**Santander Customer Transaction Prediction**

**Submitted by Kaushik Mazumdar**

**Contents**

1. Introduction 3
   1. [Problem](#_bookmark0) Statement 3
   2. [Data 3](#_bookmark1)
2. Methodology 4
   1. Exploratory Data Analysis 4
      1. [Missing](#_bookmark2) value analysis 5
      2. Attributes Distributions and trends 6
      3. Outlier Analysis 13
      4. [Feature](#_bookmark3) Selection 13
      5. Feature Engineering 14
   2. Modeling 18
      1. Model Selection 18
      2. Logistic Regression 18
      3. SMOTE or ROSE 22
      4. LightGBM 25
3. Conclusion 28
   1. [Model Evaluation 28](#_TOC_250004)
      1. Confusion Matrix 28
      2. ROC\_AUC\_score 19
   2. [Model Selection 38](#_TOC_250003)

[Appendix A - Extra Figures 39](#_TOC_250002)

Appendix B – Complete Python and R Code 51

Python Code 52

[R code 62](#_TOC_250001)

[References 71](#_TOC_250000)

**Chapter 1**

**Introduction**

## Problem Statement

At Santander, mission is to help people and businesses prosper. We are always looking for ways to help our customers understand their financial health and identify which products and services might help them achieve their monetary goals.

Our data science team is continually challenging our machine learning algorithms, working with the global data science community to make sure we can more accurately identify new ways to solve our most common challenge, binary classification problems such as: is a customer satisfied? Will a customer buy this product? Can a customer pay this loan?

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

## Data

In this project, our task is to build classification models which would be used to predict which customers will make a specific transaction in the future. Given below is a sample of the Santander customer transaction dataset:

Table 1.1: Train dataset (Columns:1-202)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID\_code** | **target** | **var\_0** | **var\_1** | **var\_2** | **………** | **……….** | **var\_199** |
| **train\_01** | 0 | 8.92 | -6.78 | 11.90 | …..... | …….. | -1.09 |
| **train\_02** | 0 | 11.5 | -4.14 | 13.85 | ……... | ...…… | 1.95 |
| **train\_03** | 0 | 8.60 | -2.74 | 12.08 | ...…… | ...…… | 0.39 |
| **train\_04** | 0 | 11.06 | -2.15 | 8.95 | ...…… | ...…… | -8.99 |
| **train\_05** | 0 | 9.83 | -1.48 | 12.87 | ...…… | ...…… | -8.81 |

Table 1.2: Test Dataset (Columns: 1-201)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID\_code** | **var\_0** | **var\_1** | **var\_2** | **var\_3** | **………** | **……….** | **var\_199** |
| **test\_01** | 11.06 | 8.92 | -6.78 | 11.90 | …..... | …….. | -1.09 |
| **test\_02** | 8.53 | 11.5 | -4.14 | 13.85 | ……... | ...…… | 1.95 |
| **test\_03** | 5.48 | 8.60 | -2.74 | 12.08 | ...…… | ...…… | 0.39 |
| **test\_04** | 8.53 | 11.06 | -2.15 | 8.95 | ...…… | ...…… | -8.99 |
| **test\_05** | 11.7 | 9.83 | -1.48 | 12.87 | ...…… | ...…… | -8.81 |

From the table below we have the following 200 variables, using which we have to predict which customer will make a specific transaction in future :

Table 1.3: Predictor Variables

|  |  |
| --- | --- |
| **SL.No.** | **Predictor** |
| **1** | ID-code |
| **2** | var0 |
| **3** | var1 |
| **4** | var2 |
| **5** | var3 |
| **6** | var4 |
| **7** | var5 |
| **.…** | .. … |
|  | ….. |
| **….** | …… |
| **….** | …… |
| **….** | ……. |
| **….** | ……. |
| **….** | ……. |
| **….** | …….. |
| **202** | var199 |

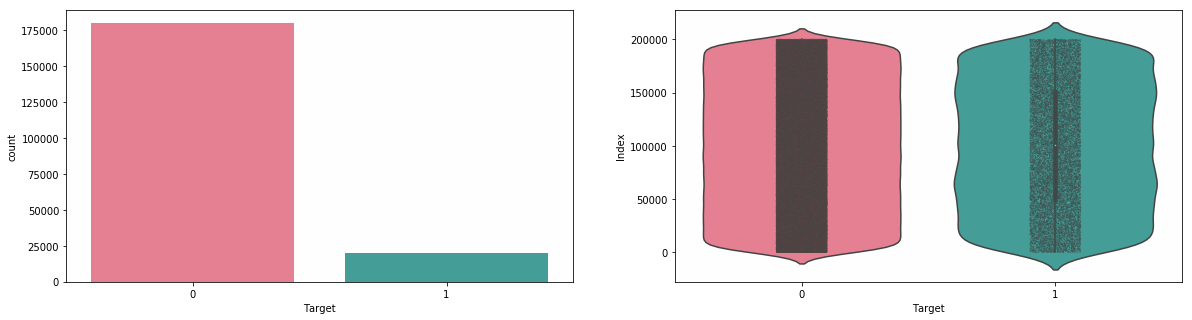
**Chapter 2**

**Methodology**

### 2.1 Exploratory Data Analysis (EDA)

Exploratory data analysis is one of the most important step in data mining in order to know features of data. It involves the loading dataset, target classes count, data cleaning, typecasting of attributes, missing value analysis, Attributes distributions and trends. So, we have to clean the data otherwise it will effect on performance of the model. Now we are going to explain one by one as follows. In this EDA I explained with seaborn visualizations.

* + 1. **Target classes count**



**Take away:**

* We have an unbalanced data, where 90% of the data is the number of customers those will not make a transaction and 10% of the data is those who will make a transaction.
* Look at the violin plots seems that there are no relationship between the targets with the index of the train data frame. This is more dominated by the zero targets then for the ones.
* Look at the jitter plots with violin plots. We can observe that targets looks uniformly distributed over the index of the data frame.
  + 1. **Missing value 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.

R and Python code as follows,

#Missing values in train and test data

# R code

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

[1] 0

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

[1] 0

# Python code

train\_missing=train\_df.isnull().sum().sum()

[1] 0 test\_missing=test\_df.isnull().sum().sum()

[1] 0

* + 1. **Attributes distributions and trends**

**Distribution of train attributes**

Let us look distribution of train attributes from var\_0 to var\_99

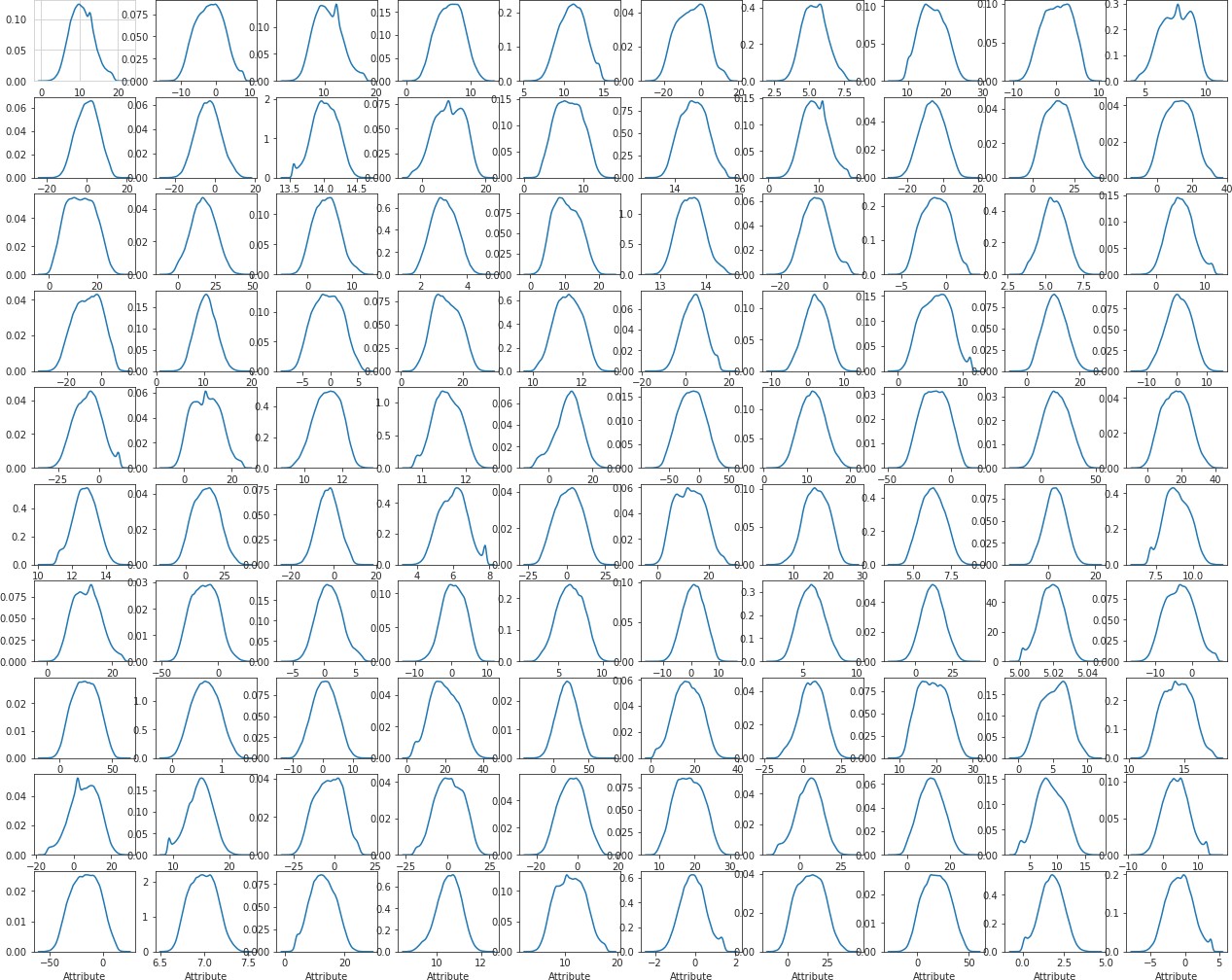


**Take away:**

* We can observe that there is a considerable number of features which are significantly have different distributions for two target variables. For example like var\_0,var\_1,var\_9,var\_19,var\_18 etc.
* We can observe that there is a considerable number of features which are significantly have same distributions for two target variables. For example like var\_3, var\_7,var\_10,var\_17,var\_35 etc.

