatrix[2]

print(paste("True Negative : ", TN))

print(paste("True Positive : ", TP))

print(paste("False Negative:", FN))

print(paste("False Positive:", FP))

ACC = (TP+TN)/(TP+TN+FP+FN)

TPR = TP/(TP+FN)

FPR = FP/(FP+TN)

TNR = TN/(TN+FP)

FNR = FN/(FN+TP)

Precision = TP/(TP+FP)

F1 = 2\*((Precision\*TPR)/(Precision+TPR))

print(paste("Accuracy :", ACC))

print(paste("Sensitivity or Recall/TPR :", TPR))

print(paste("Specificity or TNR :", TNR))

print(paste("Fall-Out or FPR :", FPR))

print(paste("Miss rate or False Negative rate :", FNR))

print(paste("Precision :" , Precision))

print(paste("F1 Score: " , F1))

}

##Evaluate the performance of classification model

ConfMatrix\_LOGIT = metrics(test$target, logit\_Predictions)

ROC1 <- roc.curve(test$target, logit\_Predictions, curve = TRUE)

plot(ROC1, lwd = 4)

# logit\_Predictions <- as.character(logit\_Predictions)

# logit\_Predictions <- as.factor(logit\_Predictions)

# confusionMatrix(test$target, logit\_Predictions)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

##Decision tree for classification

#Develop Model on training data

ctrl = C5.0Control(subset = TRUE, CF = 0.5, minCases = 2)

C50\_model = C5.0(target ~., train, trials = 2, rules = TRUE, control = ctrl)

#plot(C50\_model)

#Summary of DT model

summary(C50\_model)

#write rules into disk

write(capture.output(summary(C50\_model)), "c50Rules\_trial10.txt")

#Lets predict for test cases

C50\_Predictions = predict(C50\_model, test[,-1], type = "class")

confMatrix\_C50 = metrics(test$target, C50\_Predictions)

ROC2 <- roc.curve(test$target, C50\_Predictions, curve = TRUE)

plot(ROC2)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

###Random Forest

RF\_model = randomForest(target ~ ., train, importance = TRUE, ntree = 200)

#Predict test data using random forest model

RF\_Predictions = predict(RF\_model, test[,-1])

confMatrix\_RF = metrics(test$target, RF\_Predictions)

ROC3 <- roc.curve(test$target, RF\_Predictions, curve = TRUE)

plot(ROC3)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# #Naive Bayes Model

# library(e1071)

# #Develop model

# NB\_model = naiveBayes(target ~ ., data = train)

#

# #predict on test cases #raw

# NB\_Predictions = predict(NB\_model, test[,2:31], type = 'class')

#

# confMatrix\_NB = metrics(test$target, NB\_Predictions)

#

# ROC4 <- roc.curve(test$target, NB\_Predictions, curve = TRUE)

# plot(ROC4)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# #kNN Model

# library(class)

# #Predict test data

# KNN\_Predictions = knn(train[, 2:31], test[, 2:31], train$target, k = 7)

#

# confMatrix\_knn = metrics(test$target, KNN\_Predictions)

#

# ROC5 <- roc.curve(test$target, KNN\_Predictions, curve = TRUE)

# plot(ROC5)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Downsampling:

train\_00 <- train\_df\_1[train\_df\_1$target==0, ] #take entries with only target = 0 values

train\_11 <- train\_df\_1[train\_df\_1$target==1, ] #take entries with only target = 1 values

index = sample(1:nrow(train\_00), 0.3 \* nrow(train\_00))

train\_00 = train\_00[index,]

undersampled\_df = rbind(train\_00, train\_11)

#Divide undersampled data into train and test using stratified sampling method

set.seed(2222)

train.index = createDataPartition(undersampled\_df$target, p = .70, list = FALSE)

train = undersampled\_df[ train.index,]

# test = undersampled\_df[-train.index,]

#Plot of Stratified sample Train and test target variables

freq(train$target) #percentage of target variable

freq(test$target) #percentage of target variable

train = train[, -1]

test = test[, -1]

##############################################################################################

# **Appendix B – Python Code**

# #Load libraries\n",

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import statsmodels.api as sm

from sklearn.metrics import roc\_curve

from sklearn.metrics import roc\_auc\_score

from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier

# Load scikit's random forest classifier library

from sklearn.ensemble import RandomForestClassifier

#Set working directory

os.chdir("E:\EDW\Projects\Santander Prediction")

#os.getcwd()

#'E:\\EDW\\Projects\\Santander Prediction'

%%time

train\_df = pd.read\_csv("train.csv")

test\_df = pd.read\_csv("test.csv")

train\_df.shape #(200000, 202)

test\_df.shape #(200000, 201)

train\_df.head()

