Name: Deep Nagariya

**Background** -

At Santander, mission is to help people and businesses prosper. We are always looking

for ways to help our customers understand their financial health and identify which

products and services might help them achieve their monetary goals.

Our data science team is continually challenging our machine learning algorithms,

working with the global data science community to make sure we can more accurately

identify new ways to solve our most common challenge, binary classification problems

such as: is a customer satisfied? Will a customer buy this product? Can a customer pay

this loan?

**Problem Statement** -

In this challenge, we need to identify which customers will make a specific transaction in

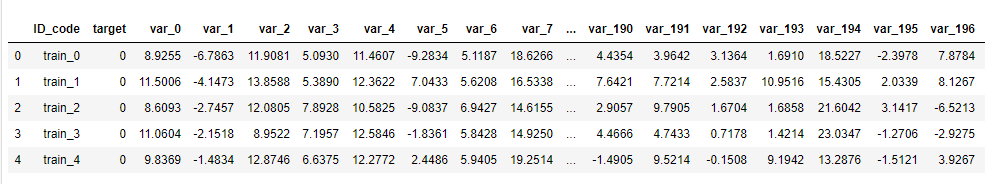
the future, irrespective of the amount of money transacted.

**1 : Data**

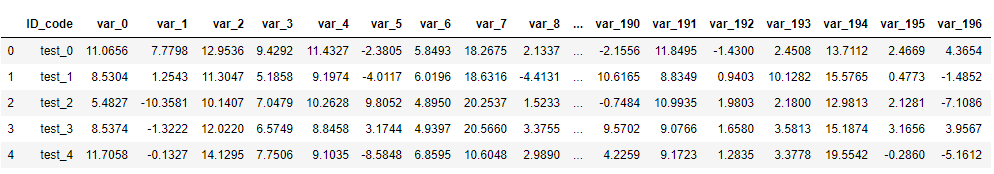
The main task is to build the classification models which will be used to predict which customers will make a specific transaction in the future.

Given is the head (5) of the data we got of both train and test:

Train:

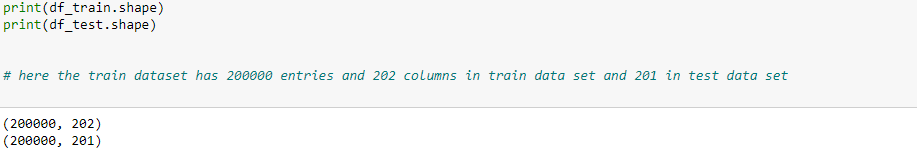


Test :



Checking the shape of the data and I found:

That it has 200000 rows and 202 columns in train data and 201 in the test data.

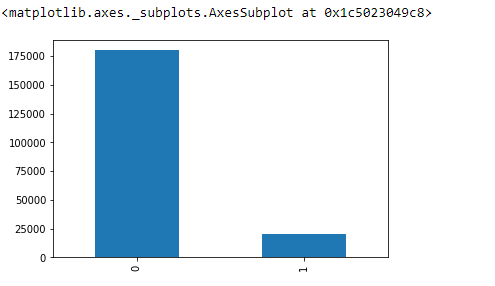


**2: Exploratory data analysis**

Exploratory data analysis is an approach to analysing data sets to summarize their main characteristics, often with visual methods. A statistical model can be used or not, but primarily EDA is for seeing what the data can tell us beyond the formal modelling or hypothesis testing task.

It involves the loading dataset, target classes count, data cleaning, typecasting of attributes, missing value analysis, Attributes distributions and trends. So, we must clean the data otherwise it will effect on performance of the model.

The first is the target classes count check where we see the distribution of the target class data.

As seen in the figure. 

Observation:

We are having an imbalanced data, where 89.951% of the data is no. of customers who will not make a transaction (0) & 10.049 % of the data are those who will make a transaction (1).

Missing values analysis:

In this, we have to find out any missing values are present in dataset. If it’s present then either delete or impute the values using mean, median and KNN imputation method. We have not found any missing values in both train and test data.

Code:

For python:



For R:

#Missing Value Analysis:-

#Finding the missing values in train data

missing\_val<-data.frame(missing\_val=apply(df\_train,2,function(x){sum(is.na(x))}))

missing\_val<-sum(missing\_val)

missing\_val

#Finding the missing values in test data

missing\_val<-data.frame(missing\_val=apply(df\_test,2,function(x){sum(is.na(x))}))

missing\_val<-sum(missing\_val)

missing\_val

Next, we are going to check is the distribution trends.

Distribution of train attributes:

Fig: 1



Figure 2:



Observations:

- We can observe that there is a considerable no. of features which are significantly have different distribution for two target variables. For example, like var\_0, var\_1, var\_9, var\_198 var\_180 etc.

- We can observe that there is a considerable number of features which are significantly have same distribution for two target variables. For example, like var\_3, var\_7, var\_10, var\_171, var\_185 etc.

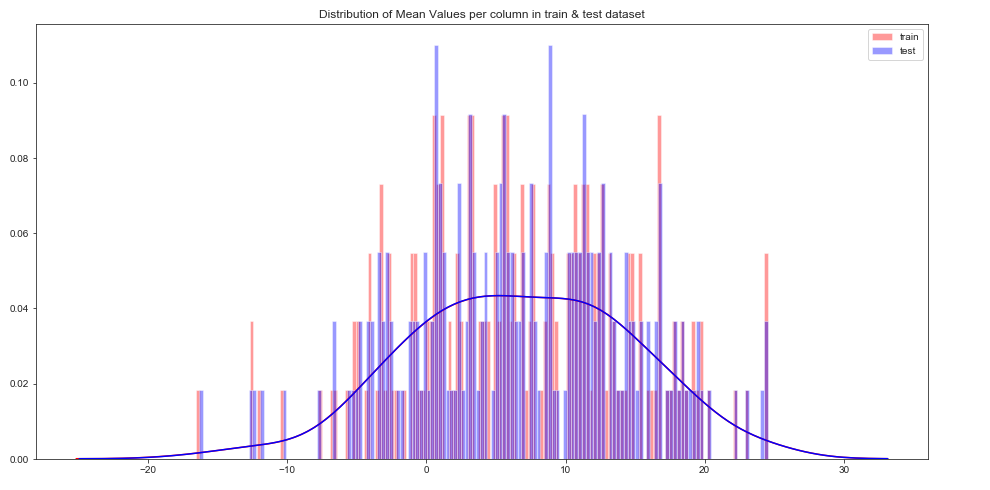
Distribution of test attributes:



