SANTANDER CUSTOMER TRANSACTION PREDICTION

BY

SAI SAMPATH KUMAR D

Contents

Chapter 1

(THIS DOCUMENT IS PREPARED USING R PROGRAMMING LANGUAGE AND PYTHON)

Introduction

1.1 Background:

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

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

1.2 Problem Statement:

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

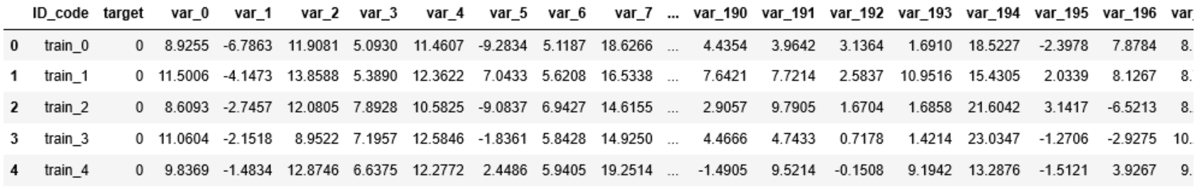
- Supervised: The labels are included in the training data and the goal is to train a model to learn to predict the labels from the features
- Classification: The label is a binary variable, 0 (will not make a specific transaction in the future), 1 (will make a specific transaction in the future)

1.3 Data:

The details of data attributes in the dataset are as follows –

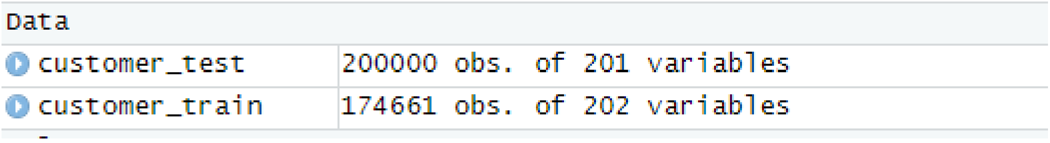
- ID_code (string)
- Target (0 or 1)
- 200 numerical variables, named from var_0 to var_199

Given below is a sample of the data set that we are using to predict customer transactions:



	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...	var_190	var_191	var_192	var_193	var_194	var_195	var_196	var
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...	4.4354	3.9642	3.1364	1.6910	18.5227	-2.3978	7.8784	8.
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...	7.6421	7.7214	2.5837	10.9516	15.4305	2.0339	8.1267	8.
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...	2.9057	9.7905	1.6704	1.6858	21.6042	3.1417	-6.5213	8.
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...	4.4666	4.7433	0.7178	1.4214	23.0347	-1.2706	-2.9275	10.
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...	-1.4905	9.5214	-0.1508	9.1942	13.2876	-1.5121	3.9267	9.

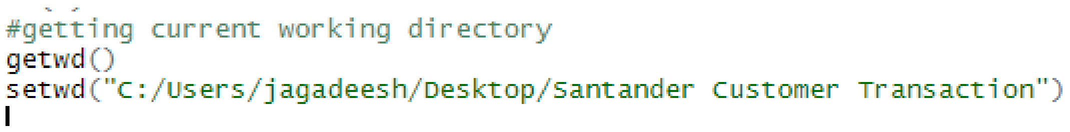
Structure of Data:



Data	
customer_test	200000 obs. of 201 variables
customer_train	174661 obs. of 202 variables

Before starting our analysis part, let us set our working directory in R Studio:

CODE:

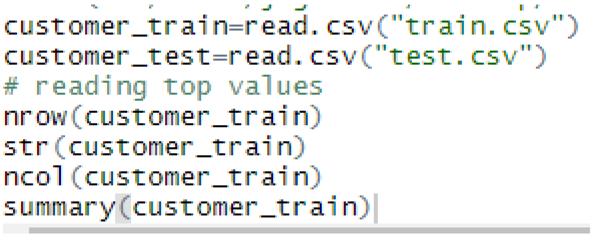


```
#getting current working directory
getwd()
setwd("C:/Users/jagadeesh/Desktop/Santander Customer Transaction")
```

For our analysis part, we have two datasets train and test dataset. Let us start our analysis with train data and validate it with test data. Our train data contains 202 columns and 174661 rows.

Here, I imported the data and stored in the variable **customer_train**.

CODE:



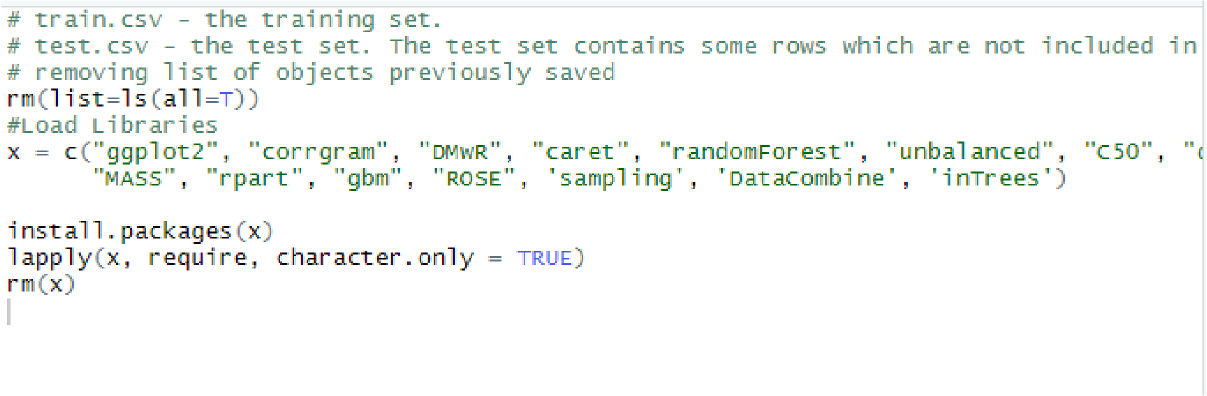
```
customer_train=read.csv("train.csv")
customer_test=read.csv("test.csv")
# reading top values
nrow(customer_train)
str(customer_train)
ncol(customer_train)
summary(customer_train)
```

We have a dataset containing numeric feature variables, the binary target column, and a string ID_code column. The task is to predict the value of target column in the test set.

As a part of this prediction before we start Exploratory Data Analysis, I started loading libraries which are required to start my analysis.

Here REQUIRE function is used to make an R add-on package.

CODE:



```
# train.csv - the training set.
# test.csv - the test set. The test set contains some rows which are not included in
# removing list of objects previously saved
rm(list=ls(all=T))
#Load Libraries
x = c("ggplot2", "corrgram", "DMwR", "caret", "randomForest", "unbalanced", "C50", "
      "MASS", "rpart", "gbm", "ROSE", 'sampling', 'DataCombine', 'inTrees')

install.packages(x)
lapply(x, require, character.only = TRUE)
rm(x)
```

Chapter 2

Methodology

2.1 Pre Processing

We begin by exploring the data, cleaning the data as well as visualizing the data through graphs and plots, which is often called as Exploratory Data Analysis (EDA).

To start this process we will first get a quick glance on the distributions of the variables. Most ML algorithms require the data to be normally distributed. We can visualize that in a glance by using pandas profiling library to generate profile reports in python programing.

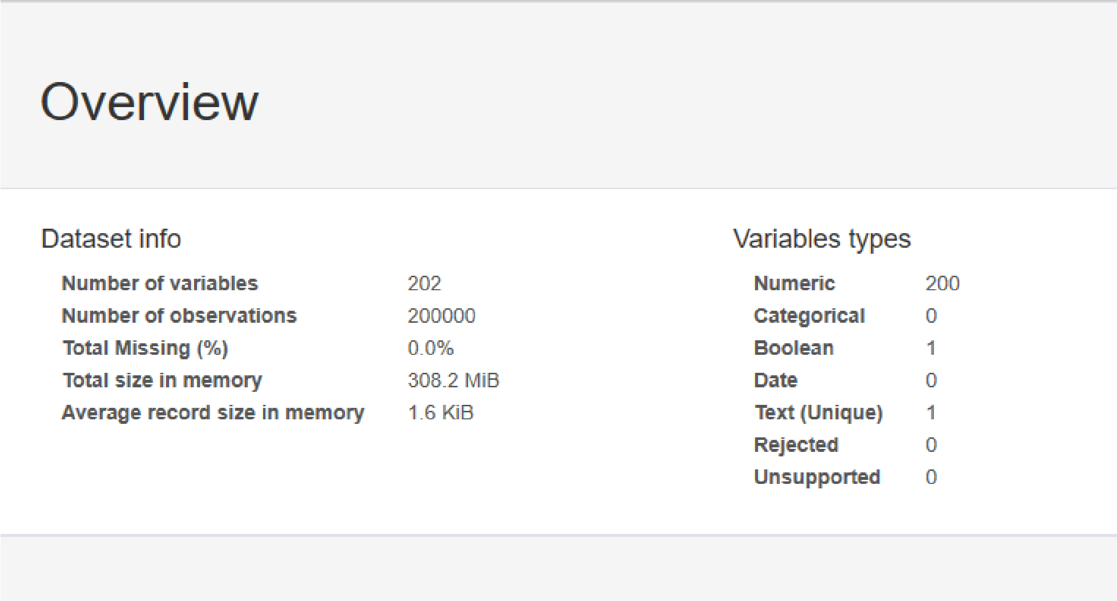
For each column the following statistics are presented in an interactive HTML report:

- Essentials: type, unique values, missing values
- Quantile statistics like minimum value, Q1, median, Q3, maximum, range, interquartile range
- Descriptive statistics like mean, mode, standard deviation, sum, median absolute deviation, coefficient of variation, kurtosis, skewness
- Most frequent values
- Histogram
- Correlations highlighting of highly correlated variables, Spearman and Pearson matrixes

We can access the statistics of each variable generated in the HTML report via our saved repository.

```
profile_report=pp.ProfileReport(customer_train)
```

```
profile_report   # this will give the analysis report in a html file.
```



Overview

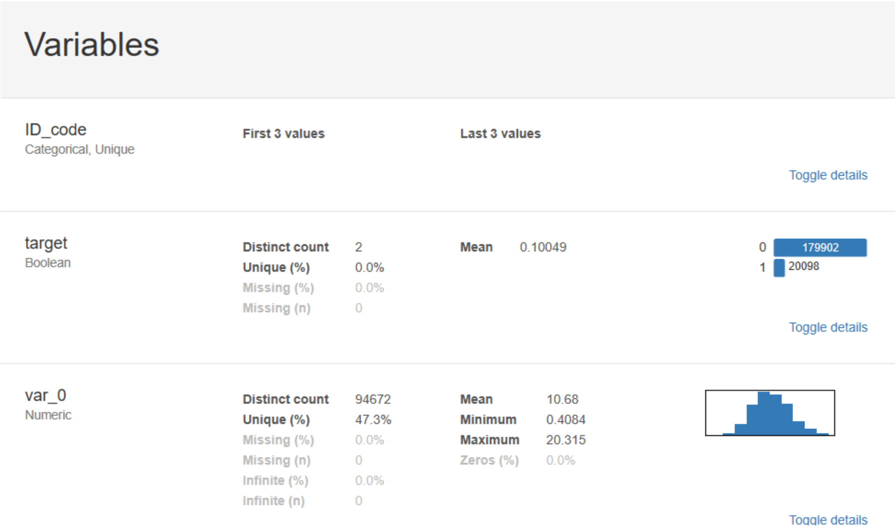
Dataset info

Number of variables	202
Number of observations	200000
Total Missing (%)	0.0%
Total size in memory	308.2 MiB
Average record size in memory	1.6 KiB

Variables types

Numeric	200
Categorical	0
Boolean	1
Date	0
Text (Unique)	1
Rejected	0
Unsupported	0

This report explains each and every variable in detail:



Variables

ID_code
Categorical, Unique

First 3 values

Last 3 values

target
Boolean

Distinct count 2
Unique (%) 0.0%
Missing (%) 0.0%
Missing (n) 0

Mean 0.10049

0	179902
1	20098

var_0
Numeric

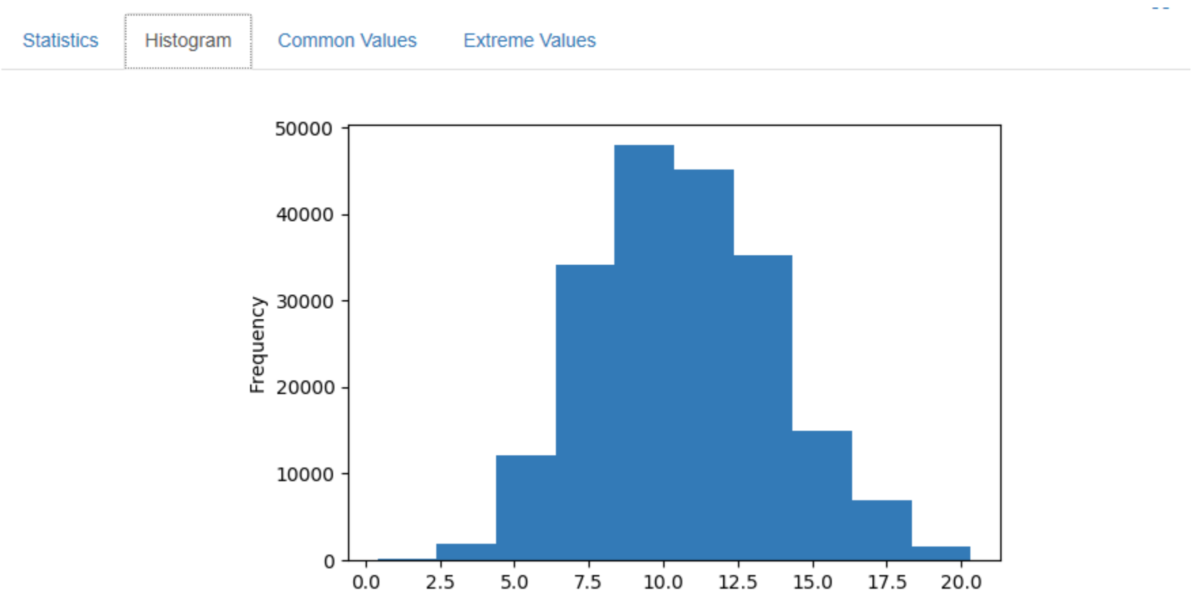
Distinct count 94672
Unique (%) 47.3%
Missing (%) 0.0%
Missing (n) 0
Infinite (%) 0.0%
Infinite (n) 0

Mean 10.68
Minimum 0.4084
Maximum 20.315
Zeros (%) 0.0%

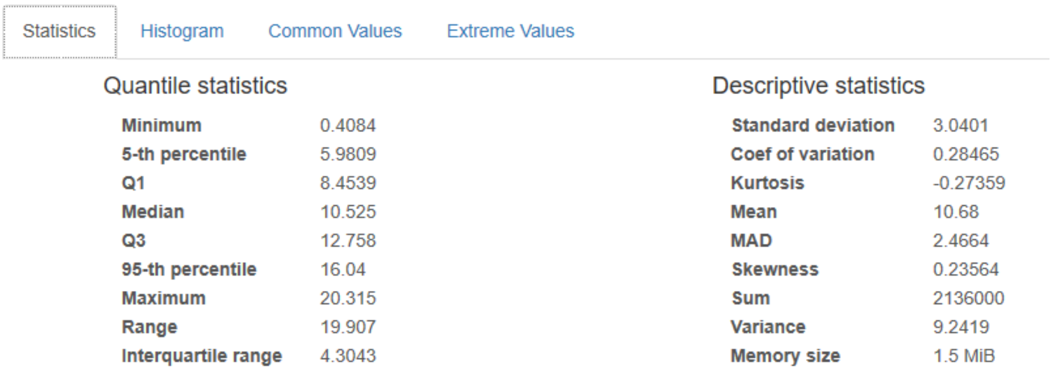
- ID_code variable is categorical and unique. The data type of this variable is string.
- Target Variable is Boolean and contains 0's and 1's. By visualization we can clearly state that about 90% of data is of 0's and remaining 10% is 1's.
- By clearly visualizing the data for ach variable we have 5-point summary like mean, minimum and maximum.

For "var_0 ", Mean is 10.68, minimum is 0.4084 and maximum is at 20.315. We have Statistics, Histogram, Common Values, and Extreme Values for each and every variable. From the histogram, we can visualize if the data is normally distributed. For example, for var_0

HISTOGRAM:



STATISTICS:



Quantile statistics

Minimum	0.4084
5-th percentile	5.9809
Q1	8.4539
Median	10.525
Q3	12.758
95-th percentile	16.04
Maximum	20.315
Range	19.907
Interquartile range	4.3043

Descriptive statistics

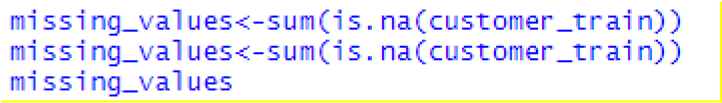
Standard deviation	3.0401
Coef of variation	0.28465
Kurtosis	-0.27359
Mean	10.68
MAD	2.4664
Skewness	0.23564
Sum	2136000
Variance	9.2419
Memory size	1.5 MiB

Similarly, we have statistics for all other variables too.

2.1.1 Missing Value Analysis

Checking for missing values in each and imputing them:

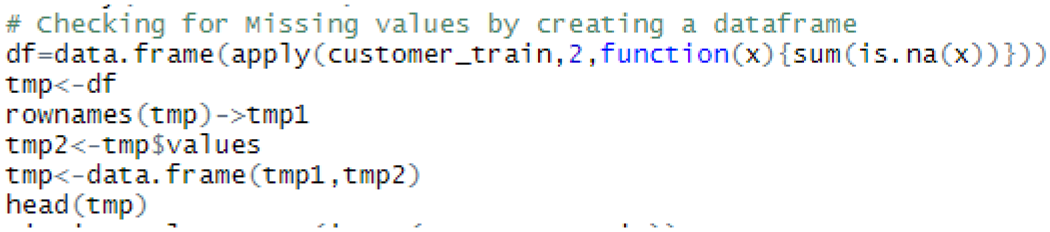
CODE:



```
missing_values<-sum(is.na(customer_train))
missing_values<-sum(is.na(customer_train))
missing_values
```

We created a dataset called tmp and stored values of missing values for each and every column.

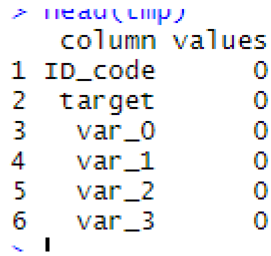
CODE:



```
# Checking for Missing values by creating a dataframe
df=data.frame(apply(customer_train,2,function(x){sum(is.na(x))}))
tmp<-df
rownames(tmp)->tmp1
tmp2<-tmp$values
tmp<-data.frame(tmp1,tmp2)
head(tmp)
```

The structure of "tmp" dataset looks like:

CODE:



```
> head(tmp)
   column values
1 ID_code      0
2  target      0
3   var_0      0
4   var_1      0
5   var_2      0
6   var_3      0
```

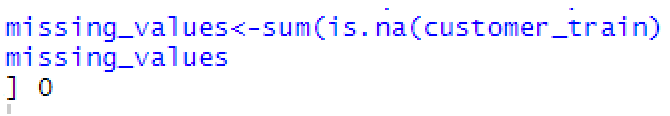
Here, I renamed the column names of tmp with "column" and "values"

Let us calculate percentage of missing values for training dataset:

tmp[order(-tmp$values),]->tmp

Here, we can see that there are no NA values in this datset.

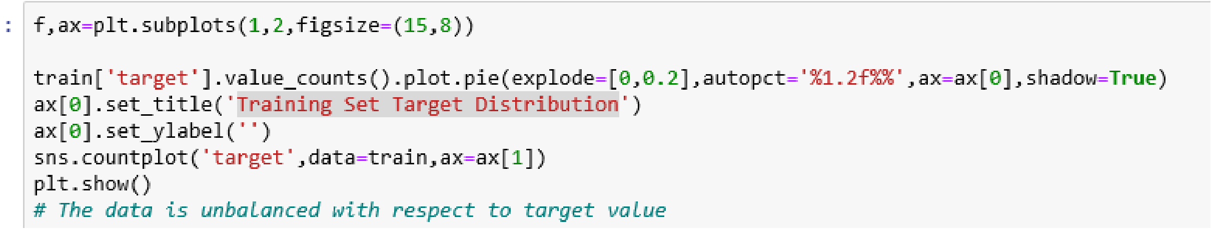
CODE:



```
missing_values<-sum(is.na(customer_train)
missing_values
] 0
```

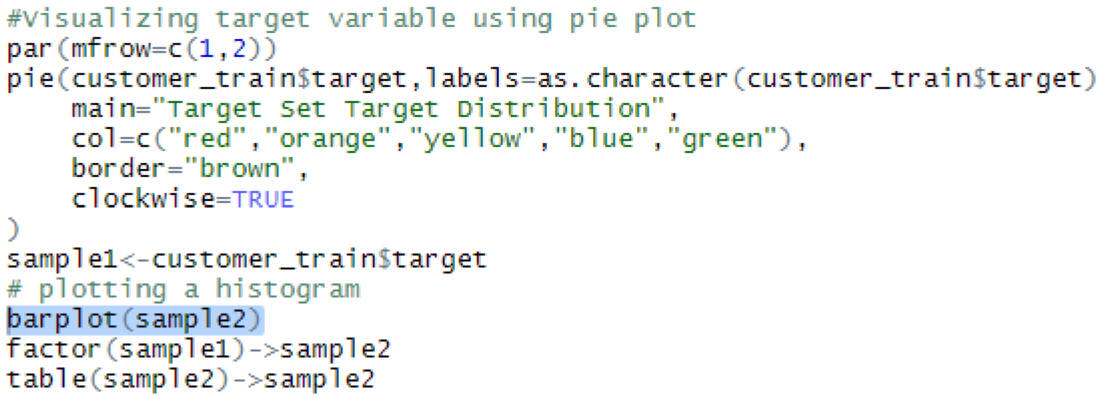
2.1.2 Data Visualization

PYTHON CODE:



```
f,ax=plt.subplots(1,2,figsize=(15,8))

train['target'].value_counts().plot.pie(explode=[0,0.2],autopct='%1.2f%%',ax=ax[0],shadow=True)
ax[0].set_title('Training Set Target Distribution')
ax[0].set_ylabel('')
sns.countplot('target',data=train,ax=ax[1])
plt.show()
# The data is unbalanced with respect to target value
```

R CODE:

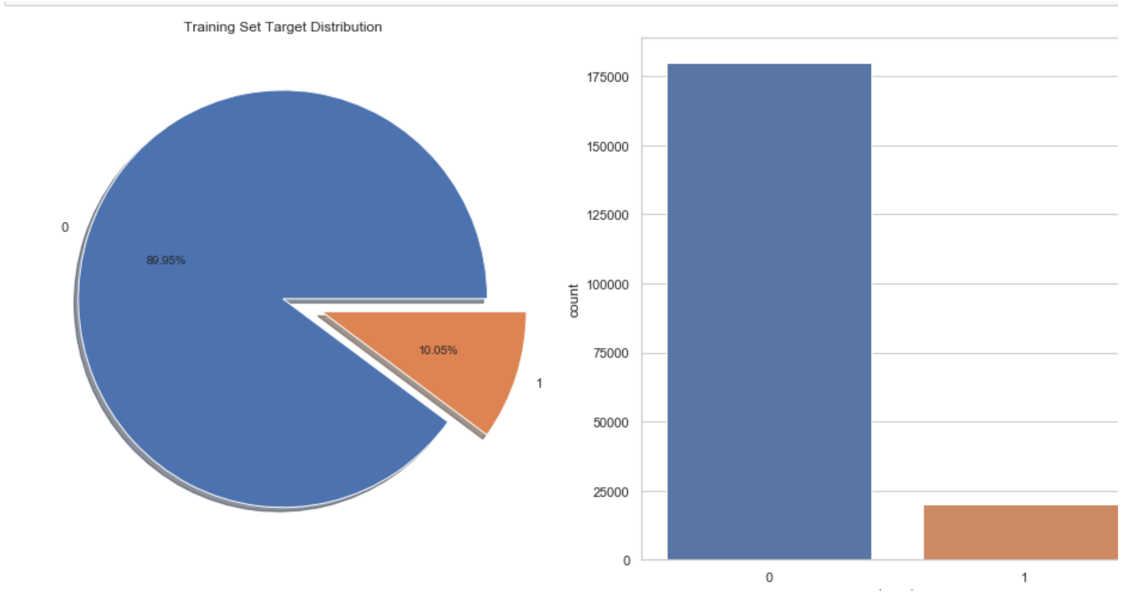


```
#Visualizing target variable using pie plot
par(mfrow=c(1,2))
pie(customer_train$target,labels=as.character(customer_train$target)
    main="Target Set Target Distribution",
    col=c("red","orange","yellow","blue","green"),
    border="brown",
    clockwise=TRUE
)
sample1<-customer_train$target
# plotting a histogram
barplot(sample2)
factor(sample1)->sample2
table(sample2)->sample2
```

From the code, 1st line states that two plots are drawn in 1*2 say 1 row 2 columns and of figure size 15*8 length and breadth each plot.