#Exploratory data analysis:

desc\_train = train\_df.describe()

desc\_test = test\_df.describe()

##Missing Value Analysis

def missing\_data(data):

total = data.isnull().sum()

percent = (data.isnull().sum()/data.isnull().count()\*100)

tt = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

types = []

for col in data.columns:

dtype = str(data[col].dtype)

types.append(dtype)

tt['Types'] = types

return(np.transpose(tt))

#Missing values in train dataset

na\_train = missing\_data(train\_df)

na\_train.max(axis = 1)

#Missing values in test dataset

na\_test = missing\_data(test\_df)

na\_test.max(axis = 1)

#Scatter Plots

def plot\_feature\_scatter(df1, df2, features):

i = 0

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(5,5,figsize=(20,20))

for feature in features:

i += 1

plt.subplot(5,5,i)

plt.scatter(df1[feature], df2[feature], marker='+')

plt.xlabel(feature, fontsize=15)

plt.show();

#Correlation Plot

features = train\_df.columns

features = features.tolist()

features = features[2:]

feature\_1 = features[0:25] #Splitting the features into smaller sets to plot correlation plot

feature\_2 = features[26:51]

feature\_3 = features[52:77]

feature\_4 = features[78:103]

plot\_feature\_scatter(train\_df[::20],test\_df[::20], feature\_1)

plot\_feature\_scatter(train\_df[::20],test\_df[::20], feature\_2)

plot\_feature\_scatter(train\_df[::20],test\_df[::20], feature\_3)

# Compute the Correlation matrix

corr=train\_df.iloc[:,:50].corr()

plt.figure(figsize=(20,20))

sns.heatmap(corr,cmap='Set1',annot=True)

#Distribution of target variable

sns.countplot(train\_df['target'], palette='tab10')

#Density Plot:

def plot\_predictor\_distribution(df1, features):

i = 0

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(10,10,figsize=(20,20))

for feature in features:

i += 1

plt.subplot(10,10,i)

sns.distplot(df1[feature], hist=False)

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();

features = train\_df.columns.values[102:202]

plot\_predictor\_distribution(train\_df, features)

#Density Plot:

def plot\_feature\_distribution(df1, df2, label1, label2, features):

i = 0

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(10,10,figsize=(20,20))

for feature in features:

i += 1

plt.subplot(10,10,i)

sns.distplot(df1[feature], hist=False,label=label1)

sns.distplot(df2[feature], hist=False,label=label2, color='r')

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();

t0 = train\_df.loc[train\_df['target'] == 0]

t1 = train\_df.loc[train\_df['target'] == 1]

features = train\_df.columns.values[2:102]

plot\_feature\_distribution(t0, t1, '0', '1', features)

features = train\_df.columns.values[103:202]

plot\_feature\_distribution(t0, t1, '0', '1', features)

features = train\_df.columns.values[2:102]

plot\_feature\_distribution(train\_df, test\_df, 'train', 'test', features)

%%time

#Feature selection1::

from sklearn.feature\_selection import SelectKBest

from sklearn.feature\_selection import f\_classif

X = train\_df.iloc[:,2:202] #independent columns

y = train\_df.iloc[:,1] #target column

#apply SelectKBest class to extract top 20 best features

bestfeatures = SelectKBest(score\_func=f\_classif, k=50)

fit\_1 = bestfeatures.fit(X,y)

dfscores = pd.DataFrame(fit\_1.scores\_)

dfcolumns = pd.DataFrame(X.columns)

#concat two dataframes for better visualization

feat\_importance\_1 = pd.concat([dfcolumns,dfscores],axis=1)

feat\_importance\_1.columns = ['Variable','Score'] #naming the dataframe columns

feat\_importance\_1 = feat\_importance\_1.sort\_values(by = ['Score'], ascending = False)

print(feat\_importance\_1.nlargest(50,'Score'))

Var\_to\_model\_1 = feat\_importance\_1.Variable[0:50] #choosing first 50 variables with high importance values

#feature Selection2:

from sklearn.ensemble import ExtraTreesClassifier

X = train\_df.iloc[:,2:202] #independent columns

y = train\_df.iloc[:,1] #target column

y=y.astype('int64')

feat\_model = ExtraTreesClassifier( )

feat\_model.fit(X,y)

print(feat\_model.feature\_importances\_)

feat\_importance\_2 = pd.DataFrame(feat\_model.feature\_importances\_, index=X.columns)

feat\_importance\_2['Variable'] = feat\_importance\_2.index

feat\_importance\_2.columns = ['Score', 'Variable']

feat\_importance\_2 = feat\_importance\_2.sort\_values(by = ['Score'], ascending = False)

feat\_importance\_2

Var\_to\_model\_2 = feat\_importance\_2.Variable[0:50]

#dfscores = pd.DataFrame(fit\_1.scores\_)