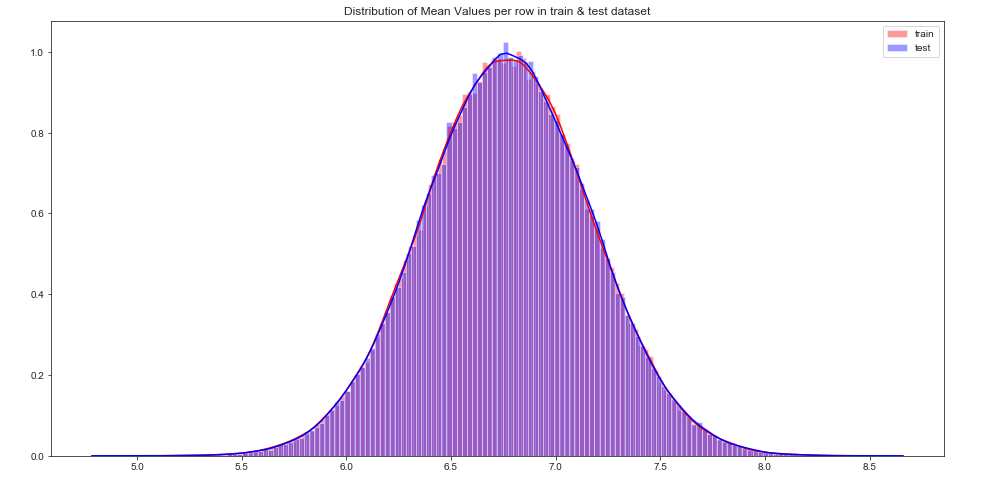


Distribution of mean values in both train and test dataset: -

Let us look distribution of mean values per column in train and test dataset

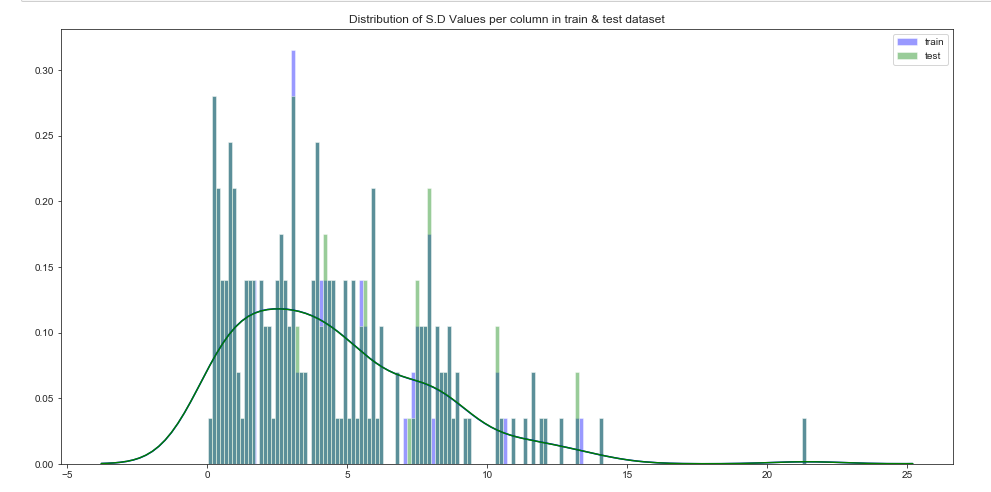


Let us look distribution of mean values per row in train and test dataset:-

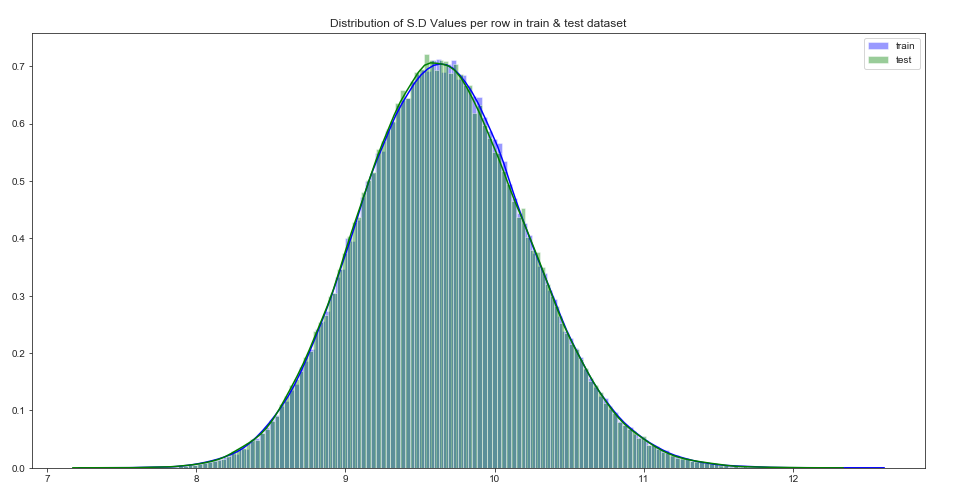


Distribution of standard deviation (std) values in train and test dataset: -

Let us look distribution of standard deviation (std) values per column in train and test dataset

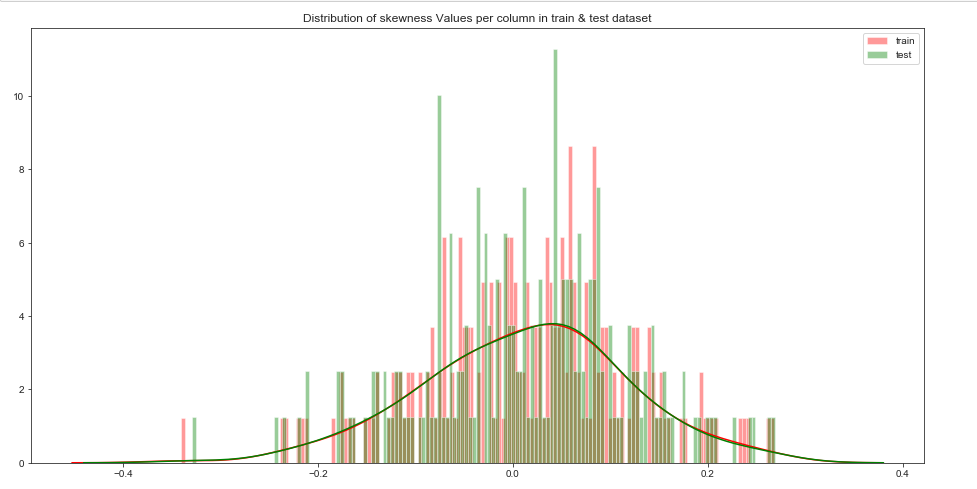


Let us look distribution of standard deviation (std) values per column in train and test dataset

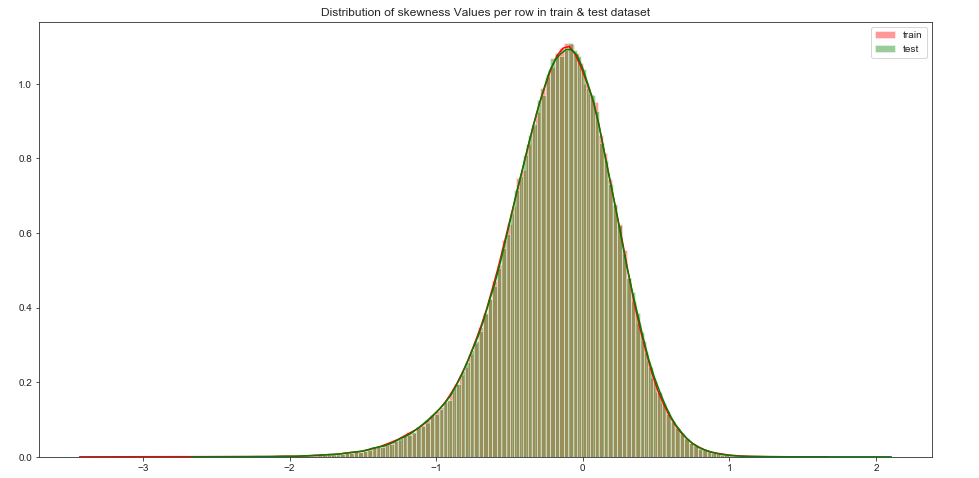


Distribution of skewness values in train and test dataset

Let us look distribution of skewness values per column in train and test dataset:-

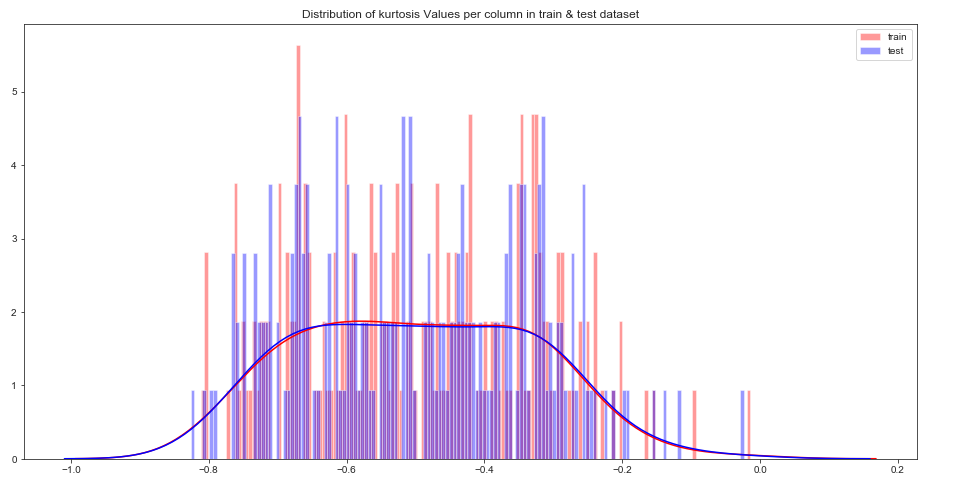


Let us look distribution of skewness values per column in train and test dataset:-

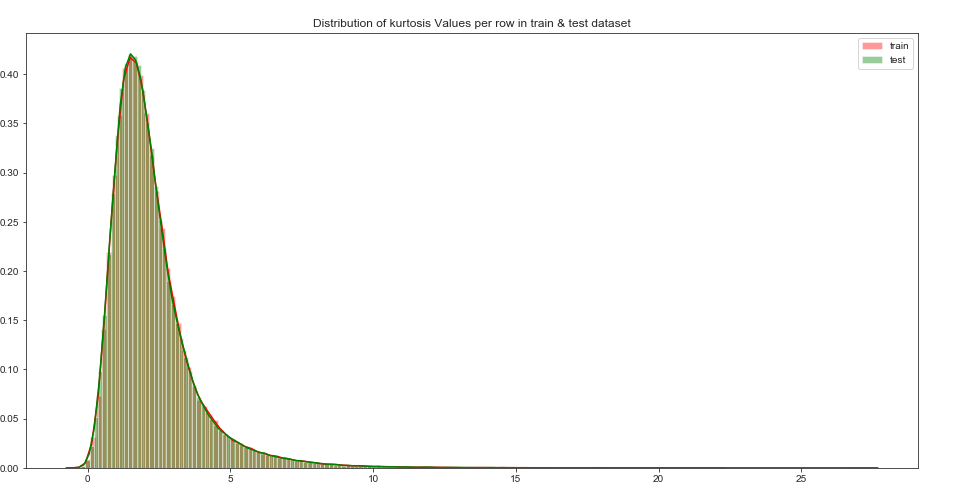


Distribution of kurtosis values in train and test dataset

Let us look distribution of kurtosis values per column in train and test dataset:-



Let us look distribution of kurtosis values per row in train and test dataset :



Outlier analysis:

we haven’t performed outlier analysis due to the data is imbalanced and also not required for imbalanced data.

Label Encoding: Here even this process isn’t required as the data which have obtained was already in the numeric form and no attributes were in the variable form.

2.1.3 Feature Selection

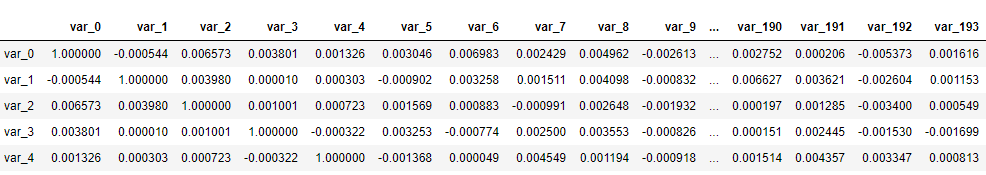
Feature selection is very important for modelling the dataset. Every dataset have good and unwanted features. The unwanted features would effect on performance of model, so we have to delete those features. We have to select best features by using ANOVA, Chi-Square test and correlation matrix statistical techniques and so on. In this, we are selecting best features by using Correlation matrix.