**Distribution of test attributes**

Let us look distribution of test attributes from var\_0 to var\_99

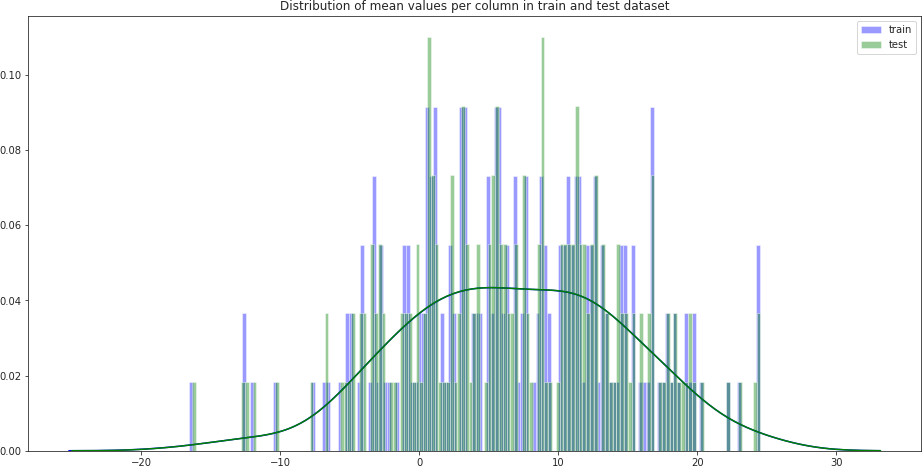


**Take away:**

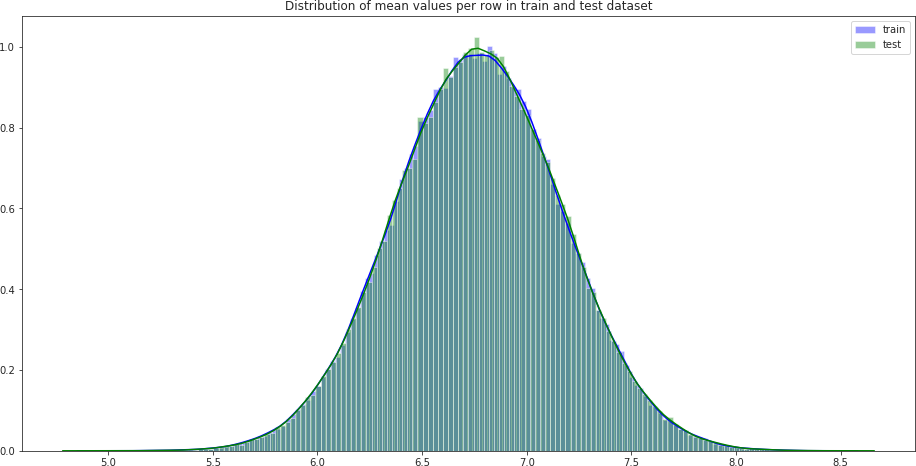
* We can observe that there is a considerable number of features which are significantly have different distributions. For example like var\_0, var\_1, var\_9, var\_18 var\_38 etc.
* We can observe that there is a considerable number of features which are significantly have same distributions. For example like var\_3, var\_7, var\_10, var\_17, var\_45, var\_192 etc.

**Distribution of mean values in both train and test dataset**

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

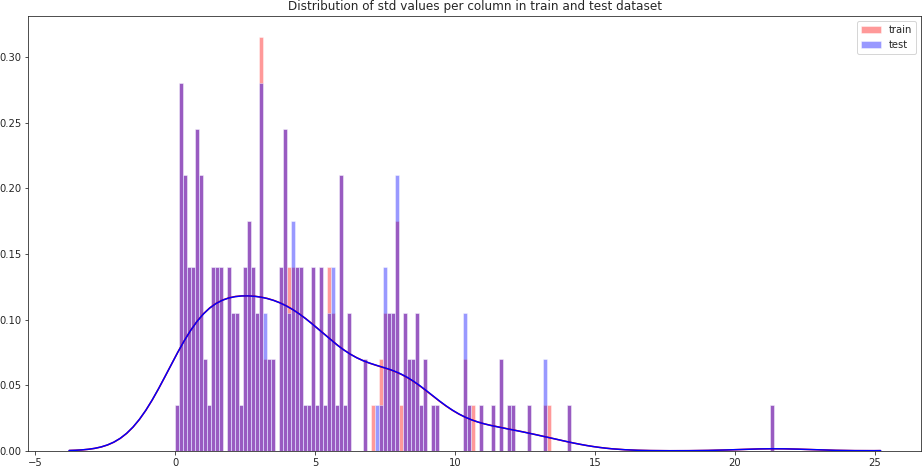


Let us look distribution of mean values per row in train and test dataset

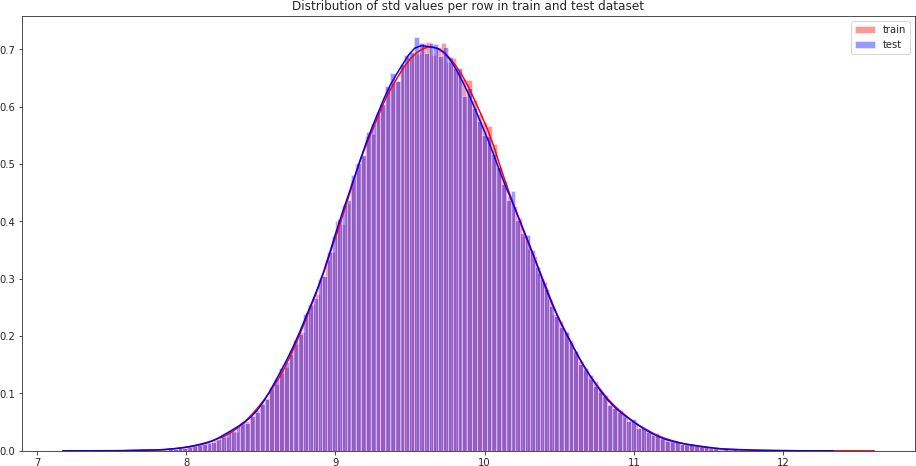


**Distribution of standard deviation (std) values in train and test dataset**

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

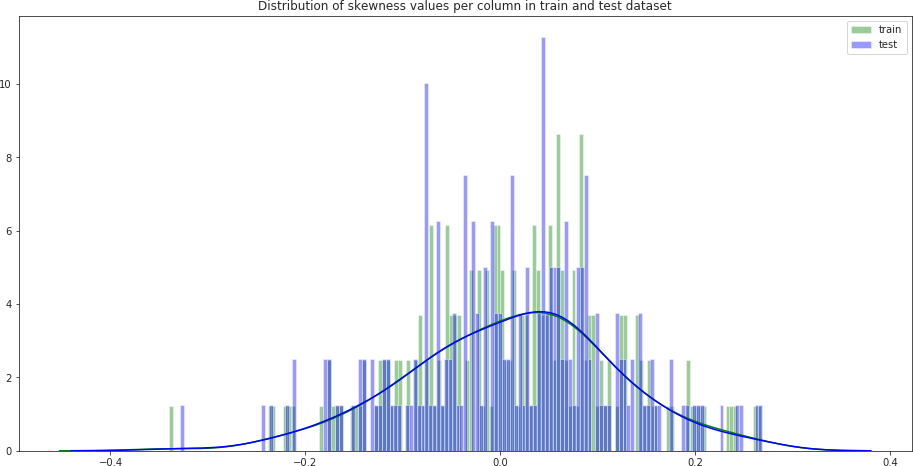


Let us look distribution of standard deviation (std) values per row in train and test dataset

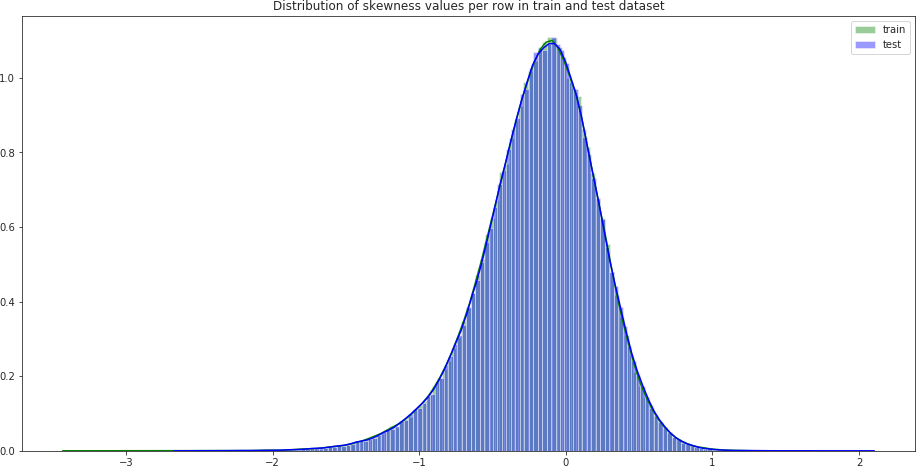


**Distribution of skewness values in train and test dataset**

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

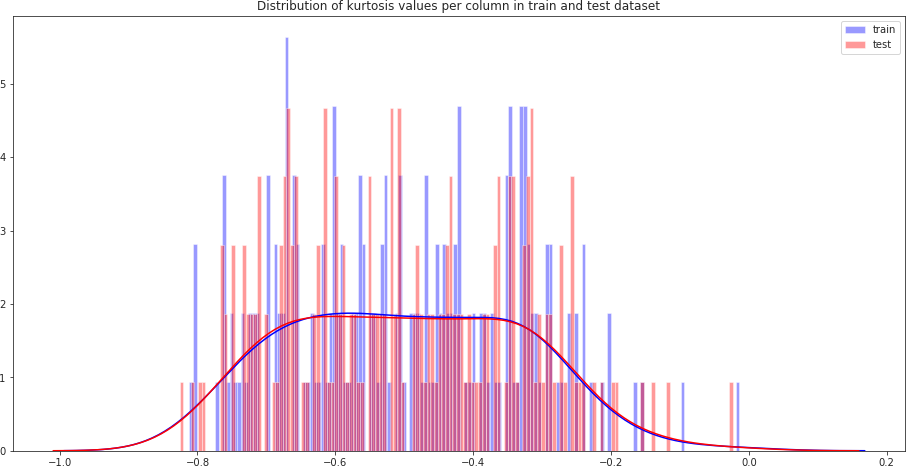


Let us look distribution of skewness per row in train and test dataset

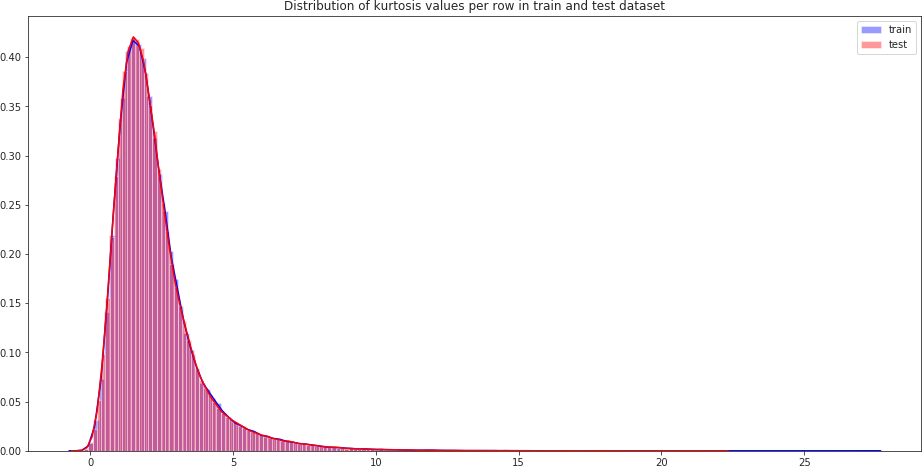


**Distribution of kurtosis values in train and test dataset**

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



* + 1. **Outlier analysis**

In this project, we haven’t perform outlier analysis due to the data is imbalanced and also not required for imbalanced data.

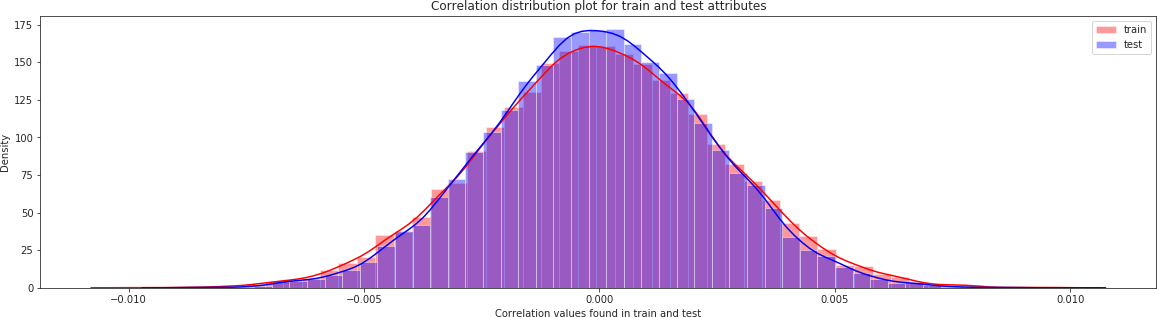
* + 1. **Feature Selection**

Feature selection is very important for modeling the dataset. The 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 is 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.