And in 2nd line for the column target.value counts gives us number of 1's and 0's and we plotted pie diagram. From sns package in line 5, we plotted count plot() which is nothing but bar plot which contains number of 1's and 0's.

PLOT:



From the plot we can see that about 90% of data is 0’s and only 10% of data is 1’s which explains only 10% of transactions done.

- Number of transactions happening account to approx. 10% in the train dataset.
- The dataset is unbalanced with respect to the target, need to consider resampling.

Machine Learning Algorithms are usually designed to improve accuracy by reducing the error. This is a scenario where the number of observations belonging to one class is significantly lower than those belonging to the other classes. In this situation, the predictive model developed using conventional machine learning algorithms could be biased and inaccurate. Here for our dataset, we have 200000 observations of which 90% are of 0’s and 10% are of 1’s.

Now let us visualize remaining variables from var_0 to var_200. From the code below it takes 4 parameters inside function plot_feature_distribution.

Df1-> it contains rows for all variables in training dataset which are 0’s based on target variable

Df2-> it contains rows for all variables in training dataset which are 1’s based on target variable

Label1->0

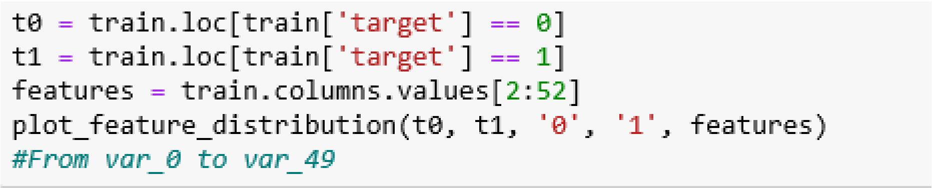
Label2->1

Features-> this is the dataset excluding target variable and train variable.

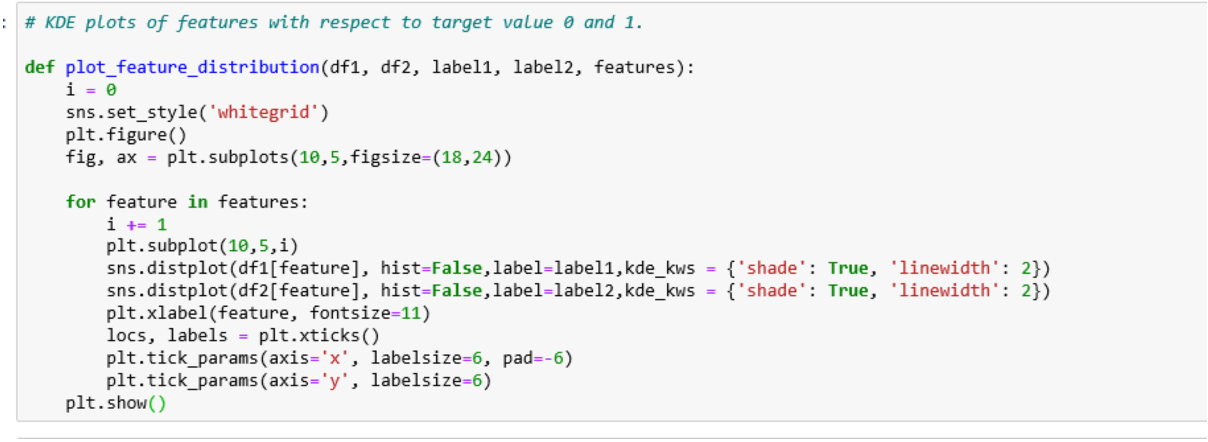
Here t0 is the variable containing rows which are 0’s 90% of data is here

Here t1 is the variable containing rows which are 1’s 10% of data is here

As it is difficult to plot all 200 variables at a single row lets plot them dividing into 4 parts from 2:52, 53:103…..etc.

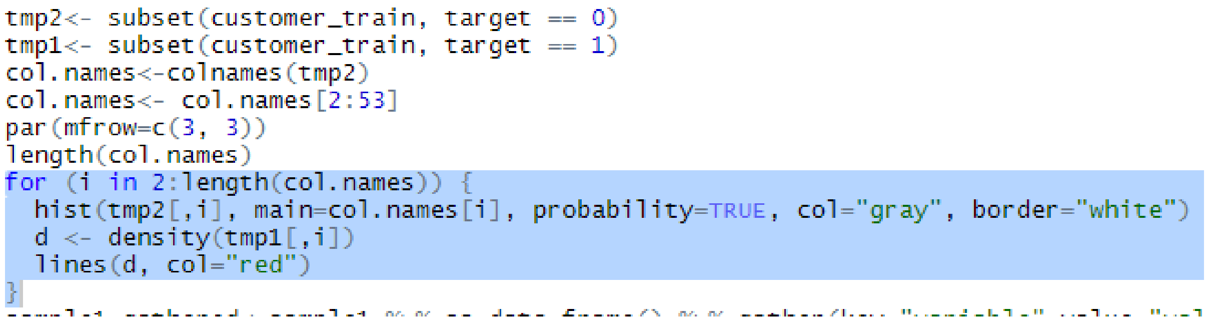


```
t0 = train.loc[train['target'] == 0]
t1 = train.loc[train['target'] == 1]
features = train.columns.values[2:52]
plot_feature_distribution(t0, t1, '0', '1', features)
#From var_0 to var_49
```



```
# KDE plots of features with respect to target value 0 and 1.

def plot_feature_distribution(df1, df2, label1, label2, features):
    i = 0
    sns.set_style('whitegrid')
    plt.figure()
    fig, ax = plt.subplots(10,5,figsize=(18,24))

    for feature in features:
        i += 1
        plt.subplot(10,5,i)
        sns.distplot(df1[feature], hist=False,label=label1,kde_kws = {'shade': True, 'linewidth': 2})
        sns.distplot(df2[feature], hist=False,label=label2,kde_kws = {'shade': True, 'linewidth': 2})
        plt.xlabel(feature, fontsize=11)
        locs, labels = plt.xticks()
        plt.tick_params(axis='x', labelsize=6, pad=-6)
        plt.tick_params(axis='y', labelsize=6)
    plt.show()
```

R CODE:



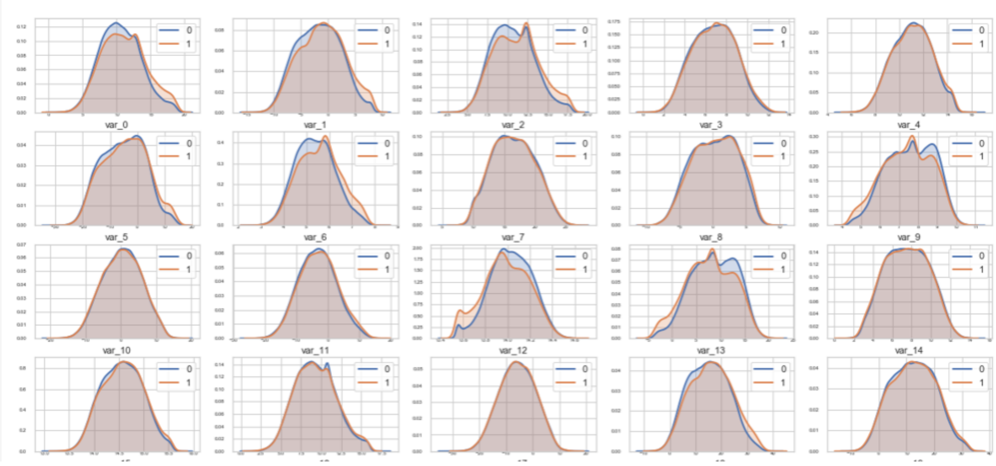
```
tmp2<- subset(customer_train, target == 0)
tmp1<- subset(customer_train, target == 1)
col.names<-colnames(tmp2)
col.names<- col.names[2:53]
par(mfrow=c(3, 3))
length(col.names)
for (i in 2:length(col.names)) {
  hist(tmp2[,i], main=col.names[i], probability=TRUE, col="gray", border="white")
  d <- density(tmp1[,i])
  lines(d, col="red")
}
```

Here variables are taken from 2 to 53

Here we imported seaborn library to plot features. We used matplotlib library to divide different plots and adjusting them in axis. Here (10,5) defines the axis is divided into 10 rows and 5 columns. Each row contains 5 figures and size of each figure is 18*24 size.

Under for loop defined below, placing the figure under axis position with plt.subplot. **sns.distplot()** is the function to plot the vector and we gave hist=false which plots a distribution graph and first is for 0's and 2nd is for 1's. Both overlaps each other.

Here xticks() adjusts default values for x-axis and y-axis.

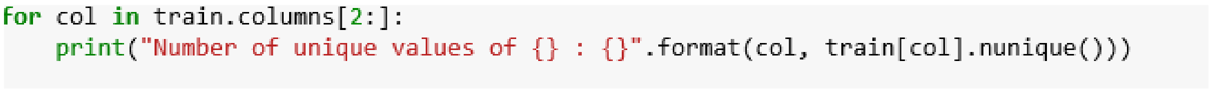


From the graph we can see that almost variable 4, variable 8, variable 19 etc is giving the resemblance of bimodal distribution (both the distributions are of same height).

If pdf(target = 1) - pdf(target = 0) > 0, which indicates customers will make a specific transaction in future.

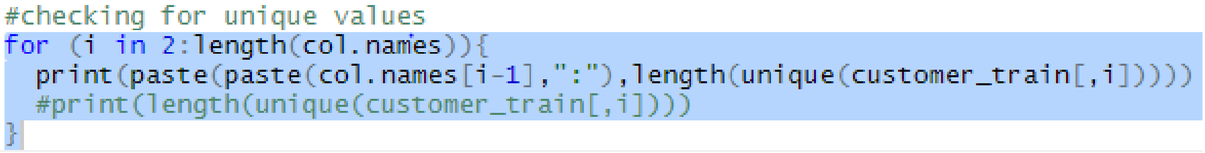
Now let us check number of unique values in each column:

PYTHON CODE:



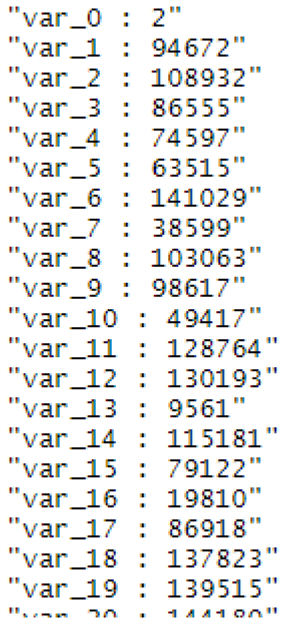
```
for col in train.columns[2:]:
    print("Number of unique values of {} : {}".format(col, train[col].nunique()))
```

R CODE:



```
#checking for unique values
for (i in 2:length(col.names)){
  print(paste(paste(col.names[i-1],":"),length(unique(customer_train[,i]))))
  #print(length(unique(customer_train[,i])))
}
```

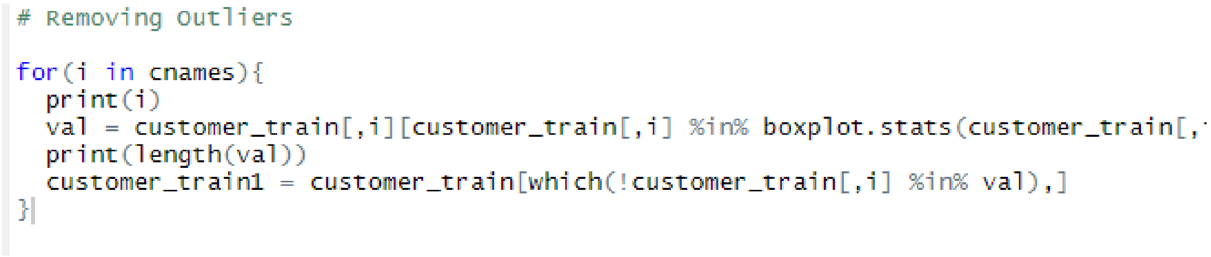
OUTPUT:



```
 "var_0 : 2"
 "var_1 : 94672"
 "var_2 : 108932"
 "var_3 : 86555"
 "var_4 : 74597"
 "var_5 : 63515"
 "var_6 : 141029"
 "var_7 : 38599"
 "var_8 : 103063"
 "var_9 : 98617"
 "var_10 : 49417"
 "var_11 : 128764"
 "var_12 : 130193"
 "var_13 : 9561"
 "var_14 : 115181"
 "var_15 : 79122"
 "var_16 : 19810"
 "var_17 : 86918"
 "var_18 : 137823"
 "var_19 : 139515"
 "var_20 : 144180"
```