Correlation matrix

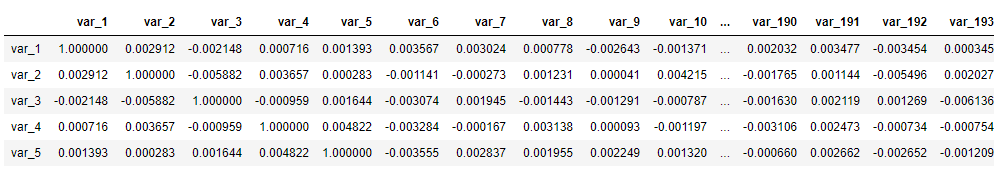
Correlation matrix, it tells about linear relationship between attributes and help us to build better models.

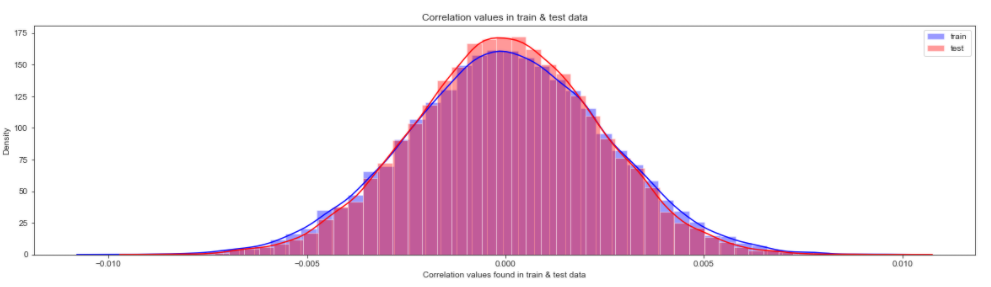
From correlation distribution plot, we can observe that correlation between both train and test attributes are very small. It means that all both train and test attributes are independent to each other

Train data:



Test data:



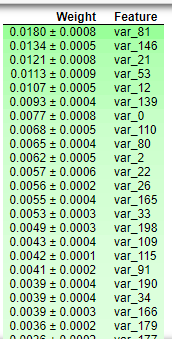


2.1.4 Feature engineering

Let us do some feature engineering by using Permutation importance:

Permutation importance: -

Permutation variable importance measure in a random forest for classification and regression. The variables which are mostly contributed to predict the model.



The observation which I got was: We can observed that the top important features are var\_12, var\_26, var\_22,v var\_174, var\_198 and so on based on Mean decrease gini.

2.1 Modelling

2.1.1 Model Selection

After all early stages of pre-processing, then model the data. So, we have to select best model for this project with the help of some metrics.

The dependent variable can fall in either of the four categories:

1. Nominal

2. Ordinal

3. Interval

4. Ratio

If the dependent variable is Nominal the only predictive analysis that we can perform is Classification, and if the dependent variable is Interval or Ratio like this project, the normal method is to do a Regression analysis, or classification after binning.

Handling of imbalance data

Now we are going to explore 5 different approaches for dealing with imbalanced datasets.

• Change the performance metric

• Oversample minority class

• Under sample majority class

• Synthetic Minority Oversampling Technique (SMOTE) in Python or Random Oversampling Examples (ROSE) in R

• Change the algorithm

We always start model building from the simplest to more complex.

Accuracy of the model is not the best metric to use when evaluating the imbalanced datasets as it may be misleading. So, we are going to change the performance metric.

Oversample Minority Class: -

-Adding more copies of minority class. -It can be a good option we don’t have that much large data to work.

-Drawback of this process is we are adding info. That can lead to overfitting or poor performance on test data.

Under sample Majority class: - -Removing some copies of majority class. -It can be a good option if we have very large amount of data say in millions to work.

-Drawback of this process is we are removing some valuable info. that can lead to underfitting & poor performance on test data.

As per the drawbacks of both the model we will use SMOTE (Synthetic Minority Oversampling technique) that is best than the above as compare to above one's.

Synthetic Minority Oversampling Technique (SMOTE)

SMOTE uses a nearest neighbour’s algorithm to generate new and synthetic data to use for training the model. In order to balance imbalanced data, we are going to use SMOTE sampling method.

Light GBM:

Light GBM is a gradient boosting framework that uses tree-based learning algorithms. We are going to use Light GBM model.

Conclusion:

Model Evaluation

Now, we have built three models for predicting the target variable, but we need to decide which model better for this project. There are many metrics used for model evaluation.

Classification accuracy may be misleading if we have an imbalanced dataset or if we have more than two classes in dataset.

For classification problems, the confusion matrix used for evaluation. But, in our case the data is imbalanced. So, roc\_auc\_score is used for evaluation.

In this project, we are using two metrics for model evaluation as follows,

Confusion Matrix: - It is a technique for summarizing the performance of a classification algorithm.

The number of correct predictions and incorrect predictions are summarized with count values and broken down by each class.

Accuracy: - The ratio of correct predictions to total predictions

Accuracy = (𝑇𝑃+𝑇𝑁)/𝑇𝑜𝑡𝑎𝑙 𝑃𝑟𝑒𝑑𝑖𝑐𝑡𝑖𝑜𝑛𝑠

Misclassification error: - The ratio of incorrect predictions to total predictions

Error rate = (𝐹𝑁+𝐹𝑃)/𝑇𝑜𝑡𝑎𝑙 𝑝𝑟𝑒𝑑𝑖𝑐𝑡𝑖𝑜𝑛𝑠

Accuracy=1-Error rate

True Positive Rate (TPR) = 𝑇𝑃/(𝑇𝑃+𝐹𝑁) ↔ Recall

Precision = 𝑇𝑃/ (𝑇𝑃+𝐹𝑃)

True Negative Rate (TNR) = 𝑇𝑁/(𝑇𝑁+𝐹𝑃) ↔ Specificity

False Positive Rate (FPR) = 𝐹𝑃/ (𝐹𝑃+𝑇𝑁)

False Negative rate (FNR) = 𝐹𝑁/ (𝐹𝑁+𝑇𝑃)

F1 Score also known as Harmonic mean of precision and recall, used to indicate balance between them.

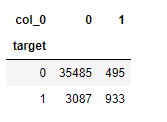
F1 score = 2∗𝑃𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛∗𝑅𝑒𝑐𝑎𝑙𝑙 / Precision + Recall

Receiver operating characteristics (ROC)\_Area under curve (AUC) Score

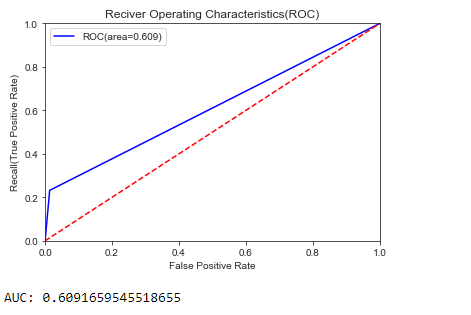
roc\_auc\_score: - It is a metric that computes the area under the Roc curve and also used metric for imbalanced data.

Roc curve is plotted true positive rate or Recall on y axis against false positive rate or specificity on x axis. The larger the area under the roc curve better the performance of the model.

Logistic Regression:

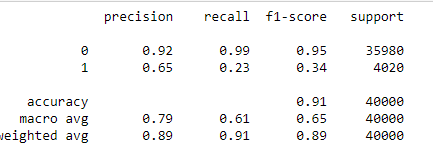


When we compare the roc\_auc\_score and cross validation score, conclude that model is not performing well on imbalanced data.



Classification report:

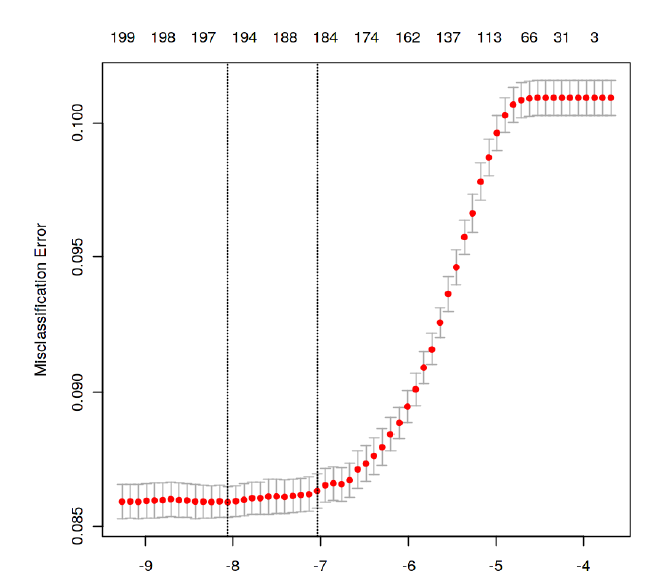
The classification report we got was:

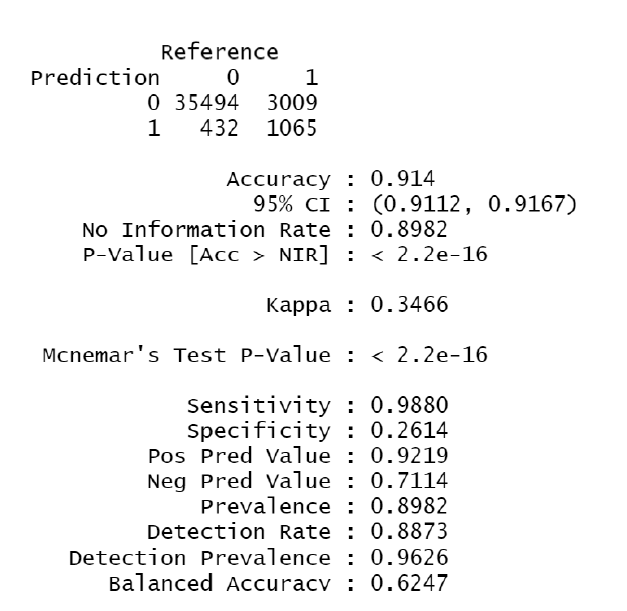


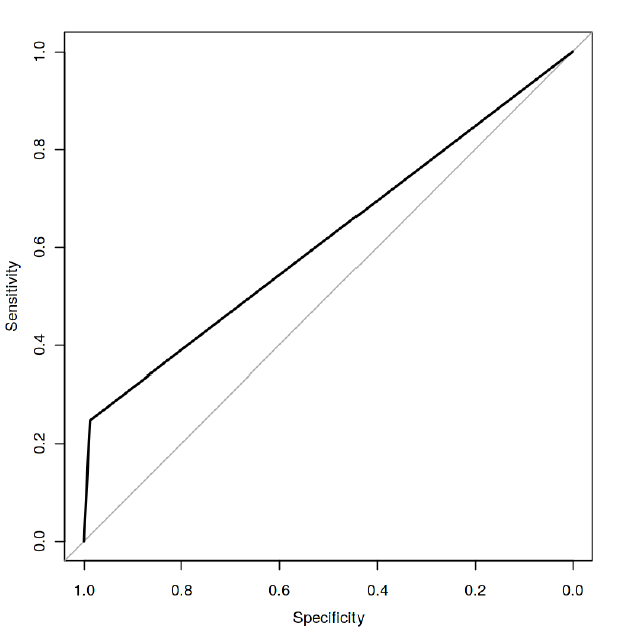
We can observe that f1 score is high for number of customers those who will not make a transaction then who will make a transaction. So, we are going to change the algorithm.

R Output:

Logistic regression

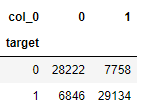


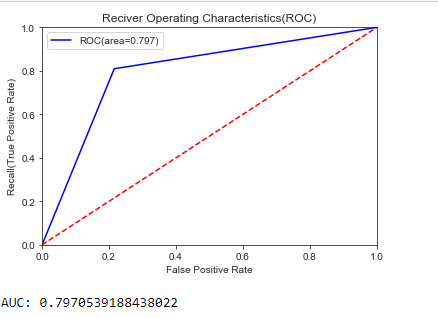




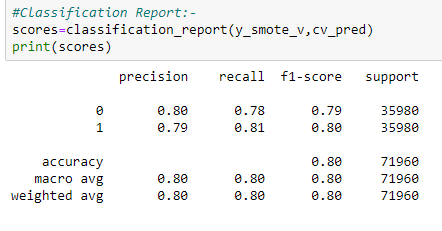
SMOTE:

Confusion matrix output:

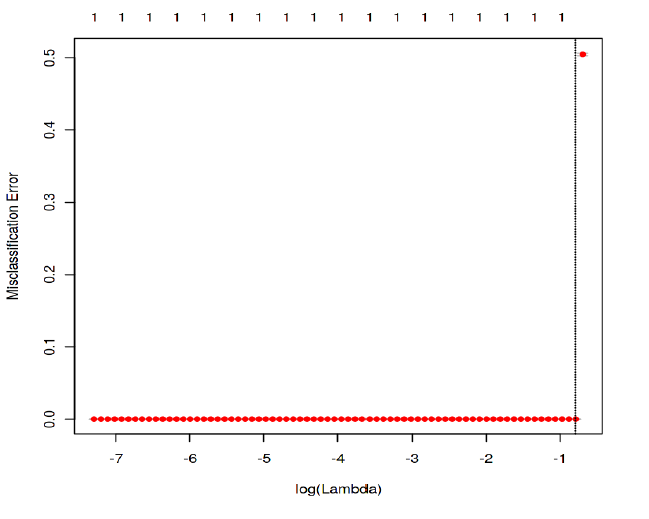


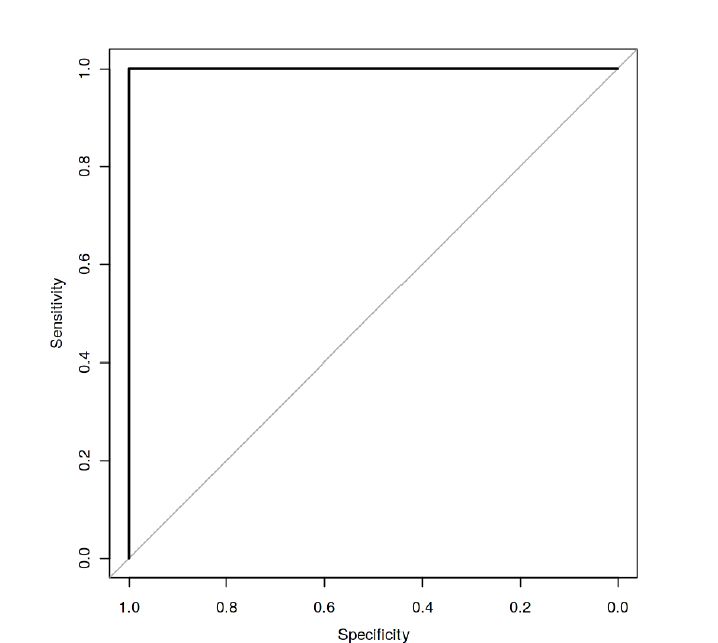


Classification report:



R output:





I tried different ways to get good accuracy like changing count of one target class variable. Finally got area under ROC curve is 1 but this may not be possible.

**Model Selection:**

When we compare scores of area under the ROC curve of all the models for an imbalanced data. We could conclude that below points as follow,

1. Logistic regression model is not performed well on imbalanced data.

2. We balance the imbalanced data using resampling techniques like SMOTE in python and ROSE in R.

3. Baseline logistic regression model is performed well on balanced data.

4. Light GBM model performed well on imbalanced data.

Finally, LightGBM is best choice for identifying which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

Thus, I think further in the future there would be identification on which customers will make a specific transaction with respect to the customer being happy or satisfied.

Also, in the future with this model we could further identify which customer segment to target in the launch of a product or which product to sell to which customer.

Further, with this we can identify which customer can pay this loan.

So with this model I think the given transaction being processed or completed will help to find which customer has a more stable credit score and thus helping the above causes as well.

My R code :

#Clearing Objects:-

rm(list=ls(all=T))

#Loading Libraries:-

library(tidyverse)

library(moments)

library(DataExplorer)

library(caret)

library(Matrix)

library(mlbench)

library(caTools)

library(randomForest)

library(glmnet)

library(mlr)

library(vita)

library(rBayesianOptimization)

library(lightgbm)

library(pROC)

library(DMwR)

library(ROSE)

library(yardstick)

library(ROCR)

getwd()

setwd("C:/Users/Deep/Desktop/Data Scientist/Project/Project 1")

#Importing the training Data:-

df\_train=read.csv("train.csv")

#Dimension of the train data:-

dim(df\_train)

#Summary of the train dataset:-

str(df\_train)

#Typecasting the target variable:-

df\_train$target=as.factor(df\_train$target)

#Target class count in train data:-

table(df\_train$target)

#Percentage count of target class in train data:-

a = table(df\_train$target)/length(df\_train$target)

a\*100

#Bar plot for count of target classes in train data:-

plot1=ggplot(df\_train,aes(target))+theme\_bw()+geom\_bar(stat='count',fill='lightgreen')

plot1

#Observation:- We are having a unbalanced data, where 89.95% of the data is no. of customers who will not make a transaction

#& 10.049 % of the data are those who will make a transaction.

#Distribution of train attributes from 3 to 102:-

for (var in names(df\_train)[c(3:102)]){

target<-df\_train$target

plot<-ggplot(df\_train, aes(df\_train[[var]],fill=target)) +

geom\_density(kernel='gaussian') + ggtitle(var)+theme\_classic()

print(plot)

}

#Distribution of train attributes from 103 to 202:-

for (var in names(df\_train)[c(103:202)]){

target<-df\_train$target

plot<-ggplot(df\_train, aes(df\_train[[var]],fill=target)) +

geom\_density(kernel='gaussian') + ggtitle(var)+theme\_classic()

print(plot)

}

#Importing the test data:-

df\_test=read.csv("test.csv")

head(df\_test)

#Dimension of test dataset:-

dim(df\_test)

#Distribution of test attributes from 2 to 101:-

plot\_density(df\_test[,c(2:101)],ggtheme = theme\_classic(),geom\_density\_args = list(color='red'))

#Distribution of test attributes from 102 to 201:-

plot\_density(df\_test[,c(102:201)],ggtheme = theme\_classic(),geom\_density\_args = list(color='red'))

#Mean value per rows and columns in train & test dataset:-

#Applying the function to find mean values per row in train and test data.

train\_mean<-apply(df\_train[,-c(1,2)],MARGIN=1,FUN=mean)

test\_mean<-apply(df\_test[,-c(1)],MARGIN=1,FUN=mean)

ggplot()+

#Distribution of mean values per row in train data

geom\_density(data=df\_train[,-c(1,2)],aes(x=train\_mean),kernel='gaussian',show.legend=TRUE,color='blue')+theme\_classic()+