* + 1. **Feature engineering**

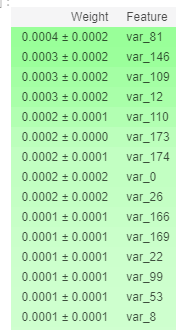
Let us do some feature engineering by using

* Permutation importance
* Partial dependence plots

**Permutation importance**

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

**Python code**



#training data X=train\_df.drop(columns=['ID\_code','target'],axis=1) test=test\_df.drop(columns=['ID\_code'],axis=1) y=train\_df['target']

#Split the training data X\_train,X\_valid,y\_train,y\_valid=train\_test\_split(X,y,random\_state=42)

#Random forest classifier rf\_model=RandomForestClassifier(n\_estimators=10,random\_state=42) #fitting the model

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

#Let us calculate weights and show important features using eli5 library. from eli5.sklearn import PermutationImportance perm\_imp=PermutationImportance(rf\_model,random\_state=42)

#fitting the model perm\_imp.fit(X\_valid,y\_valid)

#Important features eli5.show\_weights(perm\_imp,feature\_names=X\_valid.columns.tolist(),top=200)

**R code**

#Split the training data

train\_index<-sample(1:nrow(train\_df),0.75\*nrow(train\_df)) train\_data<-train\_df[train\_index,]

valid\_data<-train\_df[-train\_index,]

#Training the Random forest classifier set.seed(2732)

train\_data$target<-as.factor(train\_data$target) mtry<-floor(sqrt(200))

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

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

#Variable importance

VarImp<-importance(rf,type=2) VarImp

**Variable importance based on Mean Decrease Gini**



**Take away:**

* + We can observe that the top important features are var\_12, var\_26, var\_22, var\_174, var\_198 and so on based on Mean decrease Gini.

**Partial dependence plots**

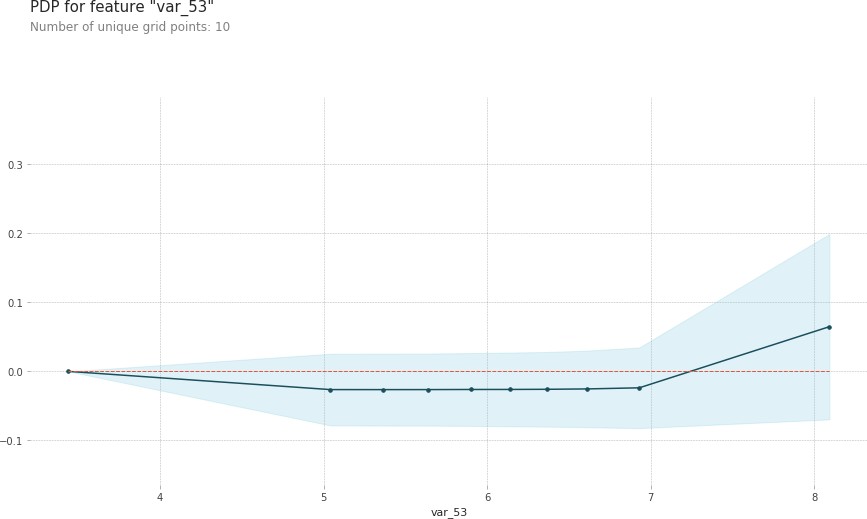
Partial dependence plot gives a graphical depiction of the marginal effect of a variable on the class probability or classification. While feature importance shows what variables most affect predictions, but partial dependence plots show how a feature affects predictions.

**Python code**

#Create the data we will plot 'var\_53'

features=[v for v in X\_valid.columns if v not in ['ID\_code','target']] pdp\_data=pdp.pdp\_isolate(rf\_model,dataset=X\_valid,model\_features=features,feature= 'var\_53')

#plot feature "var\_53" pdp.pdp\_plot(pdp\_data,'var\_53') plt.show()



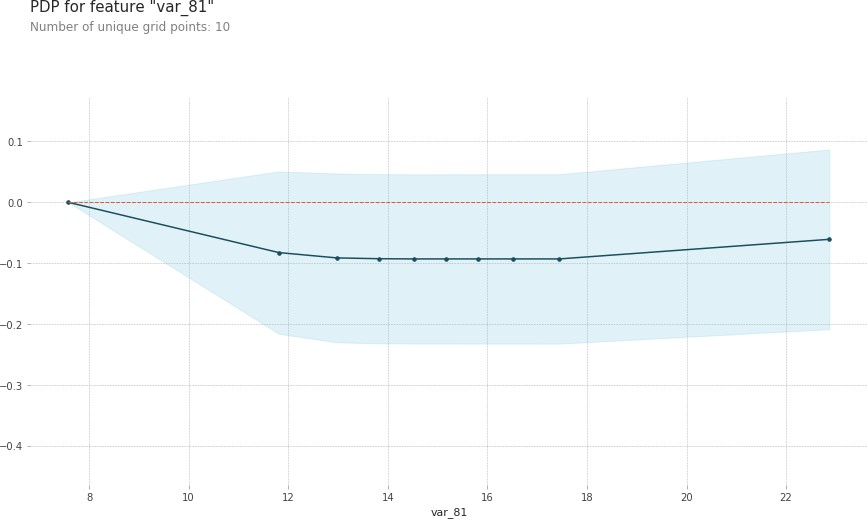
**Take away:**

* The y\_axis does not show the predictor value instead how the value changing with the change in given predictor variable.
* The blue shaded area indicates the level of confidence of 'var\_53'.
* On y-axis having a negative value means for that particular value of predictor variable it is less likely to predict the correct class and having a positive value means it has positive impact on predicting the correct class.

#Create the data we will plot 'var\_81'

features=[v for v in X\_valid.columns if v not in ['ID\_code','target']] pdp\_data=pdp.pdp\_isolate(rf\_model,dataset=X\_valid,model\_features=features,feature= 'var\_81')

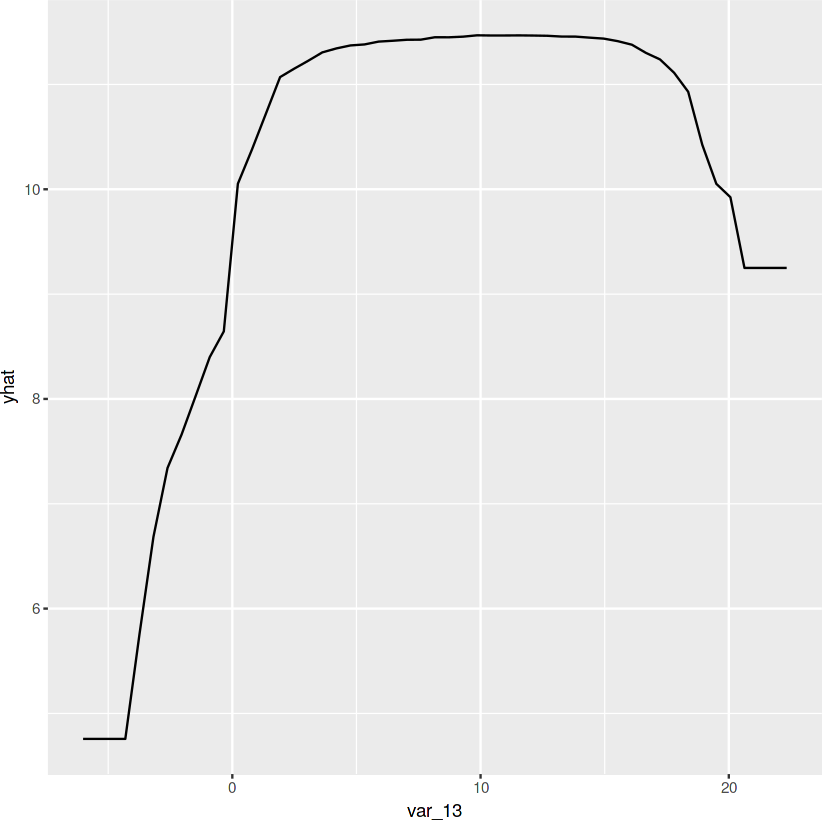
#plot feature "var\_81" pdp.pdp\_plot(pdp\_data,'var\_81') plt.show()



**Take away:**

* The y\_axis does not show the predictor value instead how the value changing with the change in given predictor variable.
* The blue shaded area indicates the level of confidence of 'var\_81'.
* On y-axis having a negative value means for that particular value of predictor variable it is less likely to predict the correct class and having a positive value means it has positive impact on predicting the correct class.

**R code**



#We will plot "var\_13"

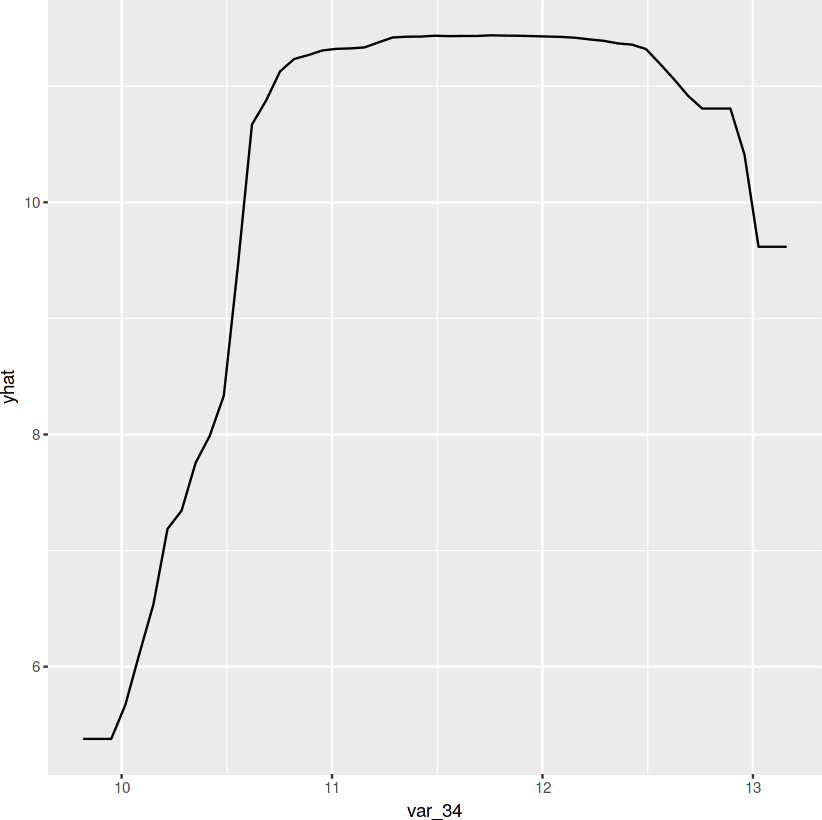
par.var\_13 <- partial(rf, pred.var = c("var\_13"), chull = TRUE) plot.var\_13 <- autoplot(par.var\_13, contour = TRUE)

**Take away:**

* The y\_axis does not show the predictor value instead how the value changing with the change in given predictor variable.
* The blue shaded area indicates the level of confidence of 'var\_13'.
* On y-axis having a positive value means for that particular value of predictor variable it is less likely to predict the correct class and having a positive value means it has positive impact on predicting the correct class.

#We will plot "var\_34"

par.var\_34 <- partial(rf, pred.var = c("var\_34"), chull = TRUE) plot.var\_34 <- autoplot(par.var\_34, contour = TRUE)



**Take away:**

* The y\_axis does not show the predictor value instead how the value changing with the change in given predictor variable.
* The blue shaded area indicates the level of confidence of 'var\_34'.
* On y-axis having a positive value means for that particular value of predictor variable it is less likely to predict the correct class and having a positive value means it has positive impact on predicting the correct class.
  1. **Modeling**
     1. **Model Selection**

After all early stages of preprocessing, 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.