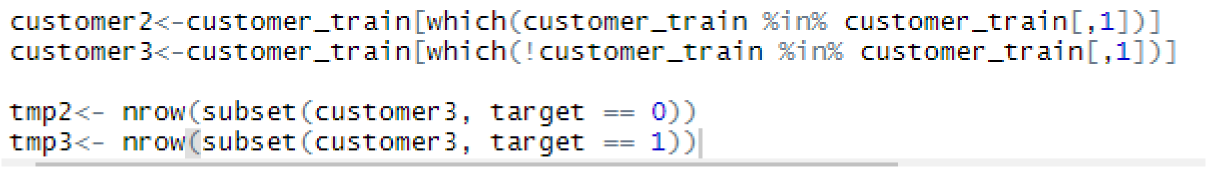
2.1.3 OUTLIER ANALYSIS:

R CODE to Remove Outliers:

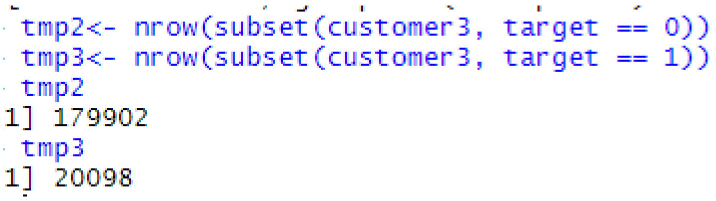


```
# Removing outliers

for(i in cnames){
  print(i)
  val = customer_train[,i][customer_train[,i] %in% boxplot.stats(customer_train[,
  print(length(val))
  customer_train1 = customer_train[which(!customer_train[,i] %in% val),]
}
```

Let us see number of target variables in the new dataset after removing outliers:



```
customer2<-customer_train[which(customer_train %in% customer_train[,1])]
customer3<-customer_train[which(!customer_train %in% customer_train[,1])]

tmp2<- nrow(subset(customer3, target == 0))
tmp3<- nrow(subset(customer3, target == 1))
```



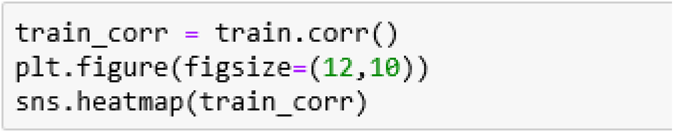
```
 tmp2<- nrow(subset(customer3, target == 0))
 tmp3<- nrow(subset(customer3, target == 1))
 tmp2
1] 179902
 tmp3
1] 20098
```

From the dataset, we found that all the target variables with target variable “1” are outliers. Here, Outliers present in our data, are meaningful and thus can’t be removed.

Let’s calculate correlation with target variable:

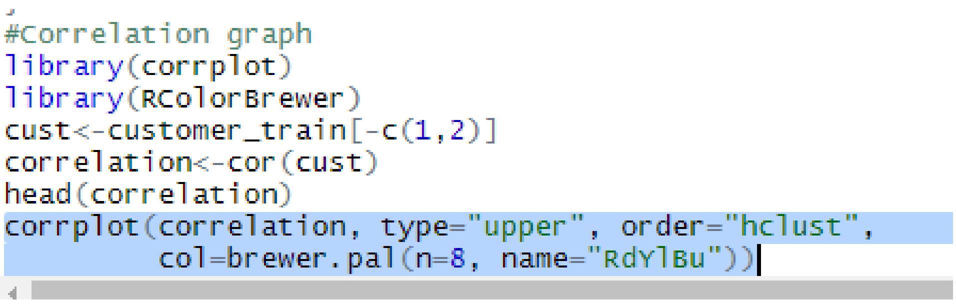
CORRELATION GRAPH:

PYTHON CODE:



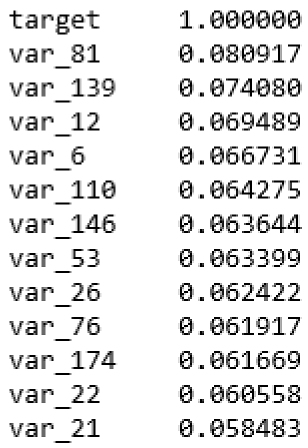
```
train_corr = train.corr()
plt.figure(figsize=(12,10))
sns.heatmap(train_corr)
```

R CODE:



```
#Correlation graph
library(corrplot)
library(RColorBrewer)
cust<-customer_train[-c(1,2)]
correlation<-cor(cust)
head(correlation)
corrplot(correlation, type="upper", order="hclust",
         col=brewer.pal(n=8, name="RdYlBu"))
```

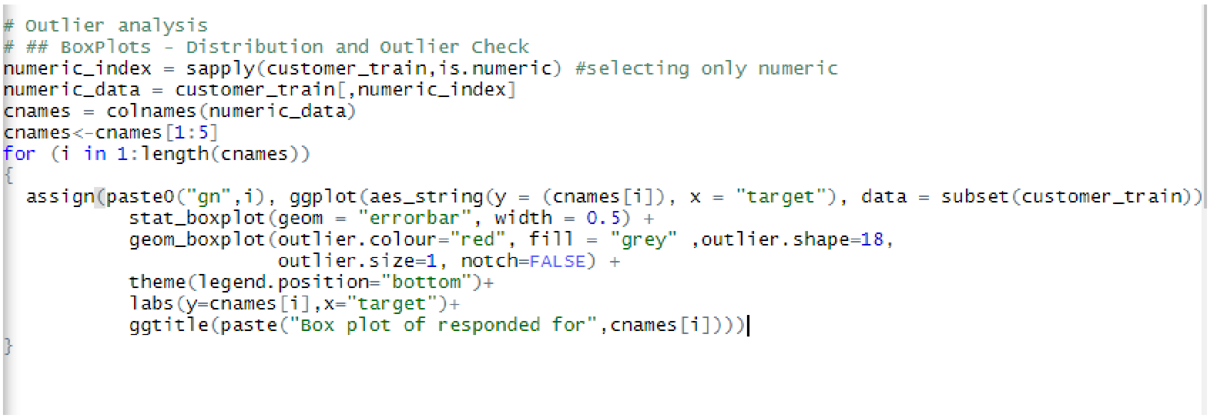
CORRELATION BETWEEN TWO VALUES:



```
target     1.000000
var_81     0.080917
var_139    0.074080
var_12     0.069489
var_6      0.066731
var_110    0.064275
var_146    0.063644
var_53     0.063399
var_26     0.062422
var_76     0.061917
var_174    0.061669
var_22     0.060558
var_21     0.058483
```

In addition, if two predictors are strongly correlated to each other, then we only need to use one of them and 0 means that there is no relationship between the two variables. Here, correlation values with target variable is very low which indicates there is relationship is very poor.

BOX PLOT METHOD:

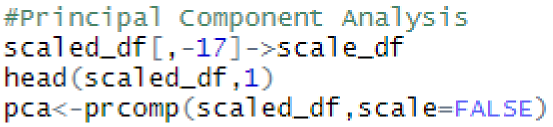


```
# Outlier analysis
# ## BoxPlots - Distribution and Outlier Check
numeric_index = sapply(customer_train,is.numeric) #selecting only numeric
numeric_data = customer_train[,numeric_index]
cnames = colnames(numeric_data)
cnames<-cnames[1:5]
for (i in 1:length(cnames))
{
  assign(paste0("gn",i), ggplot(aes_string(y = (cnames[i]), x = "target"), data = subset(customer_train))
           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 responded for",cnames[i])))
}
```

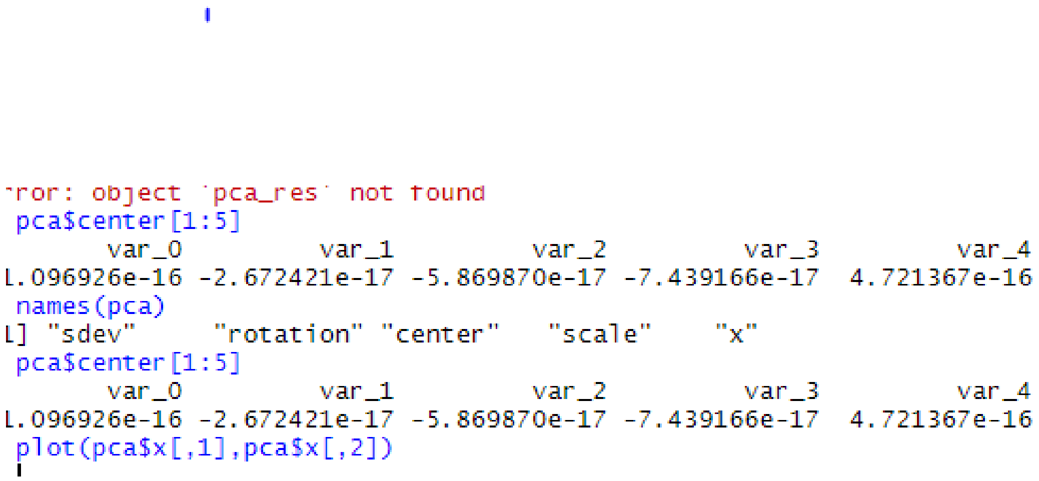
2.1.4 Principal component analysis (PCA):

PCA is a dimensionality reduction technique that reduces less-informative 'noise' features.

But PCA is sensitive to variance and different scales, so standardizing will help PCA perform better. Here. We can see that the correlation between different features in the training dataset is not that significant, so using PCA might not be meaningful and it cannot help. PCA in R can be done using

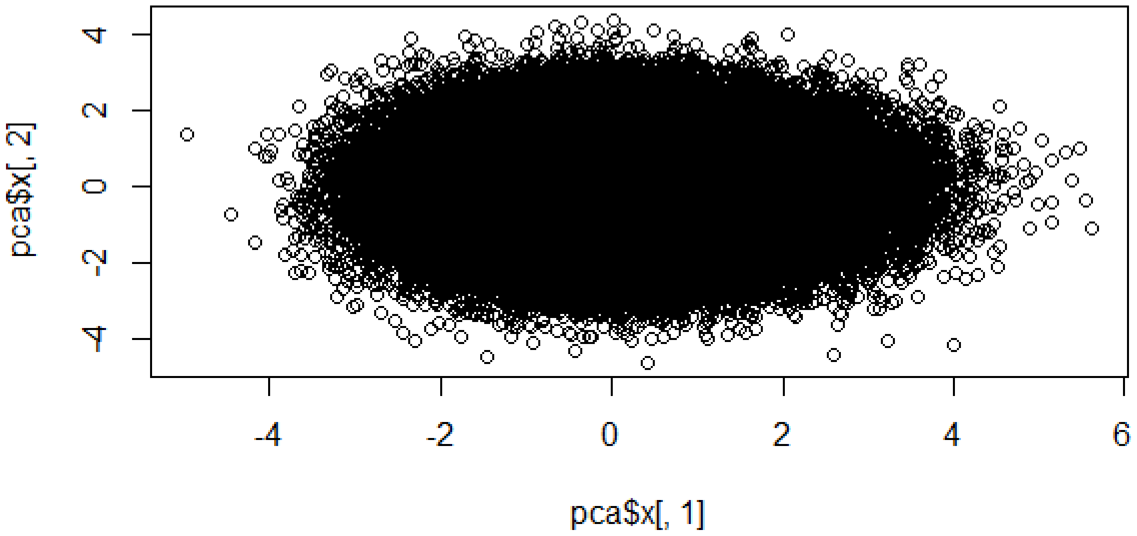


```
#Principal Component Analysis
scaled_df[,-17]->scale_df
head(scaled_df,1)
pca<-prcomp(scaled_df,scale=FALSE)
```