#Distribution of mean values per row in test data

geom\_density(data=df\_test[,-c(1)],aes(x=test\_mean),kernel='gaussian',show.legend=TRUE,color='green')+

labs(x='mean values per row',title="Distribution of mean values per row in train and test dataset")

#Applying the function to find mean values per column in train and test data.

train\_mean<-apply(df\_train[,-c(1,2)],MARGIN=2,FUN=mean)

test\_mean<-apply(df\_test[,-c(1)],MARGIN=2,FUN=mean)

ggplot()+

#Distribution of mean values per column in train data

geom\_density(aes(x=train\_mean),kernel='gaussian',show.legend=TRUE,color='blue')+theme\_classic()+

#Distribution of mean values per column in test data

geom\_density(aes(x=test\_mean),kernel='gaussian',show.legend=TRUE,color='green')+

labs(x='mean values per column',title="Distribution of mean values per column in train and test dataset")

#Applying the function to find standard deviation values per row in train and test data.

train\_sd<-apply(df\_train[,-c(1,2)],MARGIN=1,FUN=sd)

test\_sd<-apply(df\_test[,-c(1)],MARGIN=1,FUN=sd)

ggplot()+

#Distribution of sd values per row in train data

geom\_density(data=df\_train[,-c(1,2)],aes(x=train\_sd),kernel='gaussian',show.legend=TRUE,color='red')+theme\_classic()+

#Distribution of sd values per row in test data

geom\_density(data=df\_test[,-c(1)],aes(x=test\_sd),kernel='gaussian',show.legend=TRUE,color='blue')+

labs(x='sd values per row',title="Distribution of sd values per row in train and test dataset")

#Applying the function to find sd values per column in train and test data.

train\_sd<-apply(df\_train[,-c(1,2)],MARGIN=2,FUN=sd)

test\_sd<-apply(df\_test[,-c(1)],MARGIN=2,FUN=sd)

ggplot()+

#Distribution of sd values per column in train data

geom\_density(aes(x=train\_sd),kernel='gaussian',show.legend=TRUE,color='red')+theme\_classic()+

#Distribution of sd values per column in test data

geom\_density(aes(x=test\_sd),kernel='gaussian',show.legend=TRUE,color='blue')+

labs(x='sd values per column',title="Distribution of std values per column in train and test dataset")

#Applying the function to find skewness values per row in train and test data.

train\_skew<-apply(df\_train[,-c(1,2)],MARGIN=1,FUN=skewness)

test\_skew<-apply(df\_test[,-c(1)],MARGIN=1,FUN=skewness)

ggplot()+

#Distribution of skewness values per row in train data

geom\_density(aes(x=train\_skew),kernel='gaussian',show.legend=TRUE,color='green')+theme\_classic()+

#Distribution of skewness values per column in test data

geom\_density(aes(x=test\_skew),kernel='gaussian',show.legend=TRUE,color='blue')+

labs(x='skewness values per row',title="Distribution of skewness values per row in train and test dataset")

#Applying the function to find skewness values per column in train and test data.

train\_skew<-apply(df\_train[,-c(1,2)],MARGIN=2,FUN=skewness)

test\_skew<-apply(df\_test[,-c(1)],MARGIN=2,FUN=skewness)

ggplot()+

#Distribution of skewness values per column in train data

geom\_density(aes(x=train\_skew),kernel='gaussian',show.legend=TRUE,color='green')+theme\_classic()+

#Distribution of skewness values per column in test data

geom\_density(aes(x=test\_skew),kernel='gaussian',show.legend=TRUE,color='blue')+

labs(x='skewness values per column',title="Distribution of skewness values per column in train and test dataset")

#Applying the function to find kurtosis values per row in train and test data.

train\_kurtosis<-apply(df\_train[,-c(1,2)],MARGIN=1,FUN=kurtosis)

test\_kurtosis<-apply(df\_test[,-c(1)],MARGIN=1,FUN=kurtosis)

ggplot()+

#Distribution of kurtosis values per row in train data

geom\_density(aes(x=train\_kurtosis),kernel='gaussian',show.legend=TRUE,color='blue')+theme\_classic()+

#Distribution of kurtosis values per row in test data

geom\_density(aes(x=test\_kurtosis),kernel='gaussian',show.legend=TRUE,color='red')+

labs(x='kurtosis values per row',title="Distribution of kurtosis values per row in train and test dataset")

#Applying the function to find kurtosis values per column in train and test data.

train\_kurtosis<-apply(df\_train[,-c(1,2)],MARGIN=2,FUN=kurtosis)

test\_kurtosis<-apply(df\_test[,-c(1)],MARGIN=2,FUN=kurtosis)

ggplot()+

#Distribution of kurtosis values per column in train data

geom\_density(aes(x=train\_kurtosis),kernel='gaussian',show.legend=TRUE,color='blue')+theme\_classic()+

#Distribution of kurtosis values per column in test data

geom\_density(aes(x=test\_kurtosis),kernel='gaussian',show.legend=TRUE,color='red')+

labs(x='kurtosis values per column',title="Distribution of kurtosis values per column in train and test dataset")

#Missing Value Analysis:-

#Finding the missing values in train data

missing\_val<-data.frame(missing\_val=apply(df\_train,2,function(x){sum(is.na(x))}))

missing\_val<-sum(missing\_val)

missing\_val

#Finding the missing values in test data

missing\_val<-data.frame(missing\_val=apply(df\_test,2,function(x){sum(is.na(x))}))

missing\_val<-sum(missing\_val)

missing\_val

#Correlations in train data:-

#convert factor to int

df\_train$target<-as.numeric(df\_train$target)

train\_correlation<-cor(df\_train[,c(2:202)])

train\_correlation

#Observation:- We can observe that correlation between train attributes is very small.

#Correlations in test data

test\_correlation<-cor(df\_test[,c(2:201)])

test\_correlation

#Observation:- We can observe that correlation between test attributes is very small.

#Feature Enginnering:- Performing some feature engineering on datasets:-

#Variable Importance:-Variable importance is used to see top features in dataset based on mean decreases gini .

#Building a simple model to find features which are imp:-

#Split the training data using simple random sampling

train\_index<-sample(1:nrow(df\_train),0.75\*nrow(df\_train))

#train data

train\_data<-df\_train[train\_index,]

#validation data

valid\_data<-df\_train[-train\_index,]

#dimension of train and validation data

dim(train\_data)

dim(valid\_data)

#Random forest classifier:-

#Training the Random forest classifier

#convert to int to factor

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

#setting the mtry

mtry<-floor(sqrt(200))

#setting the tunegrid

tuneGrid<-expand.grid(.mtry=mtry)

#fitting the random forest

rf<-randomForest(target~.,train\_data[,-c(1)],mtry=mtry,ntree=10,importance=TRUE)

#Feature importance by random forest-

#Variable importance

VarImp = importance(rf, type = 2)

VarImp

#Observation:-We can observed that the top important features are var\_12, var\_26, var\_22,v var\_174, var\_198 and so on based on Mean decrease gini index.

#Handling of imbalanced data- Now we are going to explore 5 different approaches for dealing with imbalanced datasets.

#Change the performance metric

#Oversample minority class

#Undersample majority class

#ROSE

#LightGBM

#Logistic Regression Model:-

#Split the data using simple random sampling:-

train.index<-sample(1:nrow(df\_train),0.8\*nrow(df\_train))

#train data

train.data<-df\_train[train.index,]

#validation data

valid.data<-df\_train[-train.index,]

#dimension of train data

dim(train.data)

#dimension of validation data

dim(valid.data)

#target classes in train data

table(train.data$target)

#target classes in validation data

table(valid.data$target)

#Training and validation dataset

#Training dataset

X\_t<-as.matrix(train.data[,-c(1,2)])

y\_t<-as.matrix(train.data$target)

#validation dataset

X\_v<-as.matrix(valid.data[,-c(1,2)])

y\_v<-as.matrix(valid.data$target)

#test dataset

test<-as.matrix(df\_test[,-c(1)])

#Logistic regression model

lr\_model <-glmnet(X\_t,y\_t, family = "binomial")

summary(lr\_model)

#Cross validation prediction

cv\_lr <- cv.glmnet(X\_t,y\_t,family = "binomial", type.measure = "class")

cv\_lr

#Plotting the misclassification error vs log(lambda) where lambda is regularization parameter

#Minimum lambda

cv\_lr$lambda.min

#plot the auc score vs log(lambda)

plot(cv\_lr)

#Observation:-We can observed that miss classification error increases as increasing the log(Lambda).

#Model performance on validation dataset

cv\_predict.lr<-predict(cv\_lr,X\_v,s = "lambda.min", type = "class")

cv\_predict.lr

#Observation:-Accuracy of the model is not the best metric to use when evaluating the imbalanced datasets as it may be misleading. So, we are going to change the performance metric.

#Confusion Matrix:-

#actual target variable

target<-valid.data$target

#convert to factor

target<-as.factor(target)

#predicted target variable

#convert to factor

cv\_predict.lr<-as.factor(cv\_predict.lr)

confusionMatrix(data=cv\_predict.lr,reference=target)

#Reciever operating characteristics(ROC)-Area under curve(AUC) score and curve:-

#ROC\_AUC score and curve

cv\_predict.lr<-as.numeric(cv\_predict.lr)

roc(data=valid.data[,-c(1,2)],response=target,predictor=cv\_predict.lr,auc=TRUE,plot=TRUE)

#Oversample Minority Class:-

#-Adding more copies of minority class.