* + 1. **Logistic Regression**

We will use a Logistic Regression to predict the values of our target variable.

**Python code**

#Training dataset for modelling

#Training data X=train\_df.drop(['ID\_code','target'],axis=1) Y=train\_df['target']

#StratifiedKFold cross validator

cv=StratifiedKFold(n\_splits=5,random\_state=42,shuffle=True) for train\_index,valid\_index **in** cv.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]

#Logistic regression model

lr\_model=LogisticRegression(random\_state=42) #fitting the lr model lr\_model.fit(X\_train,y\_train)

#Accuracy of the model lr=lr\_model.score(X\_train,y\_train)

Accuracy of the model : **0.914**

#Cross validation prediction 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 : 0.9132

#Predicting the model X\_test=test\_df.drop(['ID\_code'],axis=1) lr\_pred=lr\_model.predict(X\_test)

**R code**

Glmnet is a package that fits a generalized linear model via penalized maximum likelihood.

#Split the data using CreateDataPartition

train.index<-createDataPartition(train\_df$target,p=0.8,list=FALSE) train.data<-train\_df[train.index,]

valid.data<-train\_df[-train.index,]

#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(test\_df[,-c(1)])

#Logistic regression model set.seed(667)

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

#Cross validation prediction

set.seed(8909)

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

#Model performance on validation dataset set.seed(5363)

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

#Confusion matrix set.seed(689)

#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)

#ROC\_AUC score and curve set.seed(892)

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)

#predict the model

lr\_pred<-predict(lr\_model,test\_df[,-c(1)],type='class')

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:**

* It can be defined as adding more copies of minority class.
* It can be a good choice when we don't have a ton of data to work with.
* Drawback is that we are adding information. This may leads to overfitting and poor performance on test data.

**Under sample majority class:**

* It can be defined as removing some observations of the majority class.
* It can be a good choice when we have a ton of data -think millions of rows.
* Drawback is that we are removing information that may be valuable. This may leads to under fitting and poor performance on test data.

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

**Synthetic Minority Oversampling Technique (SMOTE)**

SMOTE uses a nearest neighbor’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.

**Python code**

from imblearn.over\_sampling import SMOTE #Synthetic Minority Oversampling Technique sm = SMOTE(random\_state=42, ratio=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=42) #fitting the smote model smote.fit(X\_smote,y\_smote)

smote\_score=smote.score(X\_smote,y\_smote) print('Accuracy of the smote\_model :',smote\_score)

Accuracy of the model : 0.798 #Cross validation prediction

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\_val\_score :',np.average(cv\_score))

cross\_val\_score : 0.80 #Predicting the model

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

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

**R code**

**Random Oversampling Examples (ROSE)**

It creates a sample of synthetic data by enlarging the features space of minority and majority class examples. In order to balance imbalanced data we are going to use SMOTE sampling method.

#Random Oversampling Examples(ROSE) set.seed(699)

#train.data$target<-as.factor(train.data$target)

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

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

#Baseline logistic regression model set.seed(462)

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

summary(lr\_rose)

#Cross validation prediction set.seed(473)

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

#Minimum lambda cv\_rose$lambda.min

#plot the auc score vs log(lambda) plot(cv\_rose)

#Model performance on validation dataset set.seed(442)

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

cv\_predict.rose

#Confusion matrix set.seed(478)

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

#ROC\_AUC score and curve set.seed(843)

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)

#predict the model set.seed(6543)

rose\_pred<-predict(lr\_rose,test\_df[,-c(1)],type='class')

**LightGBM**

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

**Python code**

Let us build LightGBM model

#Training the model #training data

lgb\_train=lgb.Dataset(X\_train,label=y\_train) #validation data lgb\_valid=lgb.Dataset(X\_valid,label=y\_valid)

#Selecting best hyper parameters 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': 8, 'metric':'auc', 'num\_leaves': 100,

'learning\_rate': 0.03,

'max\_bin': 950, #default 255

'subsample\_for\_bin': 200,

'subsample': 1,

'subsample\_freq': 1,

'colsample\_bytree': 0.8,

'reg\_alpha': 1.2, #L1 regularization(>0) 'reg\_lambda': 1.2,#L2 regularization(>0) 'min\_split\_gain': 0.5, #>0

'min\_child\_weight': 1,

'min\_child\_samples': 5, 'is\_unbalance':True,

}

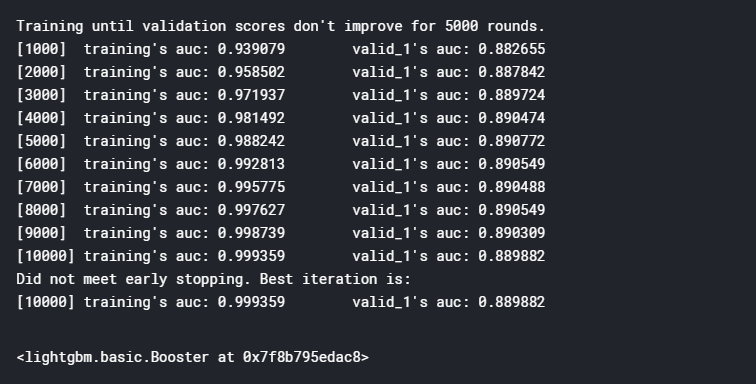
num\_rounds=3000

lgbm= lgb.train(params,lgb\_train,num\_rounds,valid\_sets=[lgb\_train,lgb\_valid], verbose\_eval=100,early\_stopping\_rounds = 1000)

X\_test=test\_df.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)



**R code**

#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(test\_df[,-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 parameters

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

metric = "auc", boost =”gbdt”

min\_sum\_hessian\_in\_leaf = 1,

feature\_fraction = 0.7,

bagging\_fraction = 0.7,

bagging\_freq = 5, learning\_rate=0.05, num\_leaves=80, num\_threads=10, min\_data = 100,

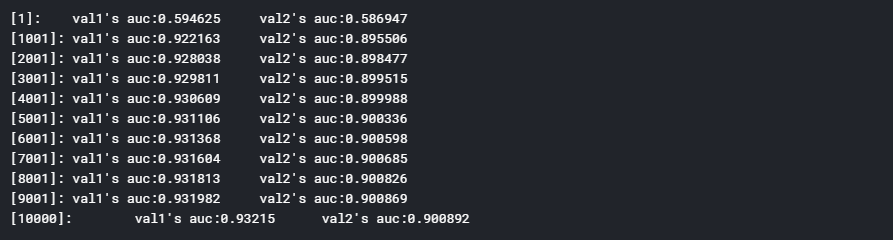
max\_bin = 200,

lambda\_l1 = 8, 25

lambda\_l2 = 1.3, min\_data\_in\_bin=150, min\_gain\_to\_split = 20,

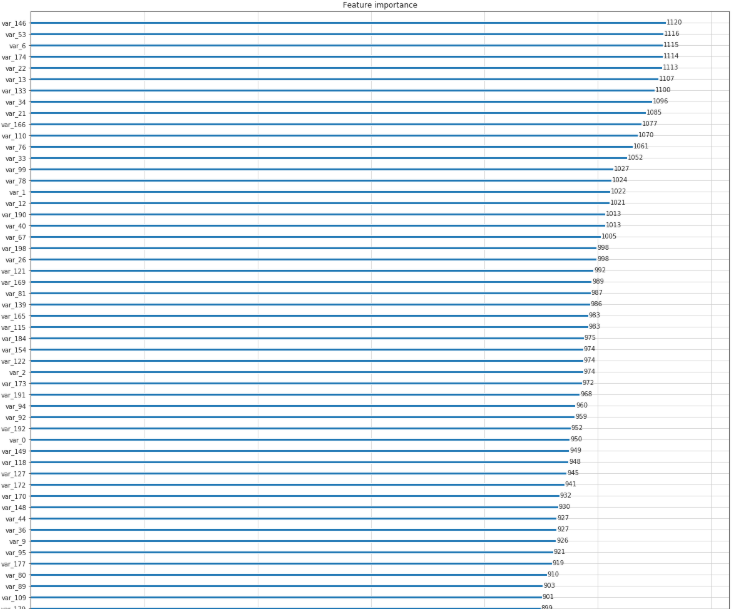
min\_data\_in\_leaf = 40, is\_unbalance = TRUE)

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)



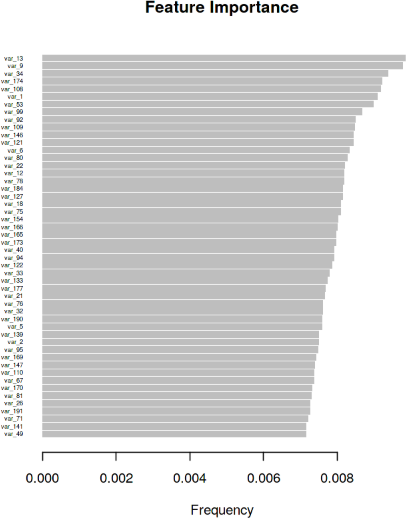
Important features plot Python code

#plot the important features lgb.plot\_importance(lgbm,max\_num\_features=150,importance\_type="split",figsize=(20, 50))



**R code**

tree\_imp <- lgb.importance(lgbm.model, percentage = TRUE) lgb.plot.importance(tree\_imp, top\_n = 50, measure = "Gain")



**Chapter 3**

# Conclusion

##### Model Evaluation

Now, we have 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.

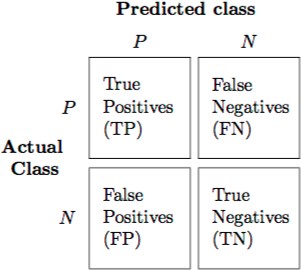
Classification accuracy may be misleading if we have an imbalanced dataset or if we have more than two classes in dataset.

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 = (FP +FN ) / 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 :- Harmonic mean of precision and recall, used to indicate balance between them.

F1 score = 2\*Precision\*Recall

Precison+Recall

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

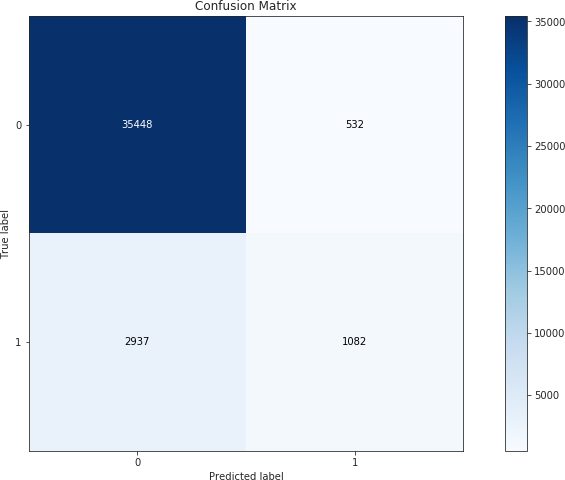
roc\_auc\_score :- It is a metric that computes the area under the Roc curve and also used metric for imbalanced data.