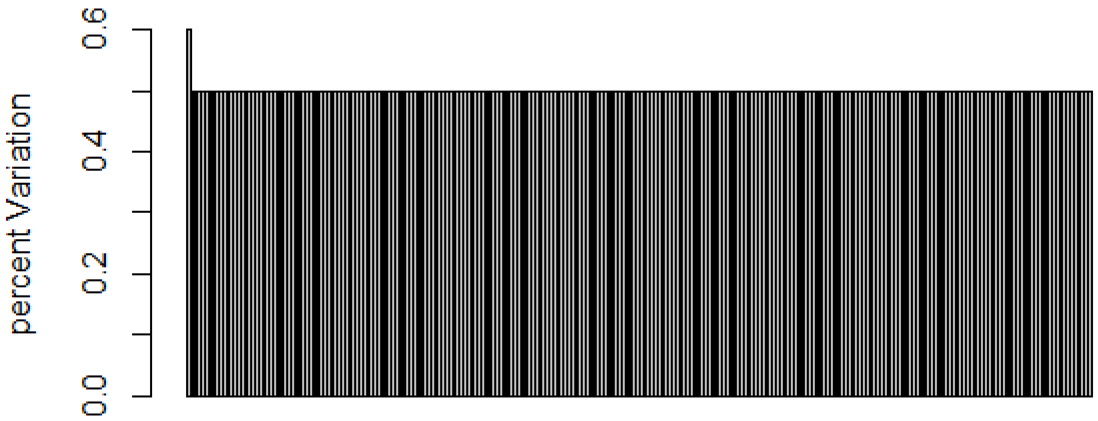


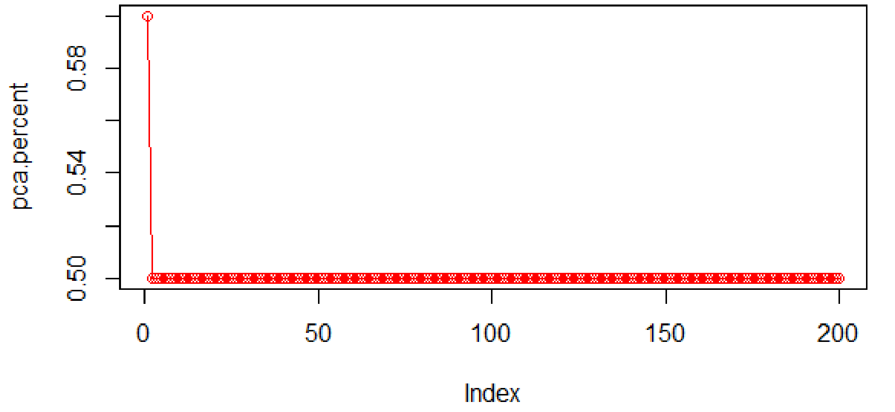
```
ror: object ‘pca_res’ not found
 pca$center[1:5]
        var_0         var_1         var_2         var_3          var_4
L.096926e-16 -2.672421e-17 -5.869870e-17 -7.439166e-17  4.721367e-16
 names(pca)
L] "sdev"     "rotation" "center"   "scale"    "x"
 pca$center[1:5]
        var_0         var_1         var_2         var_3          var_4
L.096926e-16 -2.672421e-17 -5.869870e-17 -7.439166e-17  4.721367e-16
 plot(pca$x[,1],pca$x[,2])
```

PLOT:



Let us check the percent variation for all principal components

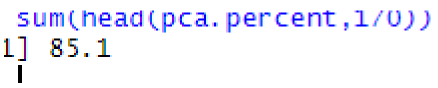




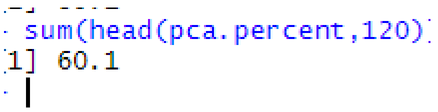
Normally, if there is a elbow looking point in the graph above, the x value (number of features) of that point is usually the ideal number of components for PCA.

However in this case, each principal component explains very little of the total variance (e.g. first principal component only explains about 0.6% of the total variance).

Even when we sum up all the variance explained by the 80 principal components, it only amounts to 40%. Let's increase the k and see what happens



```
 sum(head(pca.percent,170))
1] 85.1
```



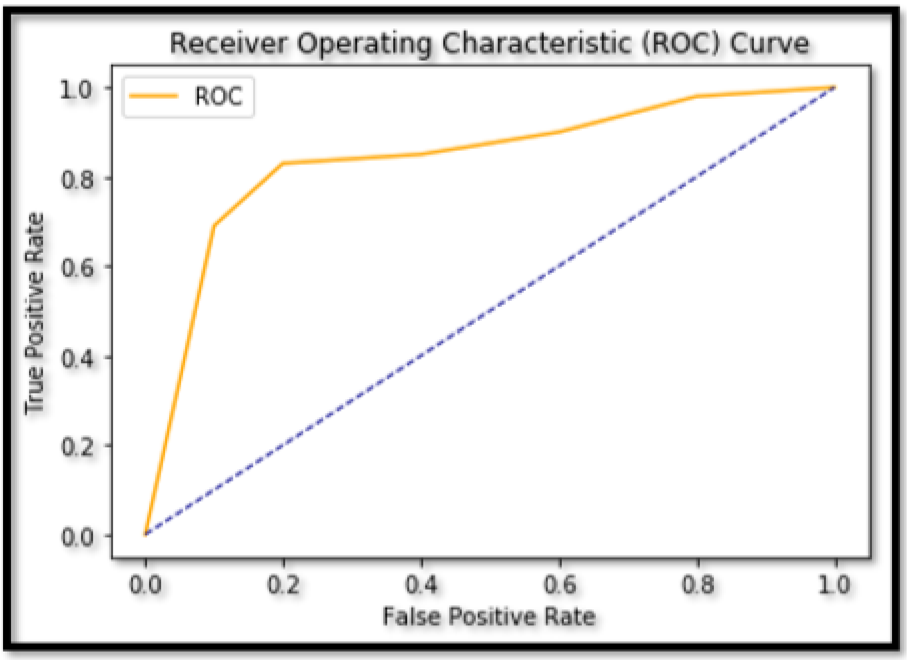
```
 sum(head(pca.percent,120)
[1] 60.1
```

Even with 170 principal components, 85% variance is explained. PCA is best when dimension is large and a lot of features are correlated to one another.

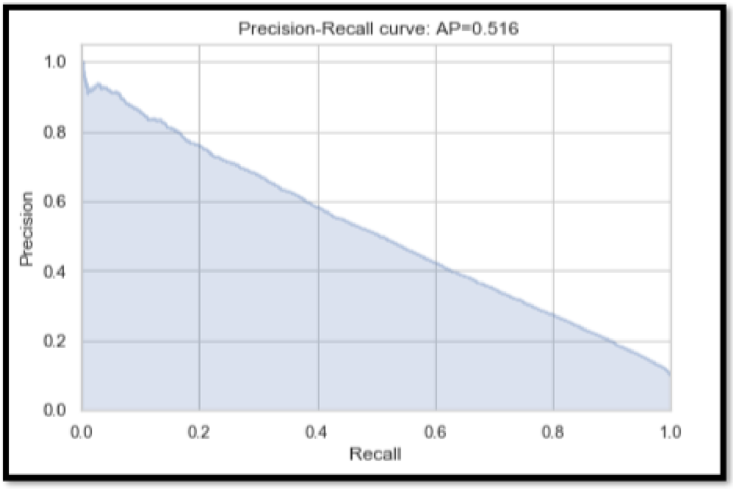
CHAPTER-3:

3.1 Evaluation Metric

- It can be more flexible to predict probabilities of an observation belonging to each class in a classification problem rather than predicting classes directly.
- The reason for this is to provide our model the capability to choose and even calibrate the threshold for how to interpret the predicted probabilities.
- There are two diagnostic tools that help in the interpretation of probabilistic forecast for binary (two-class) classification predictive modelling problems are ROC Curves and Precision-Recall curves.
- ROC is a probability curve for different classes. ROC tells us how good the model is for distinguishing the given classes, in terms of the predicted probability.
- A typical ROC curve has False Positive Rate (FPR) on the X-axis and True Positive Rate (TPR) on the Y-axis.



- The area covered by the curve is the area between the orange line (ROC) and the axis. This area covered is AUC. The bigger the area covered, the better the machine learning models is at distinguishing the given classes. Ideal value for AUC is 1.
- Precision is a ratio of the number of true positives divided by the sum of the true positives and false positives. It describes how good a model is at predicting the positive class.
- Recall is calculated as the ratio of the number of true positives divided by the sum of the true positives and the false negatives. Recall is the same as sensitivity.
- Precision-Recall curves are useful in cases where there is an imbalance in the observations between the two classes. Specifically, there are many examples of no event (class 0) and only a few examples of an event (class 1).
- Key to the calculation of precision and recall is that the calculations do not make use of the true negatives. It is only concerned with the correct prediction of the minority class, class 1.
- A precision-recall curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds, much like the ROC curve.

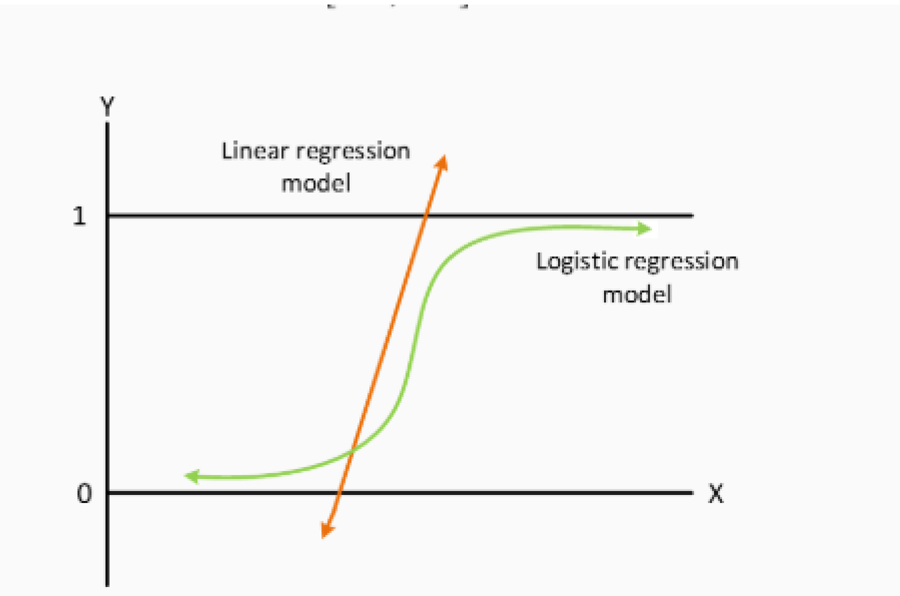


- Hence, ROC curves are appropriate when the observations are balanced between each class, whereas precision-recall curves are appropriate for imbalanced datasets.

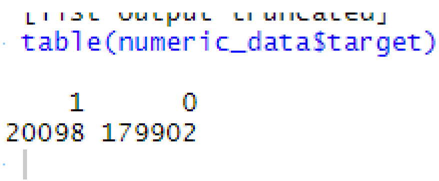
LOGISTIC REGRESSION:

In situations where the nature of dependent (target) variable is categorical, the logistic regression can be used to model the relationship**.**

The nature of logistic regression can be binomial, ordinal or multinomial. In order to predict dependent (target) variable which can take only two values (yes/no , 0/1) Binomial (or binary) logistic regression model is used. Multinomial logistic regression is used when the dependent variable can take three of more categorical values which are not ordered. Whereas the ordinal logistic regression is used when the dependent variable can take categorical values that are ordered.

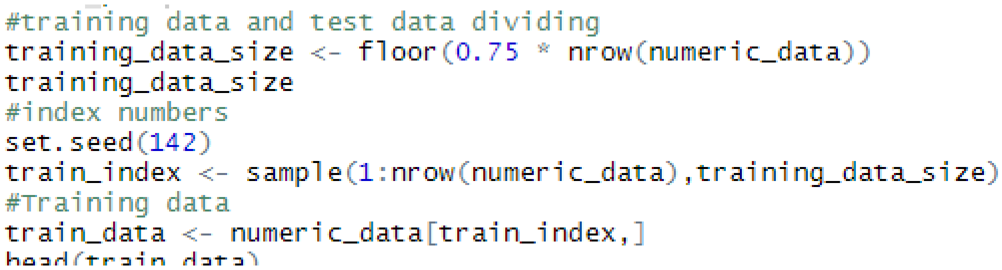


In order to perform Logistic regression analysis on our dataset, we removed the first and unused variable for our analysis ID, and let us convert target variable to “factor”. We also have to solve target class imbalance problem. Since the response variable is a binary categorical variable, you need to make sure the training data has approximately equal proportion of classes.



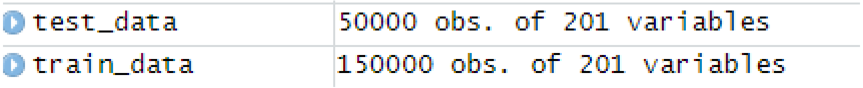
```
[list output truncated]
table(numeric_data$target)

    1      0
20098 179902
```

DIVIDING DATASET INTO TRAINING AND TEST DATASET:



```
#training data and test data dividing
training_data_size <- floor(0.75 * nrow(numeric_data))
training_data_size
#index numbers
set.seed(142)
train_index <- sample(1:nrow(numeric_data),training_data_size)
#Training data
train_data <- numeric_data[train_index,]
head(train_data)
```

Here, the dataset is split in the ratio by 75:25 training to test data



```
test_data     50000 obs. of 201 variables
train_data    150000 obs. of 201 variables
```

The target variable is split in the ratio approximately 1:6

Clearly there is a class imbalance. So, before building the logit model, you need to build the samples such that both the 1's and 0's are in approximately equal proportions.