#-It cab be a good option we dont have that much large data to work.

#-Drawback of this process is we are adding info. That can lead to overfitting or poor performance on test data.

#Undersample Mojorityclass:-

#-Removing some copies of majority class.

#-It can be a good option if we have very large amount of data say in millions to work.

#-Drawback of this process is we are removing some valuable info. that can leads to underfitting & poor performance on test data.

#Both Oversampling and undersampling techniques have some drawbacks. So, we are not going to use this models for this problem and also we will use other best algorithms.

#Random Oversampling Examples(ROSE)- It creates a sample of synthetic data by enlarging the features space of minority and majority class examples.

#Random Oversampling Examples(ROSE)

train.rose <- ROSE(target~., data =train.data[,-c(1)],seed=32)$data

#target classes in balanced train data

table(train.rose$target)

valid.rose <- ROSE(target~., data =valid.data[,-c(1)],seed=42)$data

#target classes in balanced valid data

table(valid.rose$target)

#Logistic regression model

lr\_rose <-glmnet(as.matrix(train.rose),as.matrix(train.rose$target), family = "binomial")

summary(lr\_rose)

#Cross validation prediction

cv\_rose = cv.glmnet(as.matrix(valid.rose),as.matrix(valid.rose$target),family = "binomial", type.measure = "class")

cv\_rose

#Plotting the misclassification error vs log(lambda) where lambda is regularization parameter:-

#Minimum lambda

cv\_rose$lambda.min

#plot the auc score vs log(lambda)

plot(cv\_rose)

#Model performance on validation dataset

cv\_predict.rose<-predict(cv\_rose,as.matrix(valid.rose),s = "lambda.min", type = "class")

cv\_predict.rose

#Confusion matrix

#actual target variable

target<-valid.rose$target

#convert to factor

target<-as.factor(target)

#predicted target variable

#convert to factor

cv\_predict.rose<-as.factor(cv\_predict.rose)

#Confusion matrix

confusionMatrix(data=cv\_predict.rose,reference=target)

#ROC\_AUC score and curve

#convert to numeric

cv\_predict.rose<-as.numeric(cv\_predict.rose)

roc(data=valid.rose[,-c(1,2)],response=target,predictor=cv\_predict.rose,auc=TRUE,plot=TRUE)

#LightGBM:-LightGBM is a gradient boosting framework that uses tree based learning algorithms. We are going to use LightGBM model.

#Training and validation dataset

#Convert data frame to matrix

X\_train<-as.matrix(train.data[,-c(1,2)])

y\_train<-as.matrix(train.data$target)

X\_valid<-as.matrix(valid.data[,-c(1,2)])

y\_valid<-as.matrix(valid.data$target)

test\_data<-as.matrix(df\_test[,-c(1)])

#training data

lgb.train <- lgb.Dataset(data=X\_train, label=y\_train)

#Validation data

lgb.valid <- lgb.Dataset(data=X\_valid,label=y\_valid)

#Choosing best hyperparameters

#Selecting best hyperparameters

lgb.grid = list(objective = "binary",

metric = "auc",

boost='gbdt',

max\_depth=-1,

boost\_from\_average='false',

min\_sum\_hessian\_in\_leaf = 12,

feature\_fraction = 0.05,

bagging\_fraction = 0.5,

bagging\_freq = 5,

learning\_rate=0.02,

tree\_learner='serial',

num\_leaves=20,

num\_threads=4,

min\_data\_in\_bin=150,

min\_gain\_to\_split = 30,

min\_data\_in\_leaf = 90,

verbosity=-1,

is\_unbalance = TRUE)

#Training the lgbm model

lgbm.model <- lgb.train(params = lgb.grid, data = lgb.train, nrounds =10000,eval\_freq =1000,

valids=list(val1=lgb.train,val2=lgb.valid),early\_stopping\_rounds = 5000)

#lgbm model performance on test data

lgbm\_pred\_prob <- predict(lgbm.model,test\_data)

print(lgbm\_pred\_prob)

#Convert to binary output (1 and 0) with threshold 0.5

lgbm\_pred<-ifelse(lgbm\_pred\_prob>0.5,1,0)

print(lgbm\_pred)

#Let us plot the important features

#feature importance plot

tree\_imp <- lgb.importance(lgbm.model, percentage = TRUE)

lgb.plot.importance(tree\_imp, top\_n = 50, measure = "Frequency", left\_margin = 10)

#We tried model with logistic regression,ROSE and lightgbm. But,lightgbm is performing well on imbalanced data compared to other models based on scores of roc\_auc\_score.

#Final submission

sub\_df<-data.frame(ID\_code=df\_test$ID\_code,lgb\_predict\_prob=lgbm\_pred\_prob,lgb\_predict=lgbm\_pred)

write.csv(sub\_df,'submission-R.CSV',row.names=F)

head(sub\_df)

My python code :

#importing initial libraries

import pandas as pd

import numpy as np

import os

import warnings

warnings.filterwarnings('ignore')

os.getcwd() # checking the working directory

#loading files into the dataframe

df\_train = pd.read\_csv("train.csv")

df\_test = pd.read\_csv("test.csv")

#checking shape

print(df\_train.shape)

print(df\_test.shape)

df\_train.head()

df\_test.head()

#checking for missing values in the given dataset

missing\_train = df\_train.isnull().sum().sum()

print(missing\_train)

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

missing\_test = df\_test.isnull().sum().sum()

print(missing\_test)

# there are no missing values in both the dataset

# after checking the null values or missing values in the dataset

# I think we can go further with exploratory analysis

df\_train.describe()

df\_test.describe()

# observation here the min max is very close in the range

# the standard deviation is in the higher values.

# the mean range is in a higher range

#Target Class Count

Tgt\_cls=df\_train['target'].value\_counts()

print('Count of the target class :\n' , Tgt\_cls)

#Percentage of target class count

Per\_tgt\_cls=df\_train['target'].value\_counts()/len(df\_train)\*100

print('Percentage of target class count :\n', Per\_tgt\_cls)

# We are having an imbalanced data, where 89.951% of the data is no. of customers who will not make a transaction (0)

#& 10.049 % of the data are those who will make a transaction(1).

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

df\_train['target'].value\_counts().plot(kind = 'bar')

# the target dataset distribution is imbalance set as seen in the graph above.

# checking the distribution of the train attributes

def plot\_dist(t0,t1,label1,label2,train\_attributes):

i=0

sns.set\_style('darkgrid')

fig=plt.figure()

ax=plt.subplots(10,10,figsize=(22,18))

for att in train\_attributes :

i+=1

plt.subplot(10,10,i)

sns.distplot(t0[att],hist=False,label=label1)

sns.distplot(t1[att],hist=False,label=label2)

plt.legend()

plt.xlabel('Attribute',)

sns.set\_style("ticks",{"xtick.major.size": 8, "ytick.major.size": 8})

plt.show()

#Corresponding to negative class-

t0=df\_train[df\_train.target.values==0]

#Corresponding to possitive class-

t1=df\_train[df\_train.target.values==1]

#train attributes from 2 to 102 -

train\_attributes=df\_train.columns.values[2:102]

#trai attributes from 102 to 201

train\_attributes2 =df\_train.columns.values[102:]

#Plot distribution of train attributes-

plot\_dist(t0,t1,'0','1',train\_attributes)

plot\_dist(t0,t1,'0','1',train\_attributes2)

#We can observed that their is a considerable no. of features which are significantly have different distribution for two target variables.

#For example like var\_0,var\_1,var\_9,var\_198 var\_180 etc.

#We can observed that their is a considerable number of features which are significantly have same distribution for two target variables.

#For example like var\_3,var\_7,var\_10,var\_171,var\_185 etc.

#Distribution of test attributes-

def plot\_ta\_dist(test\_attributes):

i=0

sns.set\_style('darkgrid')

fig=plt.figure()

ax=plt.subplots(10,10,figsize=(22,18))

for attribute in test\_attributes:

i+=1

plt.subplot(10,10,i)

sns.distplot(df\_test[attribute],hist=False)

plt.xlabel('Attribute',)

sns.set\_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})

plt.show()

#test attribiutes from 1 to 200 in two parts -

test\_attributes=df\_test.columns.values[1:101]

test\_attributes1=df\_test.columns.values[101:]

#Plot distribution of test attributes -

plot\_ta\_dist(test\_attributes)

plot\_ta\_dist(test\_attributes1)

#Distribution of Mean Values per column in train & test dataset:-

plt.figure(figsize=(16,8))

#Train attributes-

train\_attributes=df\_train.columns.values[2:202]

#Test attributes-

test\_attributes=df\_test.columns.values[1:201]

#Distribution plot for mean values per column in train attributes:

sns.distplot(df\_train[train\_attributes].mean(axis=0),color='red',kde=True,bins=150,label='train')

#Distribution plot for mean values per column in test attributes:

sns.distplot(df\_test[test\_attributes].mean(axis=0),color='blue',kde=True,bins=150,label='test')

plt.title('Distribution of Mean Values per column in train & test dataset')

plt.legend()

plt.show()

#Distribution of Mean Values per column in train & test dataset:-

plt.figure(figsize=(16,8))

#Distribution plot for mean values per rows in train attributes:

sns.distplot(df\_train[train\_attributes].mean(axis=1),color='red',kde=True,bins=150,label='train')

#Distribution plot for mean values per rows in test attributes:

sns.distplot(df\_test[test\_attributes].mean(axis=1),color='blue',kde=True,bins=150,label='test')

plt.title('Distribution of Mean Values per row in train & test dataset')

plt.legend()

plt.show()