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**

#Confusion matrix cm=confusion\_matrix(y\_valid,cv\_predict) #Plot the confusion matrix

plot\_confusion\_matrix(y\_valid,cv\_predict,normalize=False,figsize=(15,8))



#ROC\_AUC curve

plt.figure() false\_positive\_rate,recall,thresholds=roc\_curve(y\_valid,cv\_predict) roc\_auc=auc(false\_positive\_rate,recall)

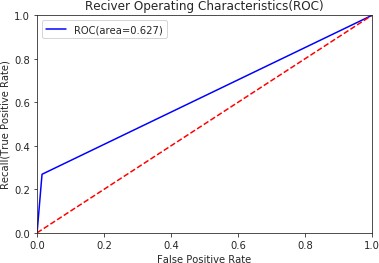
plt.title('Reciver Operating Characteristics(ROC)') plt.plot(false\_positive\_rate,recall,'b',label='ROC(area=**%0.3f**)' %roc\_auc) plt.legend()

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()

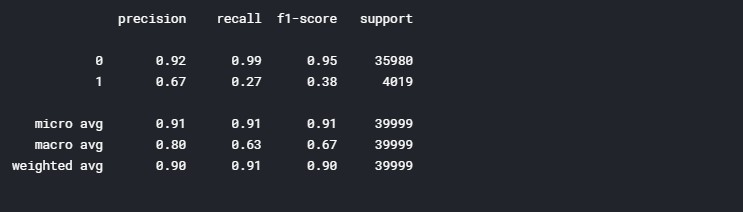
print('AUC:',roc\_auc)



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

**Classification report**

#Classification report scores=classification\_report(y\_valid,cv\_predict) print(scores)



We can observed 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 code**

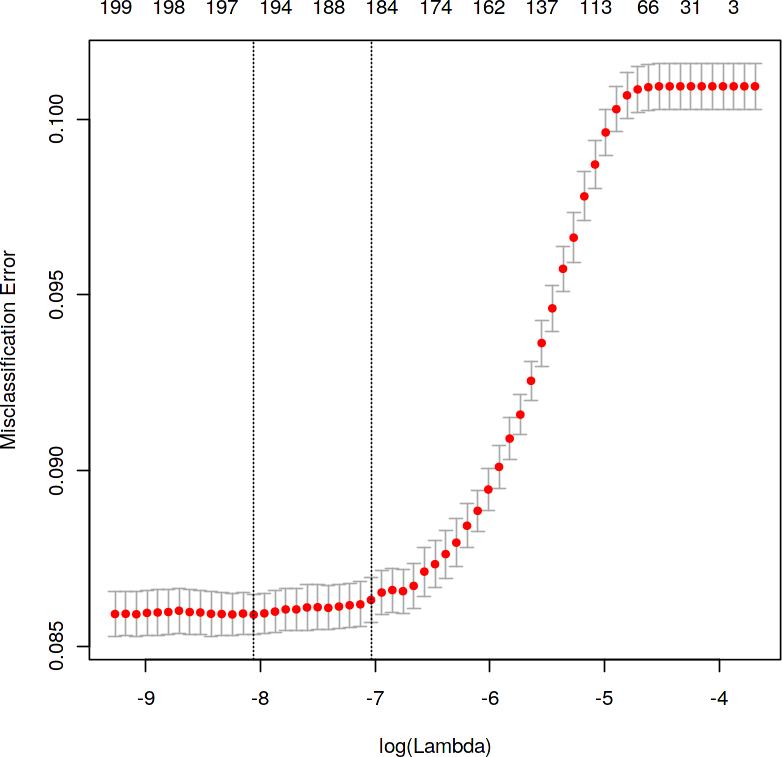
**Logistic Regression**

#Cross validation prediction set.seed(8909)

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

#Plotting misclassification error vs log(lambda) #Minimum lambda-Regularization parameter cv\_lr$lambda.min

#plot the auc score vs log(lambda) plot(cv\_lr)



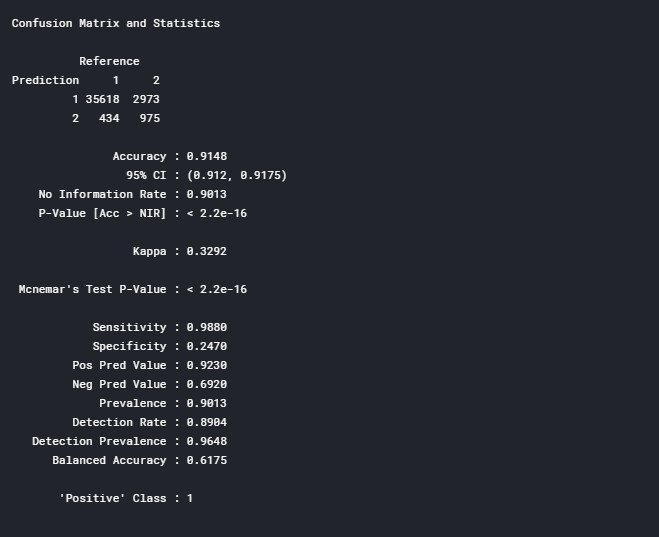
We can observe that miss classification error increases as increasing the log(Lambda).

#Confusion matrix set.seed(689)

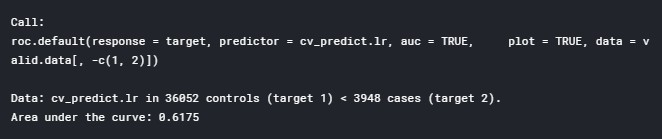
#actual target variable target<-valid.data$target #convert to factor target<-as.factor(target)

#predicted target variable #convert to factor

cv\_predict.lr<-as.factor(cv\_predict.lr) confusionMatrix(data=cv\_predict.lr,reference=target)



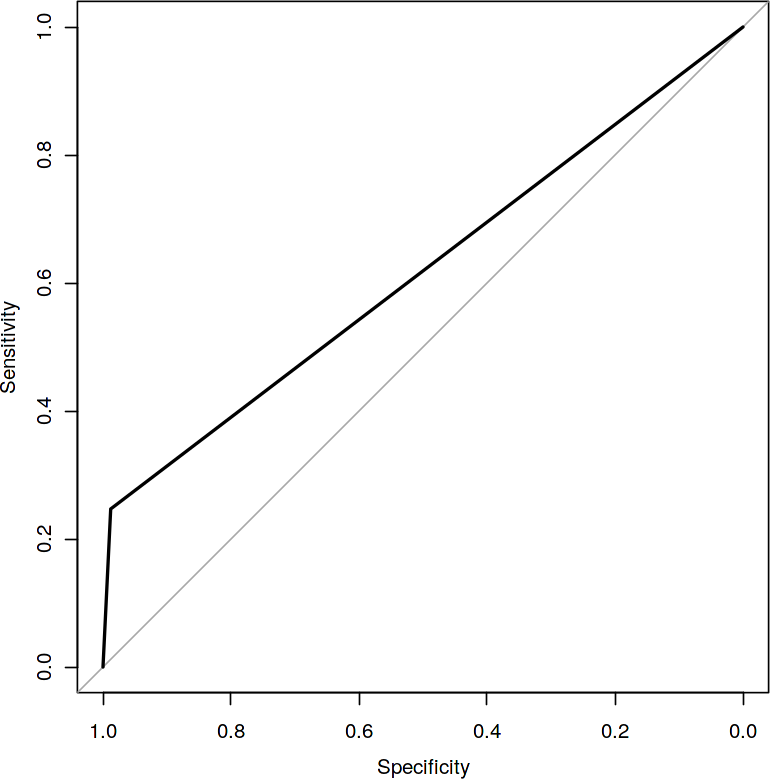
**Receiver operating characteristics(ROC)-Area under curve(AUC) score and curve**



#ROC\_AUC score and curve set.seed(892)

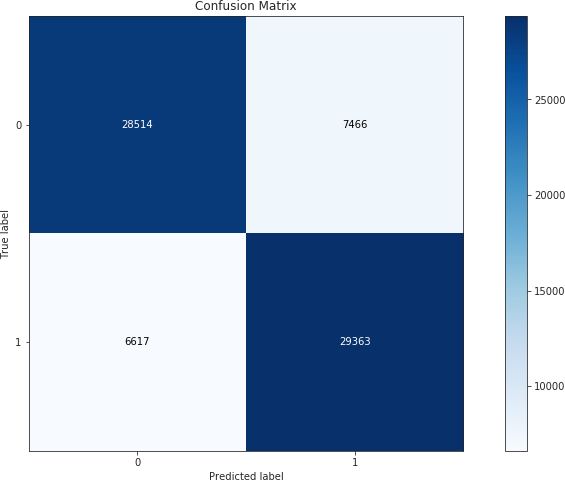
cv\_predict.lr<-as.numeric(cv\_predict.lr)

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



**Python code**

**Synthetic Minority Oversampling Technique (SMOTE)**



#Confusion matrix cm=confusion\_matrix(y\_smote\_v,cv\_pred) #Plot the confusion matrix

plot\_confusion\_matrix(y\_smote\_v,cv\_pred,normalize=False,figsize=(15,8))

**Receiver operating characteristics (ROC)-Area under curve (AUC) score and curve**

#ROC\_AUC curve plt.figure()

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

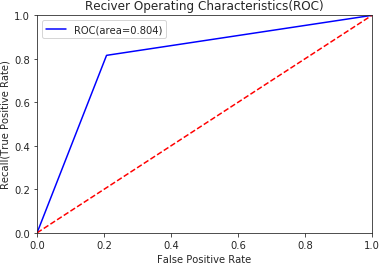
plt.title('Reciver Operating Characteristics(ROC)') plt.plot(false\_positive\_rate,recall,'b',label='ROC(area=**%0.3f**)' %roc\_auc) plt.legend()

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

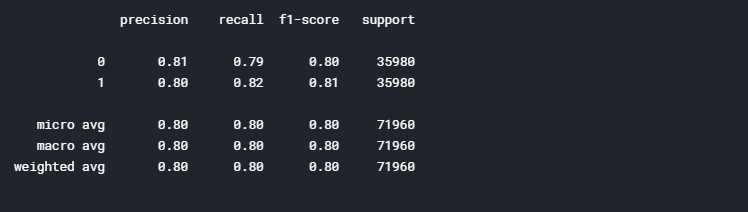
plt.xlim([0.0,1.0])

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**



*#Classification report* scores=classification\_report(y\_smote\_v,cv\_pred) print(scores)

We can observed that smote model is performing well on imbalance data compare to baseline logistic regression.

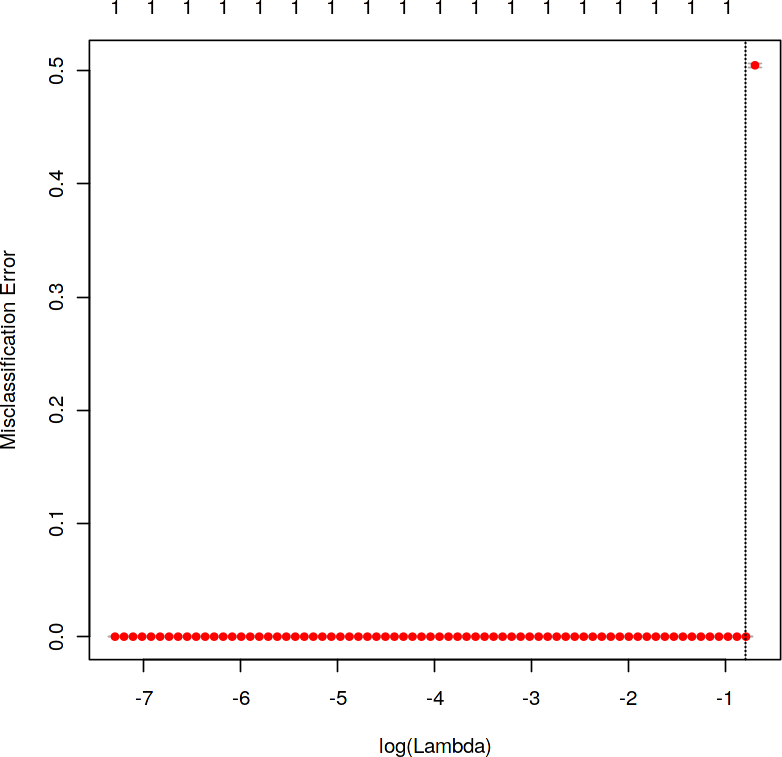
**R code**

**Random Oversampling Examples (ROSE)**

#Plotting misclassification error vs log(lambda) #lambda-Regularization parameter