#dfcolumns = pd.DataFrame(X.columns)

#concat two dataframes for better visualization

#feat\_importance\_1 = pd.concat([dfcolumns,dfscores],axis=1)

#feat\_importance\_1.columns = ['Variable','Score'] #naming the dataframe columns

#print(feat\_importance\_1.nlargest(50,'Score'))

#Variable selection from 2 methods:

Var\_to\_model\_1 = Var\_to\_model\_1.tolist( )

Var\_to\_model\_2 = Var\_to\_model\_2.tolist( )

Var\_to\_model=[i for i in Var\_to\_model\_1 if i in Var\_to\_model\_2]

Var\_to\_model #common variables in both variable selection methods

len(Var\_to\_model)

print(Var\_to\_model)

#Field selection from teh original data set

train\_df\_1 = train\_df[['ID\_code', 'target']]

train\_df\_2 = train\_df[Var\_to\_model]

train\_df\_1 = pd.concat([train\_df\_1, train\_df\_2], axis = 1)

test\_df\_1 = test\_df[['ID\_code']]

test\_df\_2 = test\_df[Var\_to\_model]

test\_df\_1 = pd.concat([test\_df\_1, test\_df\_2], axis = 1)

#Outlier Analysis:

def plot\_outlier(df1, features):

i = 0

sns.set\_style('whitegrid')

plt.figure()

fig, ax = plt.subplots(9,5,figsize=(20,20))

for var in features:

i += 1

plt.subplot(9,5,i)

sns.boxplot(x=df1[var])

plt.xlabel(var, 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();

plot\_outlier(train\_df\_1, Var\_to\_model)

#remove Outliers:

for i in Var\_to\_model:

q75, q25 = np.percentile(train\_df\_1.loc[:, i], [75 ,25])

iqr = q75 - q25

min = q25 - (iqr\*1.5)

max = q75 + (iqr\*1.5)

print(i)

print(min)

print(max)

train\_df\_1 = train\_df\_1.drop(train\_df\_1[train\_df\_1.loc[:,i] < min].index)

train\_df\_1 = train\_df\_1.drop(train\_df\_1[train\_df\_1.loc[:,i] > max].index)

#After removing outliers:

plot\_outlier(train\_df\_1, Var\_to\_model)

train\_df\_1.shape

#Stratified sampling

from sklearn.model\_selection import StratifiedShuffleSplit

ss = StratifiedShuffleSplit(n\_splits=10, test\_size=0.33, random\_state= 100)

for train\_index, test\_index in ss.split(train\_df\_1, train\_df\_1['target']):

train = train\_df\_1.iloc[train\_index]

test = train\_df\_1.iloc[test\_index]

#Target variable distribution after stratified sampling

sns.countplot(train['target'], palette='tab10')

sns.countplot(test['target'], palette='tab10')

#Model \_1

#Logistic regression:

def logistic\_regression(y,X):

global result

logit\_model=sm.Logit(y,X)

result=logit\_model.fit()

print(result.summary2())

#Model 2:

#Decision tree:

def decision\_tree\_mod(X\_train,y\_train,X\_test):

global y\_pred, clf

clf = DecisionTreeClassifier()

# Train Decision Tree Classifer

clf = clf.fit(X\_train,y\_train)

#Predict the response for test dataset

y\_pred = clf.predict(X\_test)

#Model 3:

def random\_forest\_mod(X,y,x\_test):

global y\_test, clf

# Create a random forest Classifier. By convention, clf means 'Classifier'

clf = RandomForestClassifier(n\_estimators=100, n\_jobs=2, random\_state=0)

# Train the Classifier to take the training features and learn how they relate

# to the training y (the species)

clf.fit(X, y)

# Apply the Classifier we trained to the test data (which, remember, it has never seen before)

y\_test = clf.predict(x\_test)

#Build confusion matrix

def confusion\_matrix(yy,xx):

CM = pd.crosstab(yy,xx)

global TN, FN, TP, FP

#let us save TP, TN, FP, FN

TN = CM.iloc[0,0]

FN = CM.iloc[1,0]

TP = CM.iloc[1,1]

FP = CM.iloc[0,1]

print("True Negative", TN)

print("True Positive", TP)

print("False Negative", FN)

print("False Positive", FP)

#check accuracy of model

#accuracy\_score(y\_test, y\_pred)\*100

print("Accuracy : ", ((TP+TN)\*100)/(TP+TN+FP+FN))

def Metrics(TP,TN,FP,FN):

ACC = ((TP+TN)\*100)/(TP+TN+FP+FN)

TPR = TP/(TP+FN)

FPR = FP/(FP+TN)

TNR = TN/(TN+FP)

FNR = FN/(FN+TP)

Precision = TP/(TP+FP)