#Distribution of S.D Values per column in train & test dataset:-

plt.figure(figsize=(16,8))

#Train attributes-

train\_attributes=df\_train.columns.values[2:202]

#Test attributes-

test\_attributes=df\_test.columns.values[1:201]

#Distribution plot for S.D values per column in train attributes:

sns.distplot(df\_train[train\_attributes].std(axis=0),color='blue',kde=True,bins=150,label='train')

#Distribution plot for S.D values per column in test attributes:

sns.distplot(df\_test[test\_attributes].std(axis=0),color='green',kde=True,bins=150,label='test')

plt.title('Distribution of S.D Values per column in train & test dataset')

plt.legend()

plt.show()

#Distribution of S.D Values per column in train & test dataset:-

plt.figure(figsize=(16,8))

#Distribution plot for S.D values per rows in train attributes:

sns.distplot(df\_train[train\_attributes].std(axis=1),color='blue',kde=True,bins=150,label='train')

#Distribution plot for S.D values per rows in test attributes:

sns.distplot(df\_test[test\_attributes].std(axis=1),color='green',kde=True,bins=150,label='test')

plt.title('Distribution of S.D Values per row in train & test dataset')

plt.legend()

plt.show()

#Distribution of skew Values per column in train & test dataset:-

plt.figure(figsize=(16,8))

#Train attributes-

train\_attributes=df\_train.columns.values[2:202]

#Test attributes-

test\_attributes=df\_test.columns.values[1:201]

#Distribution plot for skew values per column in train attributes:

sns.distplot(df\_train[train\_attributes].skew(axis=0),color='red',kde=True,bins=150,label='train')

#Distribution plot for skew values per column in test attributes:

sns.distplot(df\_test[test\_attributes].skew(axis=0),color='green',kde=True,bins=150,label='test')

plt.title('Distribution of skewness Values per column in train & test dataset')

plt.legend()

plt.show()

#Distribution of skew Values per column in train & test dataset:-

plt.figure(figsize=(16,8))

#Distribution plot for skew values per rows in train attributes:

sns.distplot(df\_train[train\_attributes].skew(axis=1),color='red',kde=True,bins=150,label='train')

#Distribution plot for skew values per rows in test attributes:

sns.distplot(df\_test[test\_attributes].skew(axis=1),color='green',kde=True,bins=150,label='test')

plt.title('Distribution of skewness Values per row in train & test dataset')

plt.legend()

plt.show()

#Distribution of kurtosis Values per column in train & test dataset:-

plt.figure(figsize=(16,8))

#Train attributes-

train\_attributes=df\_train.columns.values[2:202]

#Test attributes-

test\_attributes=df\_test.columns.values[1:201]

#Distribution plot for kurtosis values per column in train attributes:

sns.distplot(df\_train[train\_attributes].kurtosis(axis=0),color='red',kde=True,bins=150,label='train')

#Distribution plot for kurtosis values per column in test attributes:

sns.distplot(df\_test[test\_attributes].kurtosis(axis=0),color='blue',kde=True,bins=150,label='test')

plt.title('Distribution of kurtosis Values per column in train & test dataset')

plt.legend()

plt.show()

#Distribution of kurtosis Values per column in train & test dataset:-

plt.figure(figsize=(16,8))

#Distribution plot for kurtosis values per rows in train attributes:

sns.distplot(df\_train[train\_attributes].kurtosis(axis=1),color='red',kde=True,bins=150,label='train')

#Distribution plot for kurtosis values per rows in test attributes:

sns.distplot(df\_test[test\_attributes].kurtosis(axis=1),color='green',kde=True,bins=150,label='test')

plt.title('Distribution of kurtosis Values per row in train & test dataset')

plt.legend()

plt.show()

# finding the correlation in the train attributes

train\_attribute\_correlation = df\_train.columns.values[2:202]

train\_correlation = df\_train[train\_attribute\_correlation].corr()

train\_correlation.head()

#Its visible that correlation between train attributes is very small.

#finding the correlation in the test attributes

test\_attribute\_correlation = df\_test.columns.values[2:]

test\_correlation = df\_test[test\_attribute\_correlation].corr()

test\_correlation.head()

#Its visible that correlation between test attributes is very small.

train\_correlation=df\_train[train\_attributes].corr()

train\_correlation=train\_correlation.values.flatten()

train\_correlation=train\_correlation[train\_correlation!=1]

test\_correlation=df\_test[test\_attributes].corr()

test\_correlation=test\_correlation.values.flatten()

test\_correlation=test\_correlation[test\_correlation!=1]

plt.figure(figsize=(20,5))

sns.distplot(train\_correlation,color="blue",label="train")

sns.distplot(test\_correlation,color="red",label="test")

plt.xlabel("Correlation values found in train & test data")

plt.ylabel("Density")

plt.title ("Correlation values in train & test data")

plt.legend()

#The correlation between the train and test data is very small, its completely visible from the above graph

#Feature engineering: Permutation Importance

#training data & test data

X = df\_train.drop(columns=['ID\_code','target'],axis=1) # dropping the target values and the ID code as id is not needed and target values is a dependent variable

test = df\_test.drop(columns=['ID\_code'],axis = 1)

y = df\_train['target']

print(X.shape)

print(y.shape)

print(X)

print(y)

# Building a model to find out which features are important based on the weights value returned

from sklearn.model\_selection import train\_test\_split,cross\_val\_predict,cross\_val\_score

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import StratifiedKFold

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix,roc\_auc\_score,roc\_curve,classification\_report,auc

#Split the train data:-

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,random\_state=10)

rf\_model=RandomForestClassifier(n\_estimators=7,random\_state=10)

#fitting the model:-

rf\_model.fit(X\_test,y\_test)

#Calucating weights & Observing some Important features via using eli5 library-

#ELI5 is a Python package which helps to debug machine learning classifiers and explain their predictions.

#Permutation Importance:-

#pip install "eli5"

import eli5

from eli5.sklearn import PermutationImportance

perm\_imp=PermutationImportance(rf\_model,random\_state=10)

#fitting the model:-

perm\_imp.fit(X\_test,y\_test)

#Important Features:-

eli5.show\_weights(perm\_imp,feature\_names=X\_test.columns.tolist(),top=200)

#Importance of features is decreasing as we move down the top of column.

#Fetaures showing in green indicates they are having positive impact on our prediction.

#Features showing in white showing they have no impact on prediction.

#Most imprtant feature is var\_81.

#Handling of imblanced data:- We are going to use multiple approaches for dealing with imbalanced datasets.

#Change of performance matrix.

#Oversample minority class.

#Undersample majority class.

#SMOTE ( Synthetic Minority Oversampling technique)

#Change of algorithms.

#Logistic Regression Model:-

#Spliting the data via Sratified KFold Cross Validator:-

#Training Data:

X=df\_train.drop(['ID\_code','target'],axis=1)

Y=df\_train['target']

#Stratified KFold Cross Validator:-

skf=StratifiedKFold(n\_splits=5, random\_state=42, shuffle=True)

for train\_index, valid\_index in skf.split(X,Y):

X\_train, X\_valid = X.iloc[train\_index], X.iloc[valid\_index]

y\_train, y\_valid = Y.iloc[train\_index], Y.iloc[valid\_index]

print('Shape of X\_train :',X\_train.shape)

print('Shape of X\_valid :',X\_valid.shape)

print('Shape of y\_train :',y\_train.shape)

print('Shape of y\_valid :',y\_valid.shape)

# Logistic Regression Model:-

lr\_model=LogisticRegression(random\_state=42)

#fitting the model-

lr\_model.fit(X\_train,y\_train)

#Accuracy of model-

lr\_score=lr\_model.score(X\_train,y\_train)

print('Accuracy of lr\_model :',lr\_score)

#accuracy score : 0.91

#Cross validation prediction of lr\_model-

cv\_predict=cross\_val\_predict(lr\_model,X\_valid,y\_valid,cv=5)

#Cross validation score-

cv\_score=cross\_val\_score(lr\_model,X\_valid,y\_valid,cv=5)

print('cross val score :',np.average(cv\_score))

cross val score : 0.91045

#Accuracy of the model is not the best metric to use while evaluating the imbalanced datasets as it may be misleading.

#We are going to change the performance metric.

#Confusion matrix:-

cm=confusion\_matrix(y\_valid,cv\_predict)

cm=pd.crosstab(y\_valid,cv\_predict)

cm

#ROC\_AUC SCORE:-

roc\_score=roc\_auc\_score(y\_valid,cv\_predict)

print('ROC Score:',roc\_score)

# Roc score is 0.61

#ROC\_AUC\_Curve:-

plt.figure()

false\_positive\_rate,recall,thresholds=roc\_curve(y\_valid,cv\_predict)

roc\_auc=auc(false\_positive\_rate,recall)

plt.title('Reciver Operating Characteristics(ROC)')

plt.plot(false\_positive\_rate,recall,'b',label='ROC(area=%0.3f)' %roc\_auc)

plt.legend()

plt.plot([0,1],[0,1],'r--')

plt.xlim([0.0,1.0])

plt.ylim([0.0,1.0])

plt.ylabel('Recall(True Positive Rate)')

plt.xlabel('False Positive Rate')

plt.show()

print('AUC:',roc\_auc)

#AUC score is 0.61

#On comparing roc\_auc\_score and model accuracy, model is not performing well on imbalanced data.

#Classification report:-

classification\_scores=classification\_report(y\_valid,cv\_predict)

print(classification\_scores)

#As we see that f1 score is high for the customers who will not make a transaction, compare to those who will make a transaction.