#Minimum lambda cv\_rose$lambda.min

#plot the auc score vs log(lambda) plot(cv\_rose)

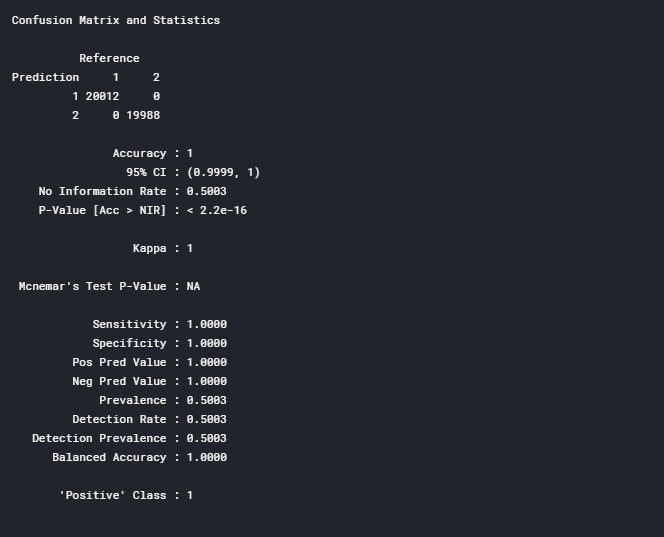


#Confusion matrix set.seed(478)

#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)



**Receiver operating characteristics (ROC)-Area under curve(AUC) score and curve**

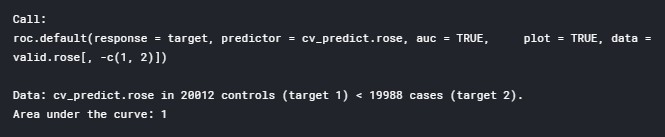
#ROC\_AUC score and curve set.seed(843)

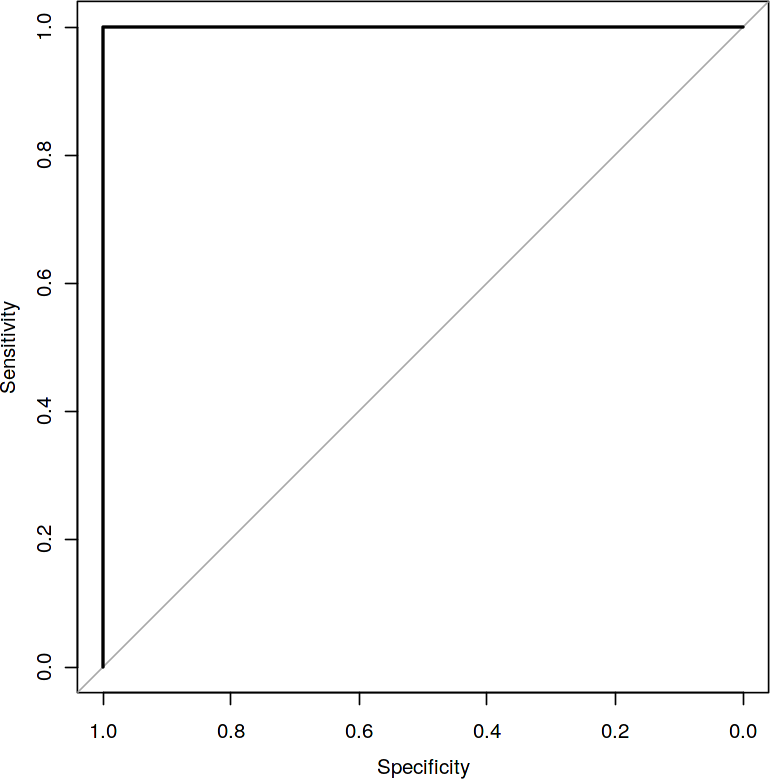
#convert to numeric

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

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

plot=TRUE)





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.

* 1. **Model Selection**

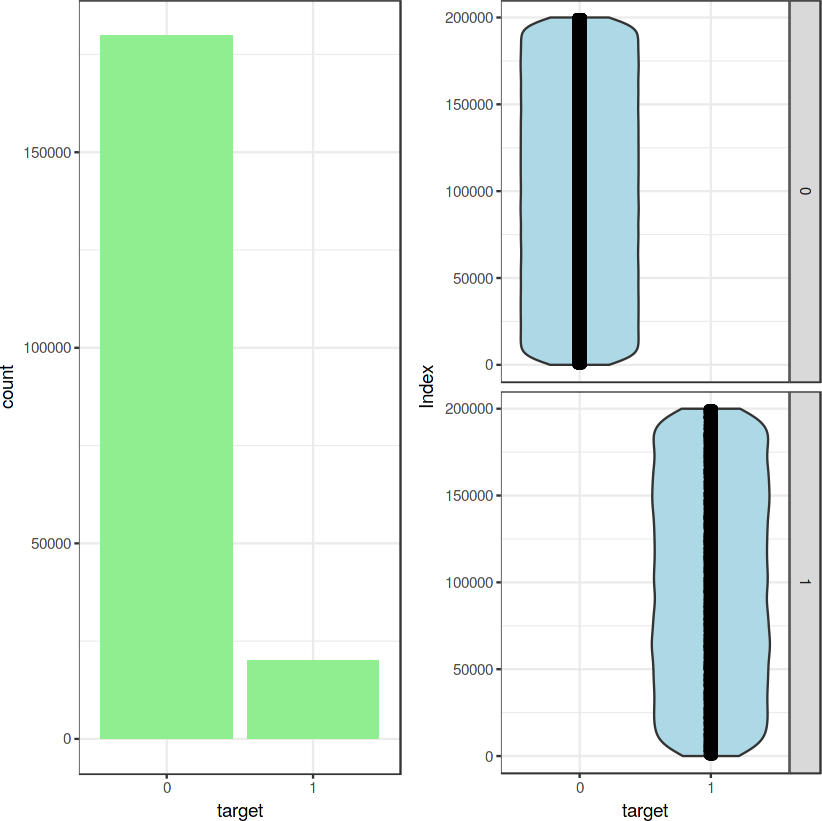
When we compare scores of area under the ROC curve of all the models for an imbalanced data. We could conclude that below points as follow,

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. LightGBM model performed well on imbalanced data.

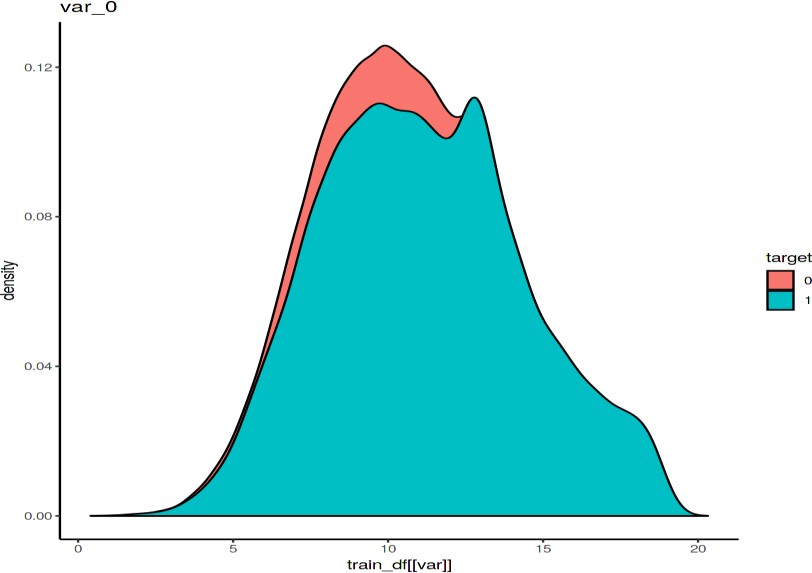
Finally LightGBM is best choice for identifying which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