This concern is normally handled with a couple of techniques called:

- Down Sampling
- Up Sampling
- Hybrid Sampling using SMOTE and ROSE.

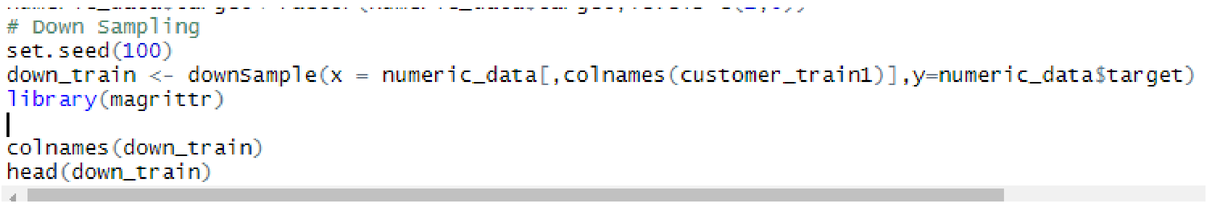
In Down sampling, the majority class is randomly down sampled to be of the same size as the smaller class. That means, when creating the training dataset, the rows with the 0's Class will be picked fewer times during the random sampling.

Similarly, in Up Sampling, rows from the minority class, that is, 1's is repeatedly sampled over and over till it reaches the same size as the majority class (0's).

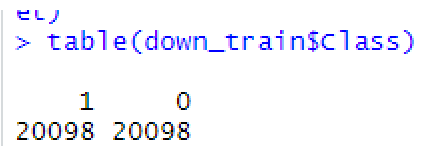
But in case of Hybrid sampling, artificial data points are generated and are systematically added around the minority class. This can be implemented using the SMOTE and ROSE packages

Here in our case let's use down sampling technique as up sampling makes the size of dataset to 400000.

I used the following code to down sample the dataset.



```
# Down Sampling
set.seed(100)
down_train <- downSample(x = numeric_data[,colnames(customer_train1)],y=numeric_data$target)
library(magrittr)

colnames(down_train)
head(down_train)
```

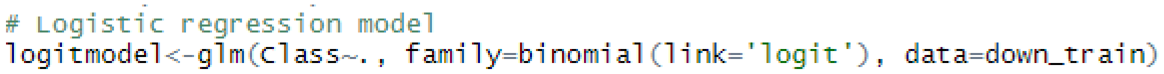


```
> table(down_train$Class)

    1     0 
20098 20098 
```

Now the target class is in the ratio 1:1

Building Logistic Regression model:

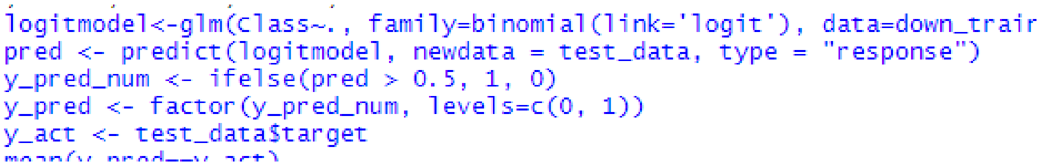
CODE:



```
# Logistic regression model
logitmodel<-glm(Class~., family=binomial(link='logit'), data=down_train)
```

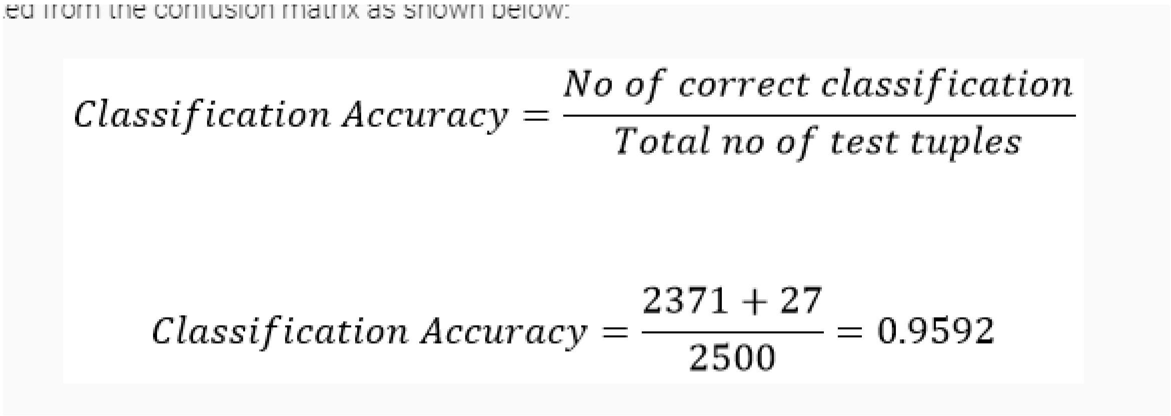
Note that, when you use logistic regression, you need to set type='response' in order to compute the prediction probabilities. This argument is not needed in case of linear regression.

The common practice is to take the probability cut off as 0.5. If the probability of Y is > 0.5, then it can be classified an event 1 else 0. So, if pred is greater than 0.5, it is said that transaction happens else 0 which mean that there is no transaction.



```
logitmodel<-glm(Class~., family=binomial(link='logit'), data=down_trair
pred <- predict(logitmodel, newdata = test_data, type = "response")
y_pred_num <- ifelse(pred > 0.5, 1, 0)
y_pred <- factor(y_pred_num, levels=c(0, 1))
y_act <- test_data$target
mean(y_pred==y_act)
```

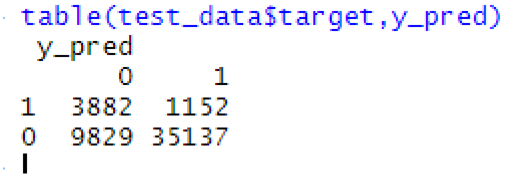
Now let us calculate accuracy of the model from confusion Matrix obtained:



ed from the confusion matrix as shown below:

$$Classification\ Accuracy = \frac{No\ of\ correct\ classification}{Total\ no\ of\ test\ tuples}$$

$$Classification\ Accuracy = \frac{2371 + 27}{2500} = 0.9592$$



```
 table(test_data$target,y_pred)
  y_pred
        0     1
 1   3882  1152
 0   9829 35137
```

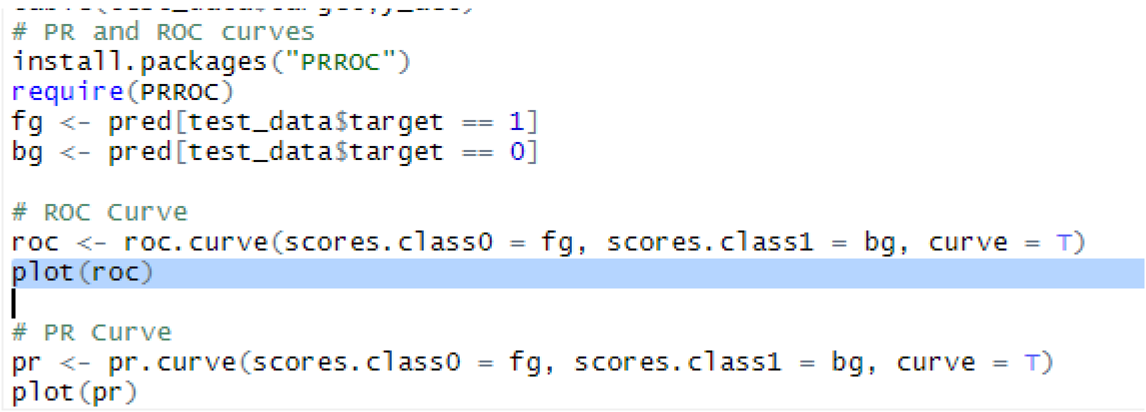
Accuracy = 3882+35137 / 50000

=0.78038

= 78.038 %

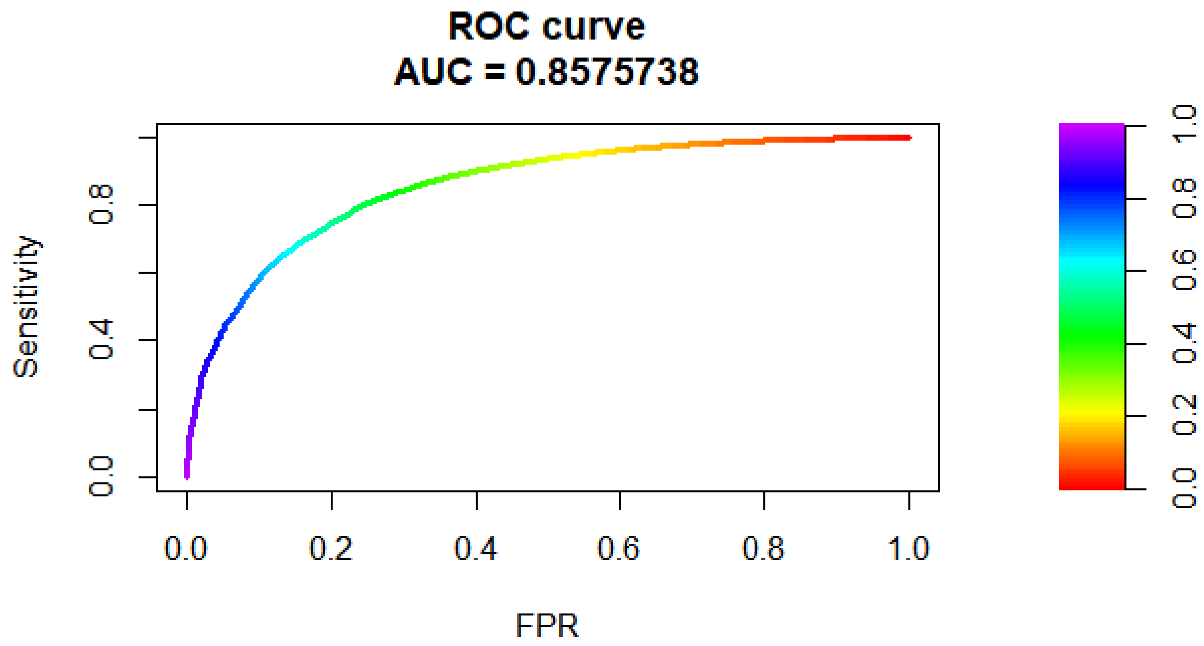
Accuracy of the model is 78.038%.