#So, we are going to change the algorithm.

#Model performance on test data:-

X\_test=df\_test.drop(['ID\_code'],axis=1)

lr\_pred=lr\_model.predict(X\_test)

print(lr\_pred)

'''Oversample Minority Class:-

-Adding more copies of minority class.

-It can be a good option we dont have that much large data to work.

-Drawback of this process is we are adding info. That can lead to overfitting or poor performance on test data.

Undersample Mojorityclass:-

-Removing some copies of majority class.

-It can be a good option if we have very large amount of data say in millions to work.

-Drawback of this process is we are removing some valuable info. that can leads to underfitting & poor performance on test data.

As per the drawbacks of both the model we will use SMOTE ( Synthetic Minority Oversampling technique) that is more best than the above as compare to above one's.

SMOTE ( Synthetic Minority Oversampling technique):- This is a statistical technique for increasing the number of cases in your dataset in a balanced way. It uses a nearest neighbors algorithm to generate new and synthetic data to used for training the model.'''

#pip install imblearn

#pip install -U imbalanced-learn –user

from imblearn.over\_sampling import SMOTE

#SMOTE:-

sm = SMOTE(random\_state=42, sampling\_strategy=1.0)

#Generating synthetic data points

X\_smote,y\_smote=sm.fit\_sample(X\_train,y\_train)

X\_smote\_v,y\_smote\_v=sm.fit\_sample(X\_valid,y\_valid)

#Logistic regression model for SMOTE:-

smote=LogisticRegression(random\_state=10)

#fitting the smote model:-

smote.fit(X\_smote,y\_smote)

#Accuracy of the model:-

smote\_score=smote.score(X\_smote,y\_smote)

print(smote\_score)

# the accuracy of the model is 0.78

#Cross validation prediction for SMOTE:-

cv\_pred=cross\_val\_predict(smote,X\_smote\_v,y\_smote\_v,cv=5)

#Cross validation score:-

cv\_score=cross\_val\_score(smote,X\_smote\_v,y\_smote\_v,cv=5)

print('Cross validation score :',np.average(cv\_score))

Cross validation score : 0.7970539188438022

#Confusion matrix:-

cm=confusion\_matrix(y\_smote\_v,cv\_pred)

cm=pd.crosstab(y\_smote\_v,cv\_pred)

cm

#ROC\_AUC SCORE:-

roc\_score=roc\_auc\_score(y\_smote\_v,cv\_pred)

print('ROC score:',roc\_score)

#ROC\_AUC SCORE:-

roc\_score=roc\_auc\_score(y\_smote\_v,cv\_pred)

print('ROC score:',roc\_score)

ROC score: 0.7970539188438022

#ROC\_AUC Curve:-

plt.figure()

false\_positive\_rate,recall,thresholds=roc\_curve(y\_smote\_v,cv\_pred)

roc\_auc=auc(false\_positive\_rate,recall)

plt.title('Reciver Operating Characteristics(ROC)')

plt.plot(false\_positive\_rate,recall,'b',label='ROC(area=%0.3f)' %roc\_auc)

plt.legend()

plt.plot([0,1],[0,1],'r--')

plt.xlim([0.0,1.0])

plt.ylim([0.0,1.0])

plt.ylabel('Recall(True Positive Rate)')

plt.xlabel('False Positive Rate')

plt.show()

print('AUC:',roc\_auc)

#Classification Report:-

scores=classification\_report(y\_smote\_v,cv\_pred)

print(scores)

#Observation:- As we see that f1 score is high for the customers who will not make a transaction,

#as well as who will make a transaction.

#Model\_performance on test data:-

#Predicting the model-

X\_test=df\_test.drop(['ID\_code'],axis=1)

smote\_pred=smote.predict(X\_test)

print(smote\_pred)

# What I observed was that the model is performing well on imbalance data as compare to logistic regression.

#Light GBM:- It is a gradient boosting framework that uses tree based learning algorithm.

#pip install lightgbm

import lightgbm as lgb

#Training data-

lgb\_train=lgb.Dataset(X\_train,label=y\_train)

#Validation data-

lgb\_valid=lgb.Dataset(X\_valid,label=y\_valid)

#Selecting best hyperparameters by tuning of different parameters:-

params={'boosting\_type': 'gbdt',

'max\_depth' : -1, #no limit for max\_depth if <0

'objective': 'binary',

'boost\_from\_average':False,

'nthread': 20,

'metric':'auc',

'num\_leaves': 50,

'learning\_rate': 0.01,

'max\_bin': 100, #default 255

'subsample\_for\_bin': 100,

'subsample': 1,

'subsample\_freq': 1,

'colsample\_bytree': 0.8,

'bagging\_fraction':0.5,

'bagging\_freq':5,

'feature\_fraction':0.08,

'min\_split\_gain': 0.45, #>0

'min\_child\_weight': 1,

'min\_child\_samples': 5,

'is\_unbalance':True,

}

#Training lgbm model:-

num\_rounds= 1000

lgbm= lgb.train(params,lgb\_train,num\_rounds,valid\_sets=[lgb\_train,lgb\_valid],verbose\_eval=1000,early\_stopping\_rounds = 500)

lgbm

[LightGBM] [Warning] feature\_fraction is set=0.08, colsample\_bytree=0.8 will be ignored. Current value: feature\_fraction=0.08

[LightGBM] [Warning] bagging\_freq is set=5, subsample\_freq=1 will be ignored. Current value: bagging\_freq=5

[LightGBM] [Warning] bagging\_fraction is set=0.5, subsample=1 will be ignored. Current value: bagging\_fraction=0.5

[LightGBM] [Warning] feature\_fraction is set=0.08, colsample\_bytree=0.8 will be ignored. Current value: feature\_fraction=0.08

[LightGBM] [Warning] bagging\_freq is set=5, subsample\_freq=1 will be ignored. Current value: bagging\_freq=5

[LightGBM] [Warning] bagging\_fraction is set=0.5, subsample=1 will be ignored. Current value: bagging\_fraction=0.5

[LightGBM] [Warning] feature\_fraction is set=0.08, colsample\_bytree=0.8 will be ignored. Current value: feature\_fraction=0.08

[LightGBM] [Warning] bagging\_freq is set=5, subsample\_freq=1 will be ignored. Current value: bagging\_freq=5

[LightGBM] [Warning] bagging\_fraction is set=0.5, subsample=1 will be ignored. Current value: bagging\_fraction=0.5

[LightGBM] [Warning] feature\_fraction is set=0.08, colsample\_bytree=0.8 will be ignored. Current value: feature\_fraction=0.08

[LightGBM] [Warning] bagging\_freq is set=5, subsample\_freq=1 will be ignored. Current value: bagging\_freq=5

[LightGBM] [Warning] bagging\_fraction is set=0.5, subsample=1 will be ignored. Current value: bagging\_fraction=0.5

[LightGBM] [Warning] feature\_fraction is set=0.08, colsample\_bytree=0.8 will be ignored. Current value: feature\_fraction=0.08

[LightGBM] [Warning] bagging\_freq is set=5, subsample\_freq=1 will be ignored. Current value: bagging\_freq=5

[LightGBM] [Warning] bagging\_fraction is set=0.5, subsample=1 will be ignored. Current value: bagging\_fraction=0.5

[LightGBM] [Info] Number of positive: 16078, number of negative: 143922

[LightGBM] [Warning] Auto-choosing col-wise multi-threading, the overhead of testing was 0.141330 seconds.

You can set `force\_col\_wise=true` to remove the overhead.

[LightGBM] [Info] Total Bins 6856

[LightGBM] [Info] Number of data points in the train set: 160000, number of used features: 200

Training until validation scores don't improve for 500 rounds

[1000] training's auc: 0.930752 valid\_1's auc: 0.873942

Did not meet early stopping. Best iteration is:

[1000] training's auc: 0.930752 valid\_1's auc: 0.873942

#Did not meet early stopping. Best iteration is:

#[1000] training's auc: 0.930752 valid\_1's auc: 0.873942

#LGBM model performance on test data

X\_test=df\_test.drop(['ID\_code'],axis=1)

#Predict the model:-

#probability predictions

lgbm\_predict\_prob=lgbm.predict(X\_test,random\_state=42,num\_iteration=lgbm.best\_iteration)

#Convert to binary output 1 or 0

lgbm\_predict=np.where(lgbm\_predict\_prob>=0.5,1,0)

print(lgbm\_predict\_prob)

print(lgbm\_predict)

#Plotting of important Features:-

lgb.plot\_importance(lgbm,max\_num\_features=50,importance\_type="split",figsize=(20,50))

#Conclusion:- We have tried with diff model like Logistic regression,smote & lightgbm.

#But we observed that light gbm is performing well on imbalanced data as compare to other models based on the roc\_auc scores.

df\_sub=pd.DataFrame({'ID\_code':df\_test['ID\_code'].values})

df\_sub['lgbm\_predict\_prob']=lgbm\_predict\_prob

df\_sub['lgbm\_predict']=lgbm\_predict

df\_sub.to\_csv('submission\_python.csv',index=False)

df\_sub.head()

References :

<https://www.kaggle.com/learn/feature-engineering>

<https://www.kaggle.com/matleonard/feature-generation>

<https://www.kaggle.com/dansbecker/permutation-importance>

<https://towardsdatascience.com/model-based-feature-importance-d4f6fb2ad403>

<https://www.youtube.com/watch?v=NPdn3YPkg9w>

<https://www.youtube.com/user/krishnaik06/about>

<https://www.youtube.com/c/joshstarmer/about>