F1 = 2\*((Precision\*TPR)/(Precision+TPR))

print("Accuracy :", ACC)

print("Sensitivity or Recall/TPR :", TPR)

print("Specificity or TNR :", TNR)

print("Fall-Out or FPR :", FPR)

print("Miss rate or False Negative rate :", FNR)

print("Precision :" , Precision)

print("F1 Score: " , F1)

#Logistic regression with stratified samples

logistic\_regression(train['target'],train[Var\_to\_model])

#Predict test data

test['predicted\_prob'] = result.predict(test[Var\_to\_model])

test['PredictedVal'] = 1

test.loc[test.predicted\_prob < 0.5, 'PredictedVal'] = 0

confusion\_matrix(test['target'], test['PredictedVal'])

Metrics(TP,TN,FP,FN)

#ROC Curve

X\_test = test[Var\_to\_model]

log\_ROC\_auc = roc\_auc\_score(test['target'], result.predict(X\_test))

fpr, tpr, threshold = roc\_curve(test['target'], test['predicted\_prob'])

plt.figure()

plt.plot(fpr,tpr, label = "ROC Curve ( area = %0.2f )" %log\_ROC\_auc)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic ')

plt.legend(loc="lower right")

plt.show()

#Decision tree with Stratified samples

decision\_tree\_mod(train[Var\_to\_model], train['target'], test[Var\_to\_model])

confusion\_matrix(test['target'], y\_pred)

Metrics(TP,TN,FP,FN)

#ROC Curve

X\_test = test[Var\_to\_model]

log\_ROC\_auc = roc\_auc\_score(test['target'], clf.predict(X\_test))

fpr, tpr, threshold = roc\_curve(test['target'], y\_pred)

plt.figure()

plt.plot(fpr,tpr, label = "ROC Curve ( area = %0.2f )" %log\_ROC\_auc)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic ')

plt.legend(loc="lower right")

plt.show()

#Random Forest with Stratified samples

random\_forest\_mod(train[Var\_to\_model], train['target'], test[Var\_to\_model])

confusion\_matrix(test['target'], y\_test)

Metrics(TP,TN,FP,FN)

#ROC Curve

X\_test = test[Var\_to\_model]

log\_ROC\_auc = roc\_auc\_score(test['target'], clf.predict(X\_test))

fpr, tpr, threshold = roc\_curve(test['target'], y\_test)

plt.figure()

plt.plot(fpr,tpr, label = "ROC Curve ( area = %0.2f )" %log\_ROC\_auc)

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver operating characteristic ')

plt.legend(loc="lower right")

plt.show()

#Undersampling

# Class count

count\_class\_0, count\_class\_1 = train\_df\_1.target.value\_counts()

# Divide by class

df\_class\_0 = train\_df\_1[train\_df\_1['target'] == 0]

df\_class\_1 = train\_df\_1[train\_df\_1['target'] == 1]

df\_class\_0\_under = df\_class\_0.sample(count\_class\_1, replace = False)

df\_under = pd.concat([df\_class\_0\_under, df\_class\_1], axis=0)

for train\_index, test\_index in ss.split(df\_under, df\_under['target']):

train = df\_under.iloc[train\_index]

test = df\_under.iloc[test\_index]

#OverSampling

# Class count

count\_class\_0, count\_class\_1 = train\_df\_1.target.value\_counts()

# Divide by class

df\_class\_0 = train\_df\_1[train\_df\_1['target'] == 0]

df\_class\_1 = train\_df\_1[train\_df\_1['target'] == 1]

df\_class\_1\_over = df\_class\_1.sample(count\_class\_0//2, replace = True)

df\_over = pd.concat([df\_class\_1\_over, df\_class\_0], axis=0)

for train\_index, test\_index in ss.split(df\_under, df\_under['target']):

train = df\_under.iloc[train\_index]

test = df\_under.iloc[test\_index]

#Logistic regression with stratified over-sampled data

logistic\_regression(train['target'],train[Var\_to\_model])

#Predict test data

test['predicted\_prob'] = result.predict(test[Var\_to\_model])

test['PredictedVal'] = 1

test.loc[test.predicted\_prob < 0.5, 'PredictedVal'] = 0

confusion\_matrix(test['target'], test['PredictedVal'])

Metrics(TP,TN,FP,FN)

#Decision tree with stratified over-sampled data

decision\_tree\_mod(train[Var\_to\_model], train['target'], test[Var\_to\_model])

confusion\_matrix(test['target'], y\_pred)

Metrics(TP,TN,FP,FN)

#Random Forest with stratified over-sampled data

random\_forest\_mod(train[Var\_to\_model], train['target'], test[Var\_to\_model])

confusion\_matrix(test['target'], y\_test)

Metrics(TP,TN,FP,FN)