PR and ROC curves:

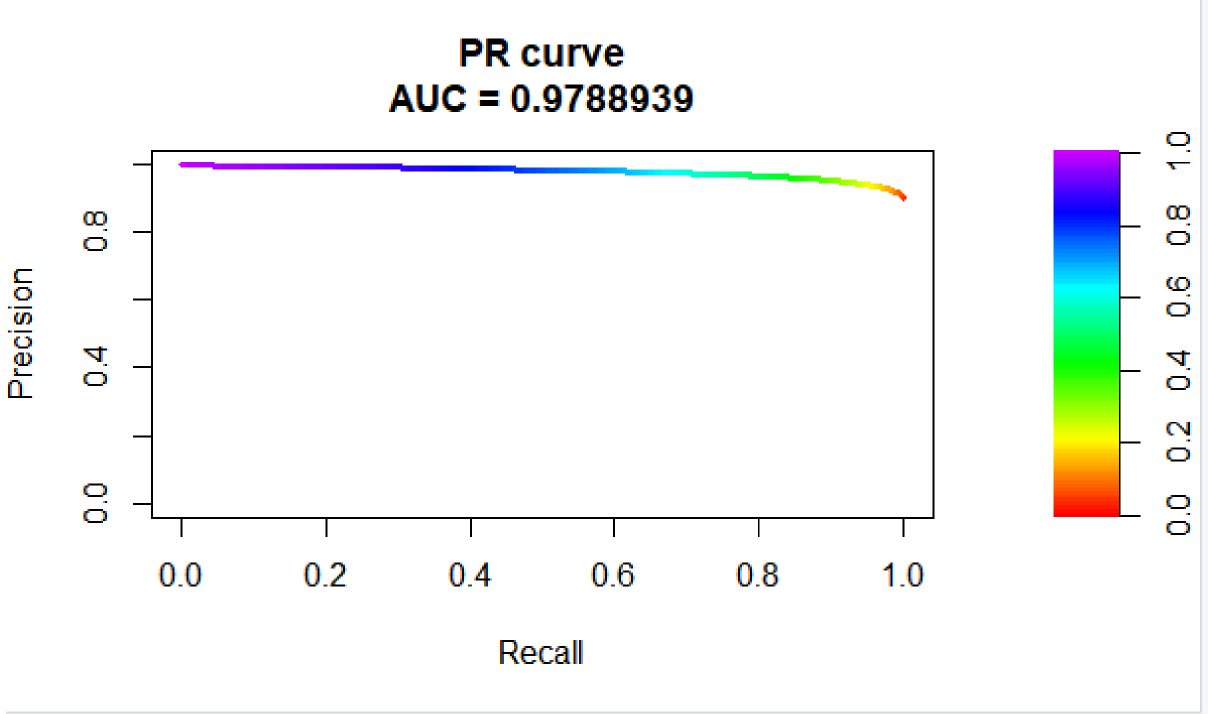


```
# PR and ROC curves
install.packages("PRROC")
require(PRROC)
fg <- pred[test_data$target == 1]
bg <- pred[test_data$target == 0]

# ROC Curve
roc <- roc.curve(scores.class0 = fg, scores.class1 = bg, curve = T)
plot(roc)

# PR Curve
pr <- pr.curve(scores.class0 = fg, scores.class1 = bg, curve = T)
plot(pr)
```

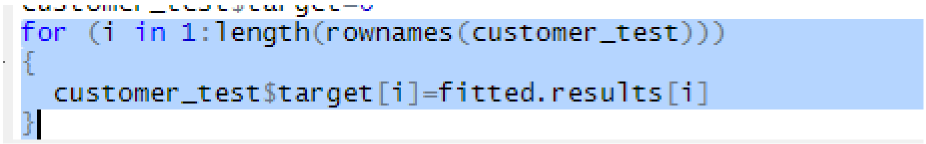
ROC curve:



PR CURVE:



Finding target variable for data Customer_test:



```
customer_test$target=0
for (i in 1:length(rownames(customer_test)))
{
  customer_test$target[i]=fitted.results[i]
}
```

KNN ALGORITHM:

kNN is based on finding the nearest neighbours of a new data. We find the nearest neighbors with the help of Euclidean distance - a commonly used distance metric.

The Euclidean distance between two (tuples) X1 = (x11,x12,x13...x1n) and X2 = (x21,x22,x23...x2n) can be computed as

$$Euclidean\_dist(X_1, X_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

where x11,x12,x13...x1n are numeric attributes of X1 and x21,x22,x23...x2n are the numeric attributes of X2.

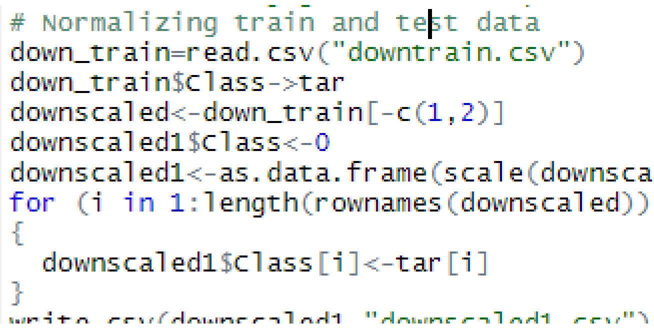
Note: Euclidean distance is used as a distance metric when the data tuple comprises of numeric attributes. Distance metrics such as Hamming distance can be used when the data tuple comprises of categorical attributes.

If KNN algorithm is used without normalizing the data, then the attributes with high range values may have a higher influence on the Euclidean distance as illustrated below:

$$v' = \frac{v - min_A}{max_A - min_A}$$

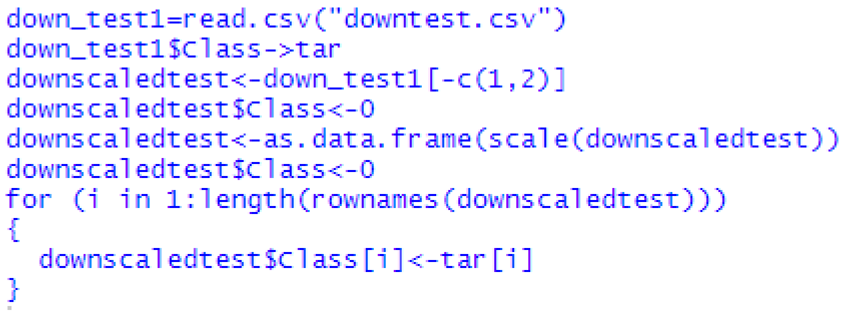
Normalizing the data:

R CODE:



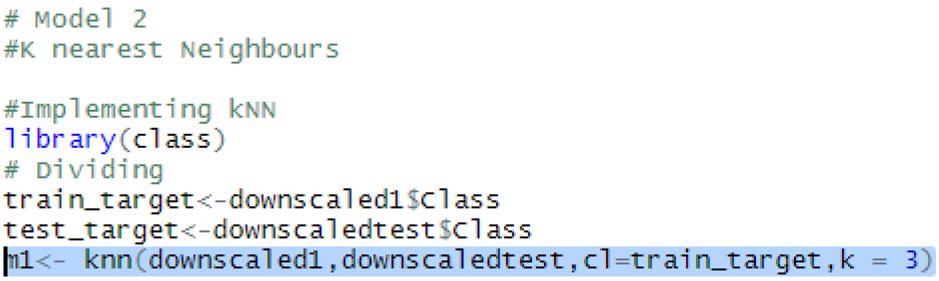
```
# Normalizing train and test data
down_train=read.csv("downtrain.csv")
down_train$Class->tar
downscaled<-down_train[-c(1,2)]
downscaled1$Class<-0
downscaled1<-as.data.frame(scale(downscaled))
for (i in 1:length(rownames(downscaled)))
{
  downscaled1$Class[i]<-tar[i]
}
write.csv(downscaled1,"downscaled1.csv")
```

R Code for test data:



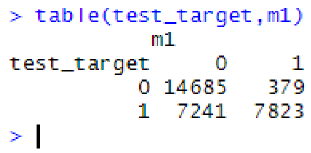
```
down_test1=read.csv("downtest.csv")
down_test1$Class->tar
downscaledtest<-down_test1[-c(1,2)]
downscaledtest$Class<-0
downscaledtest<-as.data.frame(scale(downscaledtest))
downscaledtest$Class<-0
for (i in 1:length(rownames(downscaledtest)))
{
  downscaledtest$Class[i]<-tar[i]
}
```

Now Let us use knn() function from class library to predict the outcomes :



```
# Model 2
#K nearest Neighbours

#Implementing kNN
library(class)
# Dividing
train_target<-downscaled1$Class
test_target<-downscaledtest$Class
m1<- knn(downscaled1,downscaledtest,cl=train_target,k = 3)
```

Accuracy:



```
> table(test_target,m1)
           m1
test_target     0     1
          0 14685   379
          1  7241  7823
> 
```

Accuracy = (14685 +7823)/ Total

=74.70%

This accuracy can be increased by increasing K value.

This prediction accuracy increases with increase in K Value like k=3,4,5…so on

Now let us predict best k value using Cross Validation:

In the data set which has been used so far for our analysis, let us consider 75% of the data as training data and rest 25% data as validation data. In K-fold cross validation the training set is split into ‘K’ smaller sets, each sets is used for training and validation. The following procedure is followed for each of K “folds”:

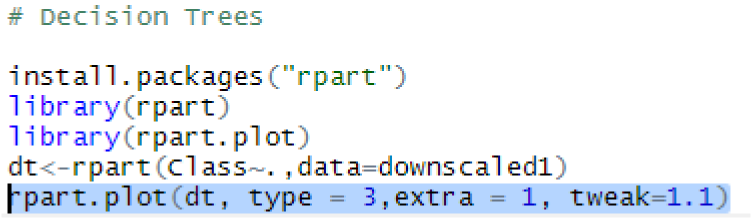
Randomly partition the observations into K groups of equal length

A classifier is trained using K-1 of the folds as training data, the left out group is used as a validation set to compute a performance measure such as accuracy.

But this algorithm is comparatively slow as it finds Euclidean distance between vectors. We can also find optimal K value by using traincontrol() from Caret library.

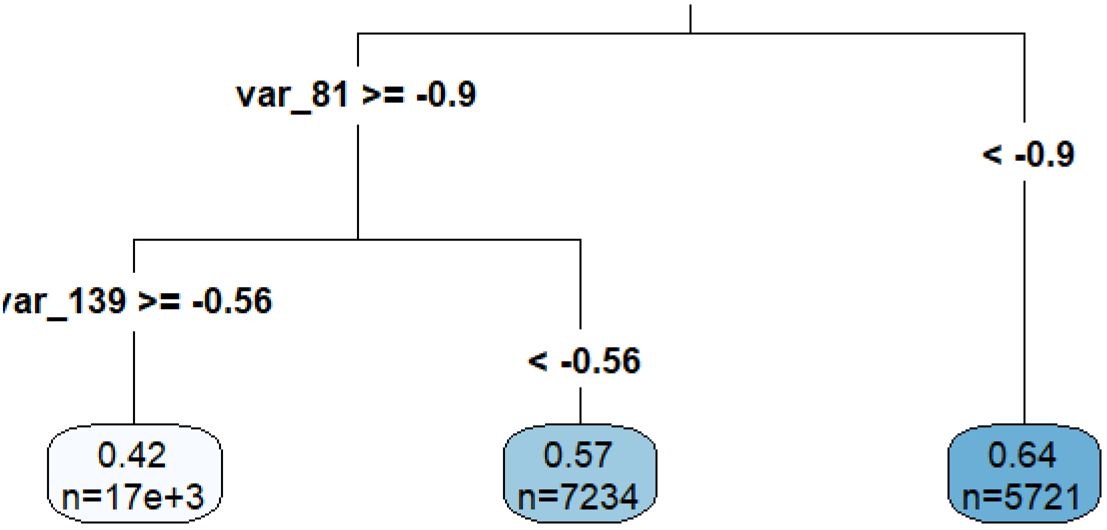
3.4 DECISION TREES:

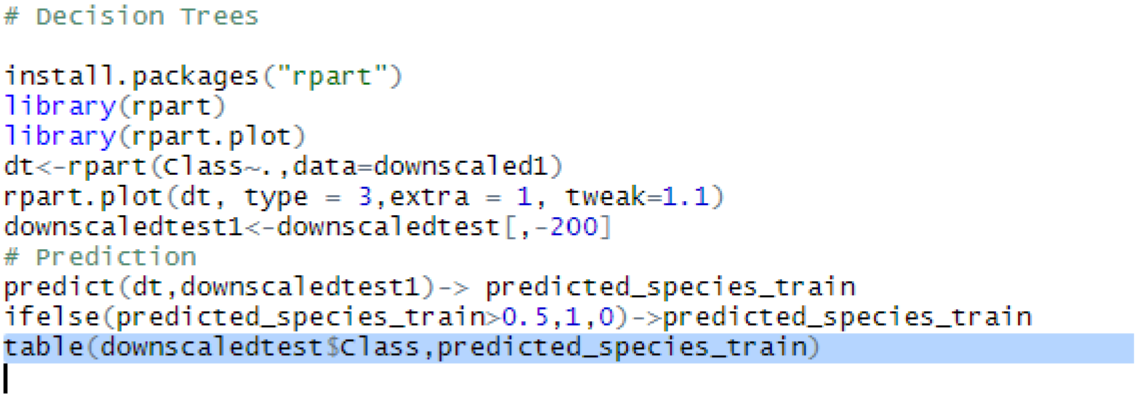
R CODE:



```
# Decision Trees

install.packages("rpart")
library(rpart)
library(rpart.plot)
dt<-rpart(Class~.,data=downscaled1)
rpart.plot(dt, type = 3,extra = 1, tweak=1.1)
```

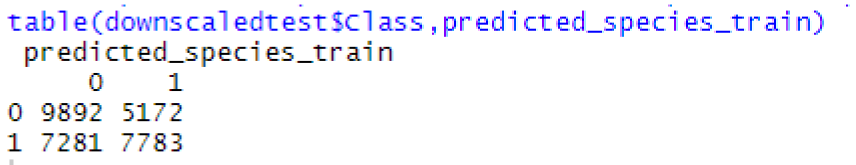
PLOT:





```
# Decision Trees

install.packages("rpart")
library(rpart)
library(rpart.plot)
dt<-rpart(Class~.,data=downscaled1)
rpart.plot(dt, type = 3,extra = 1, tweak=1.1)
downscaledtest1<-downscaledtest[,-200]
# Prediction
predict(dt,downscaledtest1)-> predicted_species_train
ifelse(predicted_species_train>0.5,1,0)->predicted_species_train
table(downscaledtest$Class,predicted_species_train)
```

Accuracy:



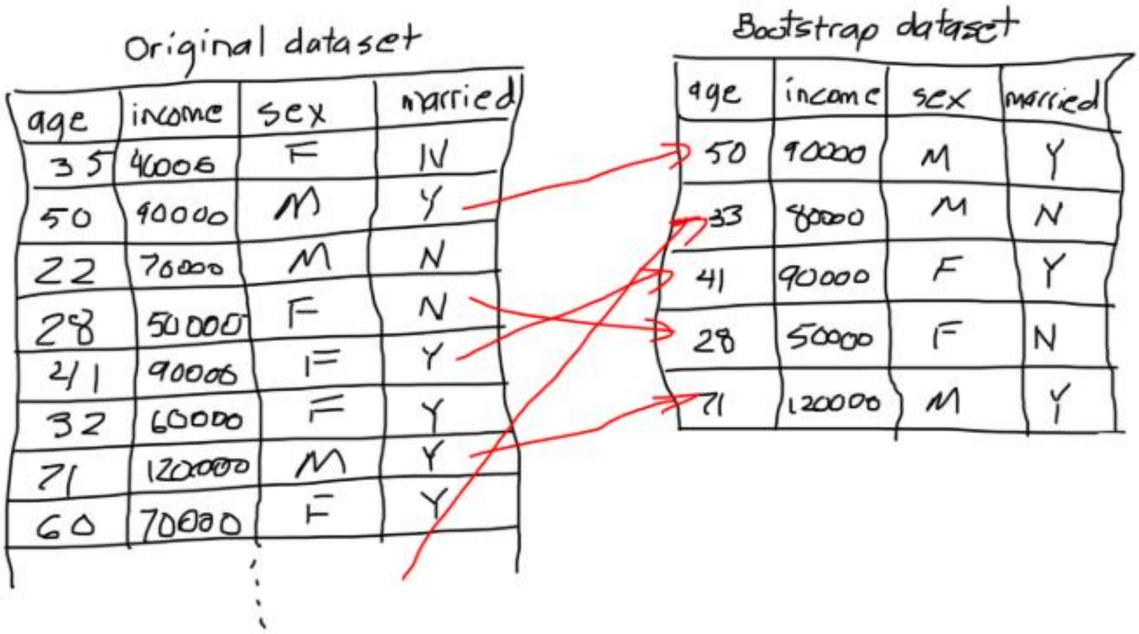
```
table(downscaledtest$Class,predicted_species_train)
 predicted_species_train
      0    1
0  9892 5172
1  7281 7783
```

Accuracy for this model is 60%

RANDOM FOREST ALGORITHM:

The random forest algorithm works by aggregating the predictions made by multiple decision trees of varying depth. Every decision tree in the forest is trained on a subset of the dataset called the bootstrapped dataset.

Example:



Original dataset

age	income	sex	married
35	46006	F	N
50	90000	M	Y
22	70000	M	N
28	50000	F	N
41	90006	F	Y
32	60000	F	Y
71	120000	M	Y
60	70000	F	Y

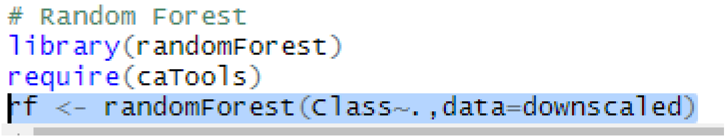
Bootstrap dataset

age	income	sex	married
50	90000	M	Y
33	80000	M	N
41	90000	F	Y
28	50000	F	N
71	120000	M	Y

Recall how when deciding on the criteria with which to split a decision tree, we measured the impurity produced by each feature using the Gini index or entropy. In random forest, however, we randomly select a predefined number of feature as candidates. The latter will result in a larger variance between the trees which would otherwise contain the same features (i.e those which are highly correlated with the target label).

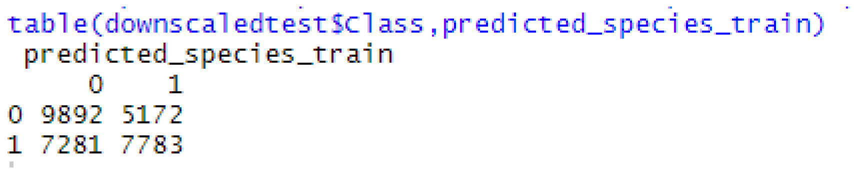
When the random forest is used for classification and is presented with a new sample, the final prediction is made by taking the majority of the predictions made by each individual decision tree in the forest. In the event, it is used for regression and it is presented with a new sample, the final prediction is made by taking the average of the predictions made by each individual decision tree in the forest

R CODE:



```
# Random Forest
library(randomForest)
require(caTools)
rf <- randomForest(Class~.,data=downscaled)
```

Accuracy:



```
table(downscaledtest$Class,predicted_species_train)
 predicted_species_train
     0    1
0 9892 5172
1 7281 7783
```

3.5 Need for Bayesian approach of hyper parameter optimization:

An ML algorithm generally has a loss/cost function which needs to be minimized subject to the values of hyper parameters and parameters of the algorithm.

Weights and Biases are decided on the basis of Optimization function of the ML algorithm. Selecting good hyper parameters further minimizes the loss function, and makes model more robust and accurate, increasing the overall efficiency of the model.

- For instance in Gradient Descent, it is very important to select a good learning rate to reach the convergence point in shortest time, because if it is large the cost function will overshoot and won't be able to find minima or if too small it will take forever to reach the minima.
- Now even though Scikit-Learn provides us Grid Search and Random Search, these algorithms are brute force and the computation time grows as grid becomes denser. Even the best possible combination might be far from the optimal.
- For example, the common implementation of random search completely ignores information on the trials already computed, and each new sample is drawn from the same initial distribution.
- Fortunately, there is a way to account for them. Say, you were tuning C - regularization parameter for logistic regression. If one particular value gives really bad results - the points in vicinity will also perform poorly, so there is really very little need to sample from this region.
- We would like to incorporate this information into our strategy - in other words, we want to get more points from the regions with high probability of yielding good result and get less points from elsewhere.

Bayesian Optimization library uses Bayesian approach for intelligent points

3.6 Light BGM:

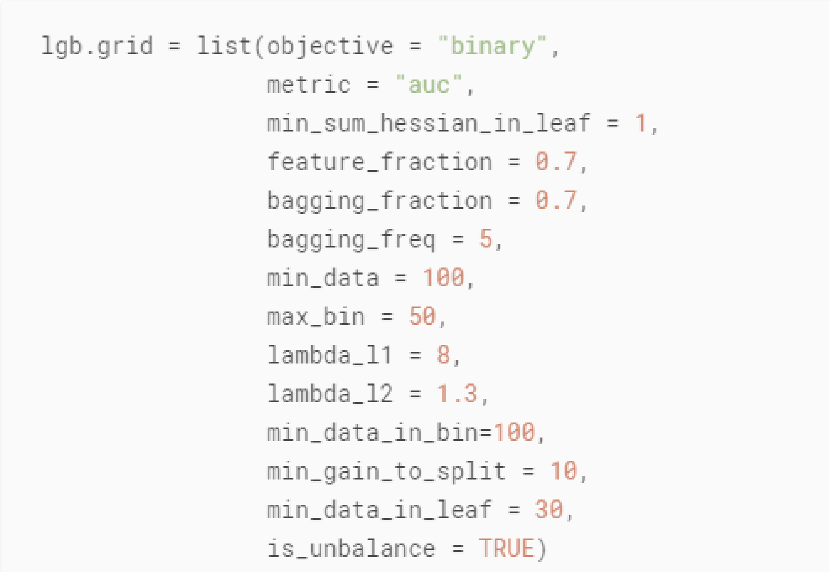
Light GBM is a gradient boosting framework that uses tree based learning algorithm. It grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

It is 'Light' because of its high speed. It can handle large data, requires low memory to run and focuses on accuracy of results.

Since in our dataset, all the features are independent of each other, it's better to train an ensemble of 200 LGB models.

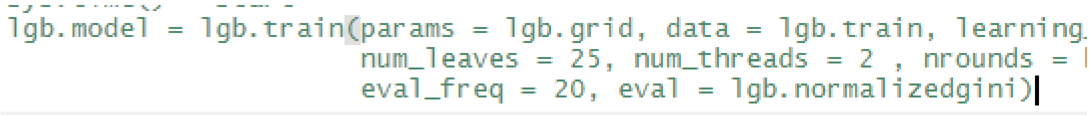
Each training model with take 2 features at a time: The original one and an extra column with the unique values count. (For example: var_0 and new_var_0)

Setting up LGBM Parameters



```
lgb.grid = list(objective = "binary",
                metric = "auc",
                min_sum_hessian_in_leaf = 1,
                feature_fraction = 0.7,
                bagging_fraction = 0.7,
                bagging_freq = 5,
                min_data = 100,
                max_bin = 50,
                lambda_l1 = 8,
                lambda_l2 = 1.3,
                min_data_in_bin=100,
                min_gain_to_split = 10,
                min_data_in_leaf = 30,
                is_unbalance = TRUE)
```

Using this method, saved us a lot of time for hyper parameter optimisation, instead of running a single LGB model which takes into account the interaction of all 400 features.



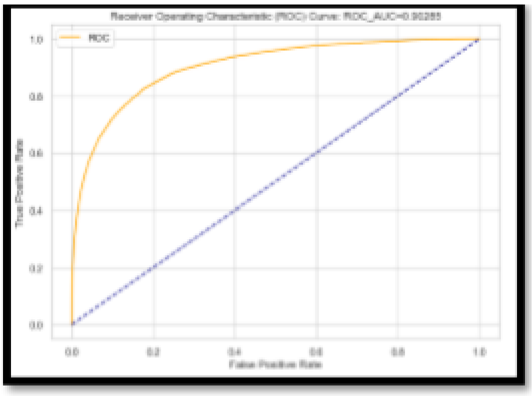
```
lgb.model = lgb.train(params = lgb.grid, data = lgb.train, learning
                      num_leaves = 25, num_threads = 2 , nrounds =
                      eval_freq = 20, eval = lgb.normalizedgini)
```

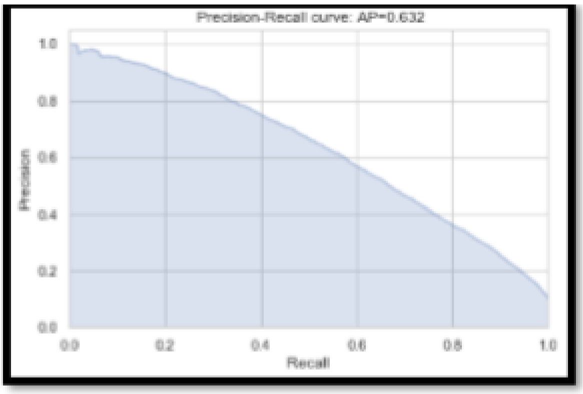
Without hyperparameter tuning of LGB model we got average precision score of 0.593, and ROC_AUC score of 0.88645.

Bayesian Optimization gave us the following optimal parameters:

{'bagging_fraction': 0.7, 'bagging_freq': 2, 'boost_from_average': False, 'is_unbalance': True, 'lambda_l1': 0.7, 'lambda_l2': 1.9999999964082154, 'learning_rate': 0.009999999503478726, 'max_depth': 5, 'metric': 'auc', 'min_data_in_leaf': 25, 'min_gain_to_split': 0.9880270270985563, 'min_sum_hessian_in_leaf': 20.0, 'num_leaves': 30, 'objective': 'binary'}

After hyperparameter tuning of LGB model with BayesianOptimization library, we achieved the following results.





From the plots above, we got an average precision score of 0.632, and ROC_AUC score of 0.90285.

Chapter 4

Summary

Santander is interested in finding which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

Hence, it is interested in correctly identifying the customers with target label as 1, (i.e. customers who will make a specific transaction in the future)

Since our dataset is an imbalance class dataset, where the proportion of positive samples is low (around 10%), we should aim for higher precision since it does not include True negatives in calculation, and hence it will not affected by class imbalance.

Therefore, precision-recall (PR) curve should be chosen as an evaluation metric instead of ROC curves in this scenario.

Among the Machine Learning models, I used I prefer Random Forest among decision trees and logistic regression as it has more accuracy. KNN algorithm is much slower in execution as it calculates distances between two rows.