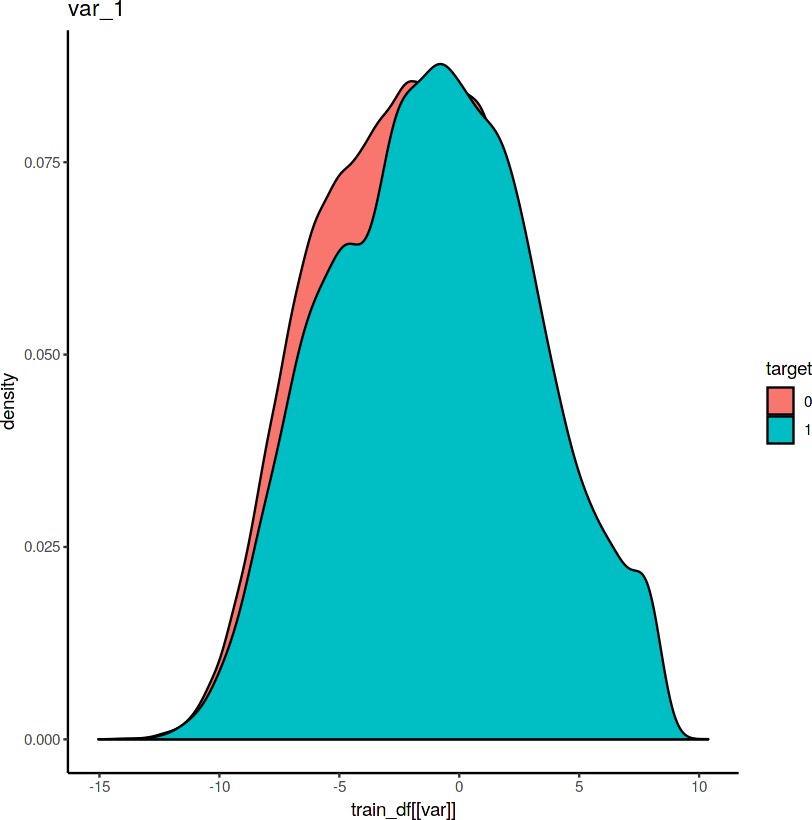
# Appendix A - Extra Figures

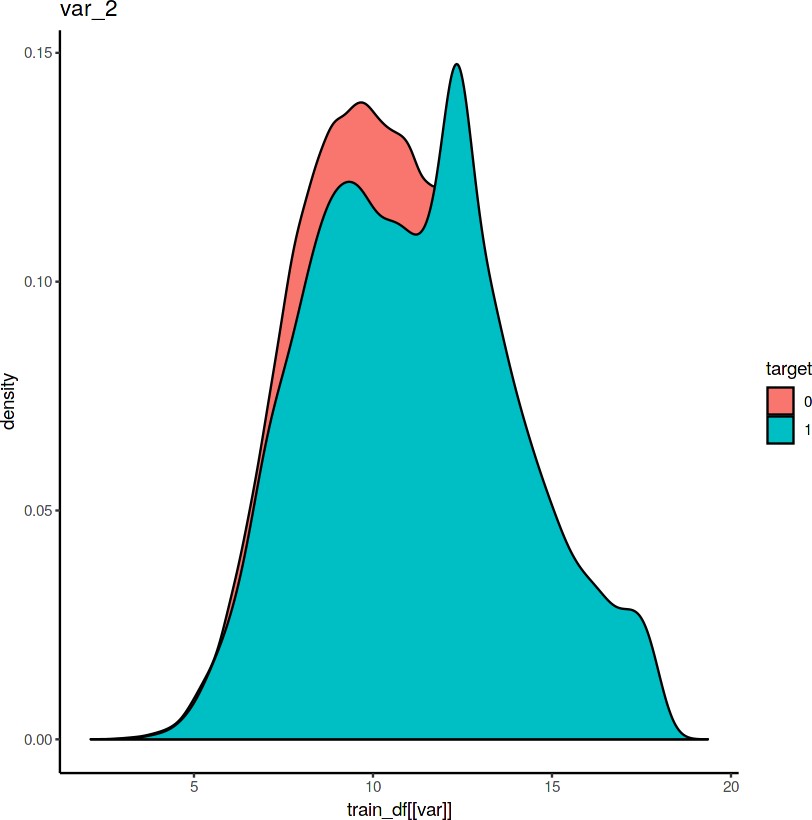
**ggplot2 visualizations**

**Target classes count**

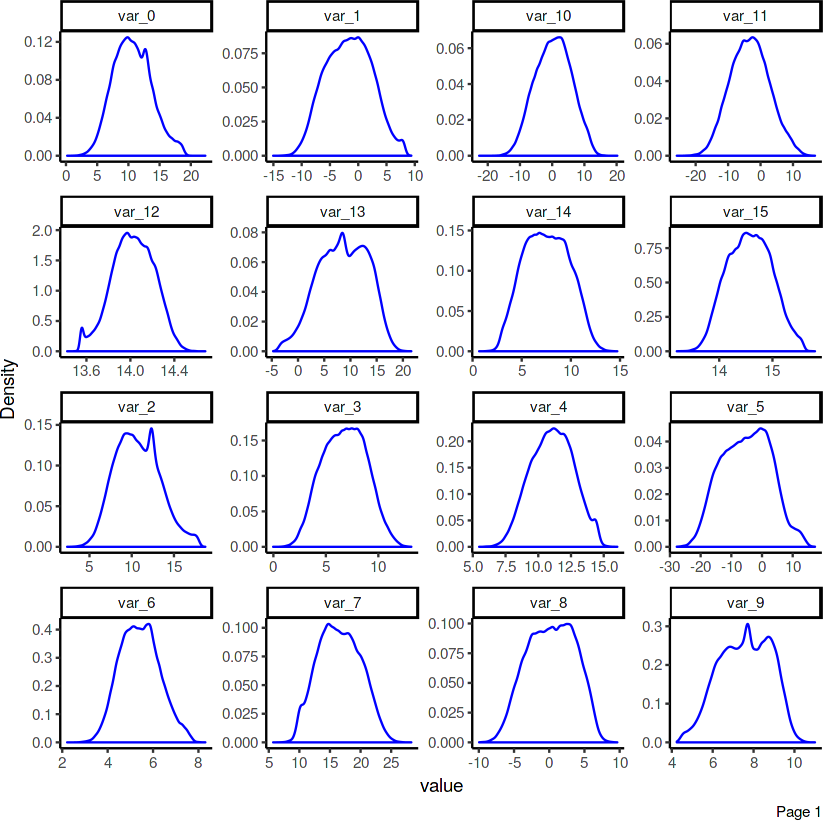
**Distribution of train attributes**

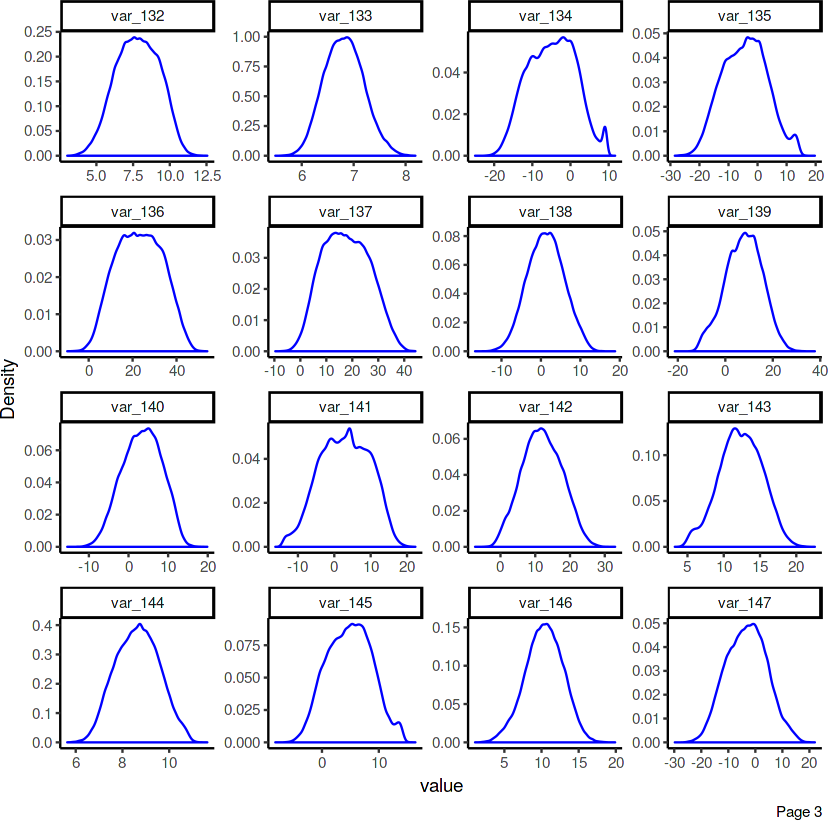


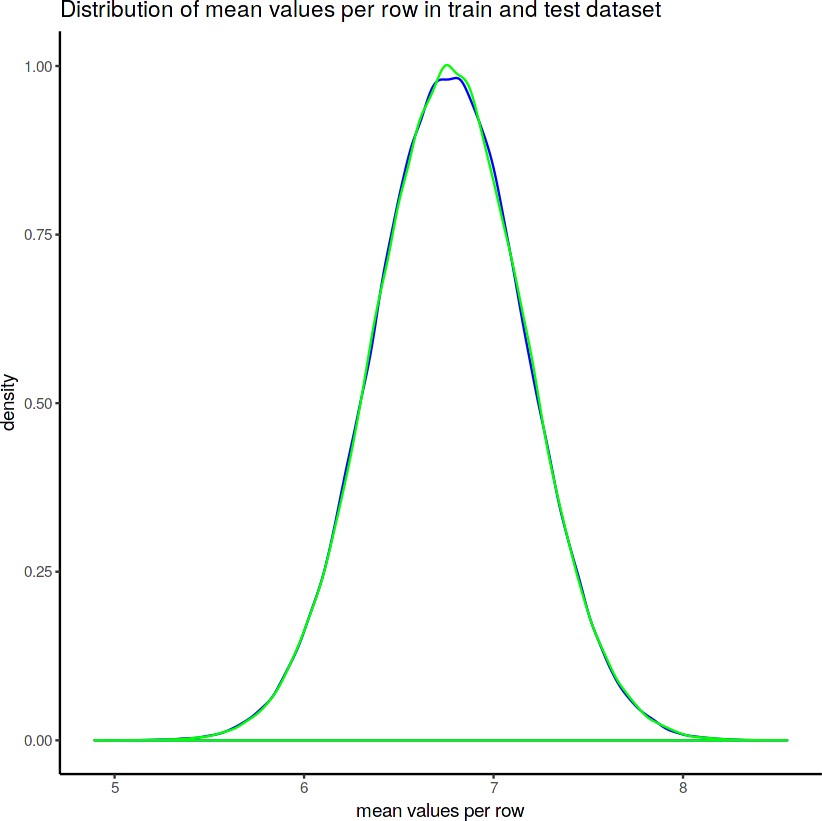


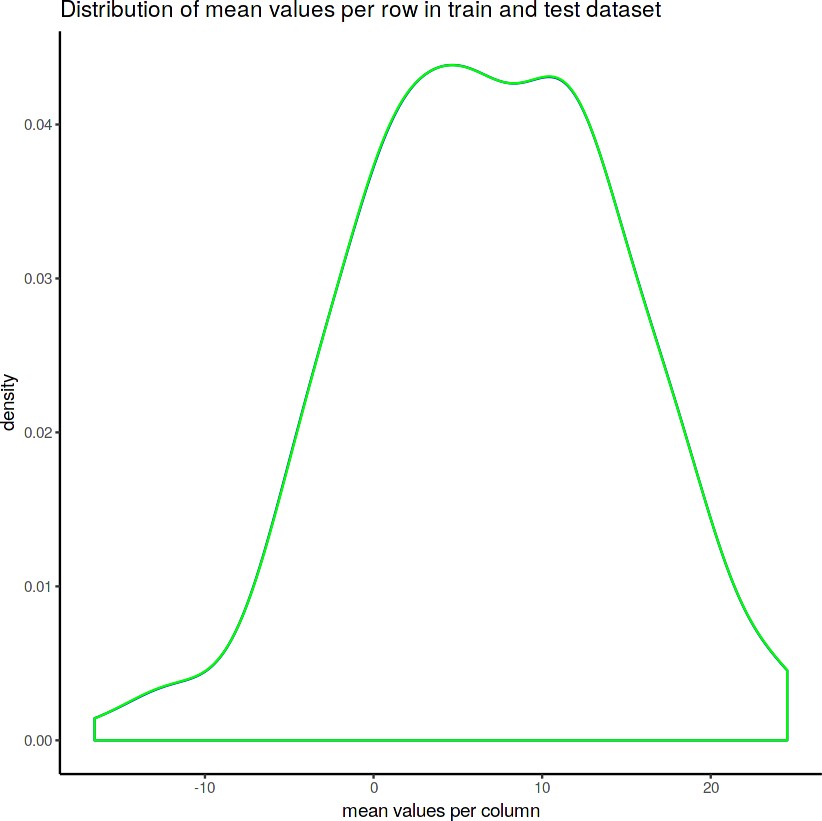


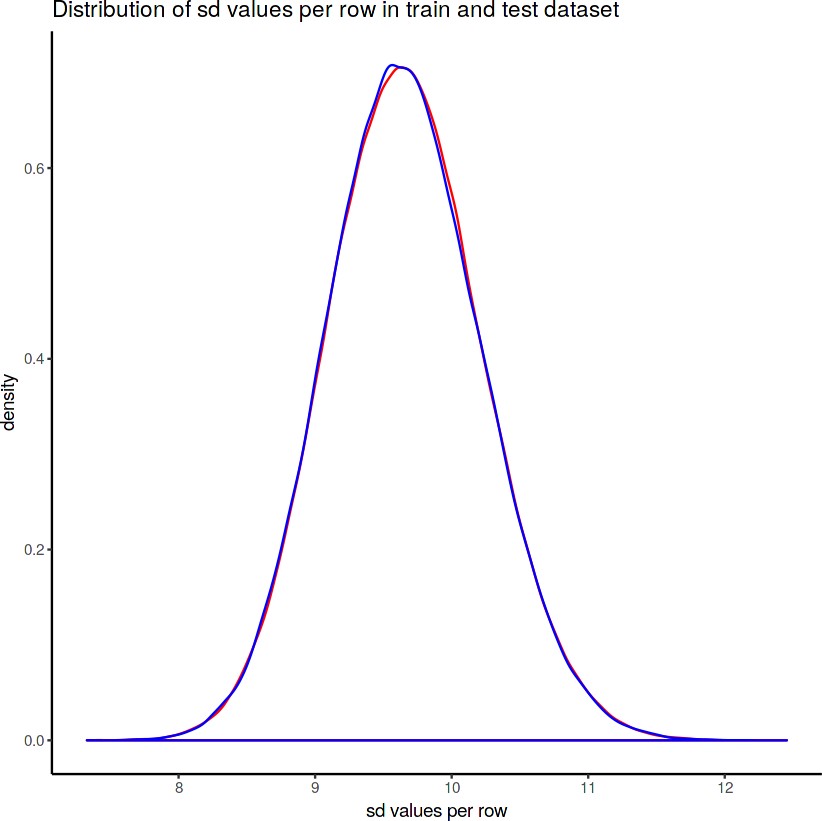
**Distribution of test attributes**

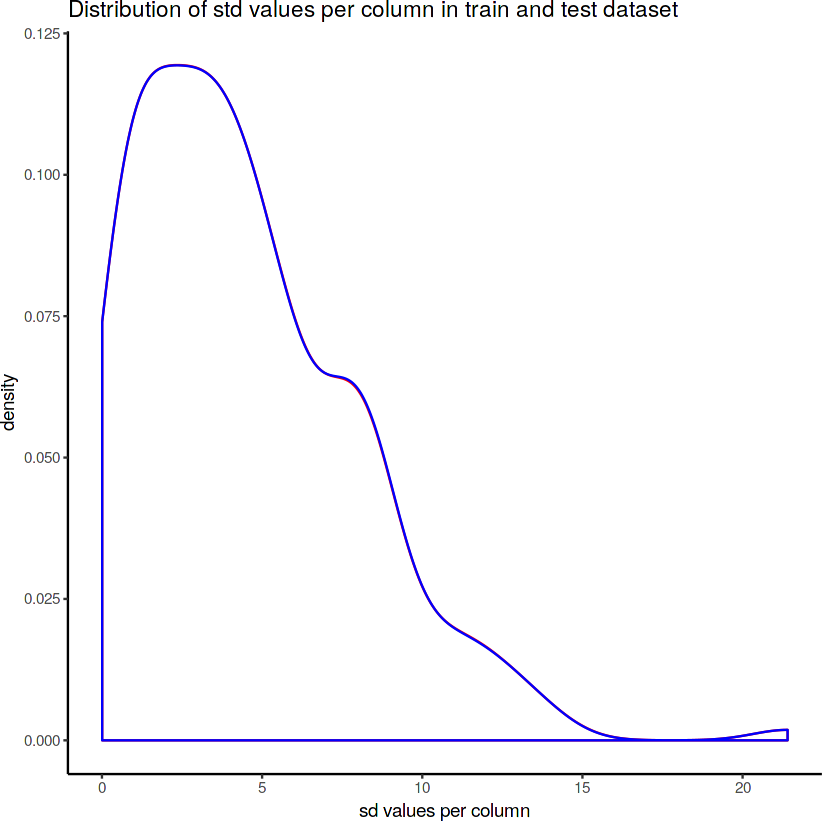


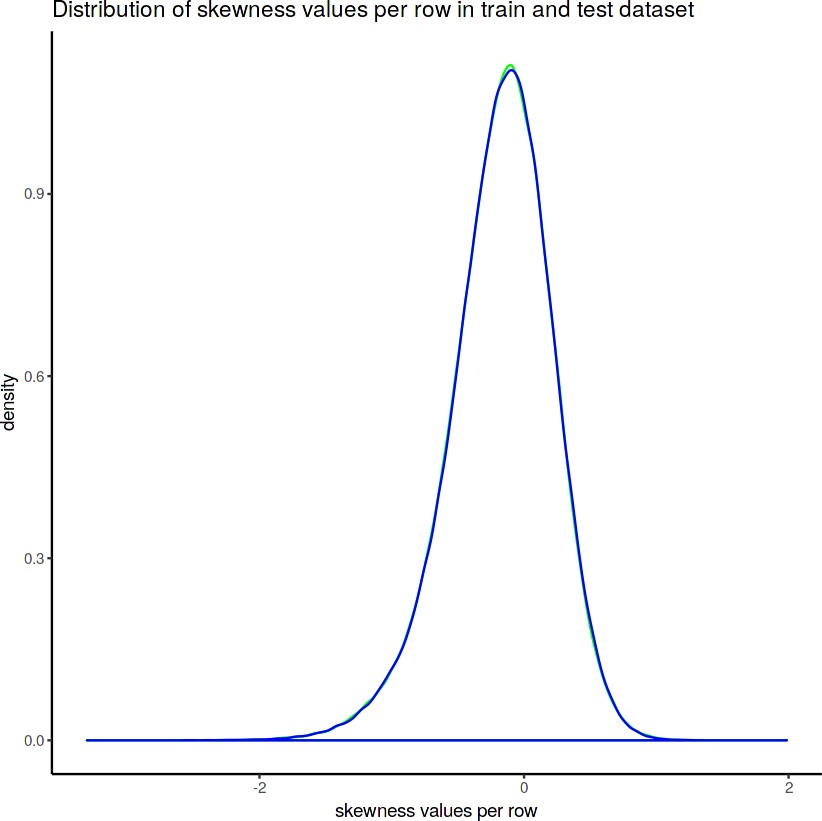


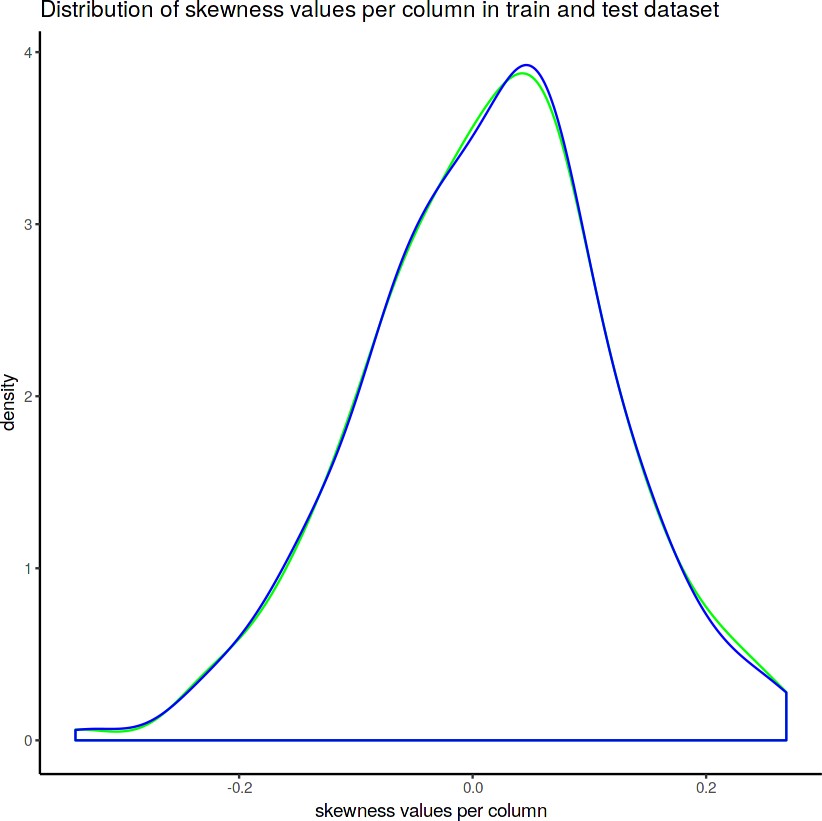


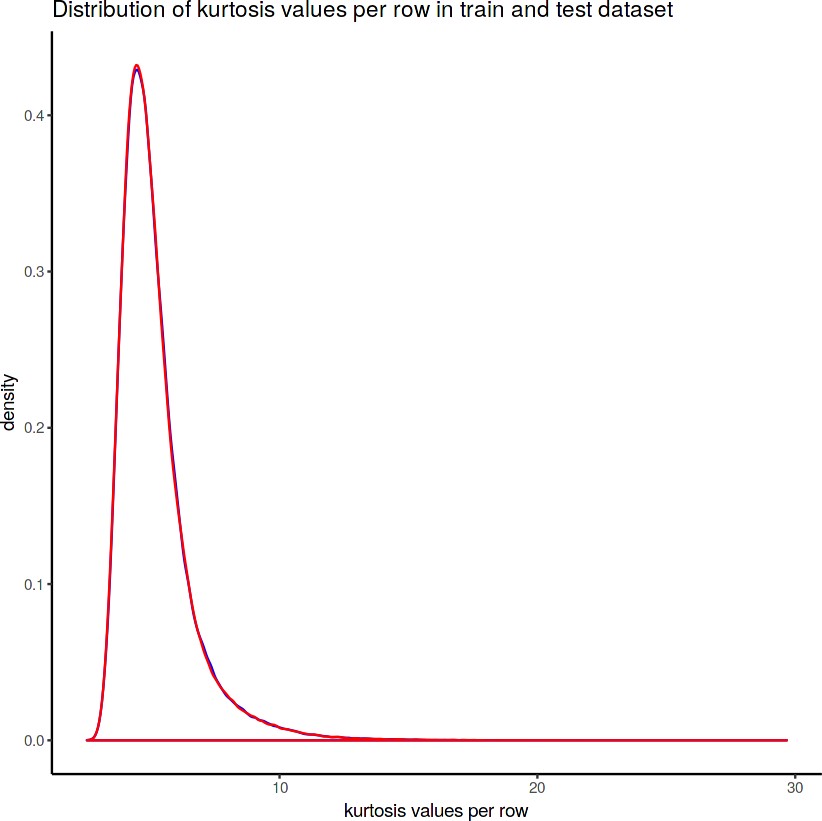


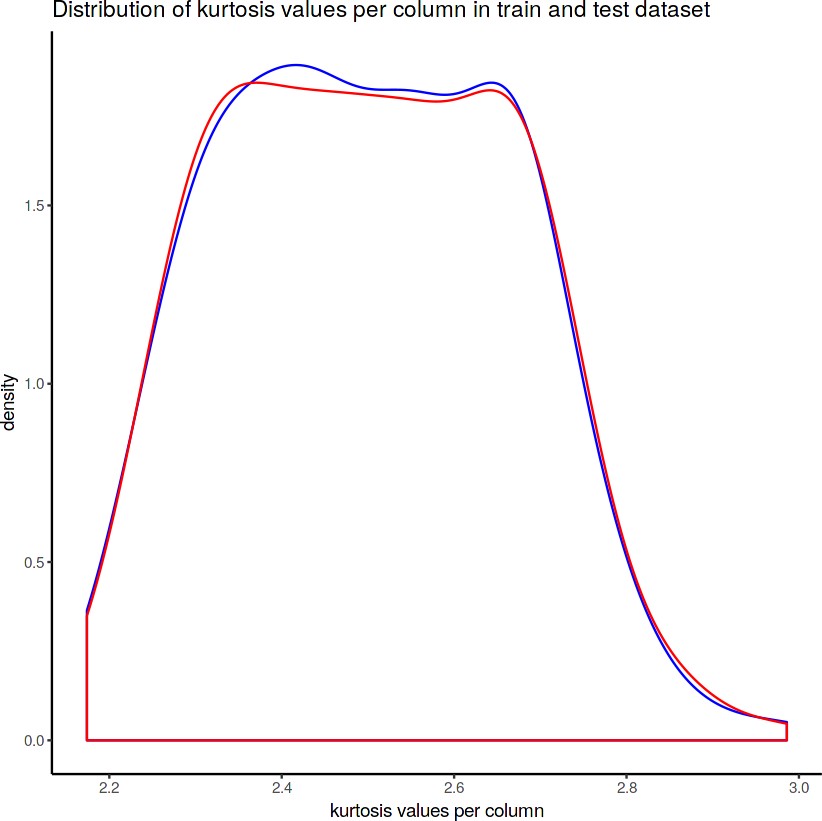












**Appendix B – C o m p l e t e Python a n d R Code Python Code**

**Exploratory Data Analysis**

import numpy as np import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.linear\_model import LogisticRegression from sklearn.model\_selection import RandomizedSearchCV from sklearn.ensemble import RandomForestClassifier from sklearn.tree import DecisionTreeClassifier

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

from sklearn.metrics import roc\_auc\_score,confusion\_matrix,make\_scorer,classifica tion\_report,roc\_curve,auc

from sklearn.model\_selection import StratifiedKFold

from imblearn.over\_sampling import SMOTE, RandomOverSampler

from imblearn.under\_sampling import ClusterCentroids,NearMiss, RandomUnderSampler import lightgbm as lgb

import eli5

from eli5.sklearn import PermutationImportance from sklearn import tree

import graphviz

from pdpbox import pdp, get\_dataset, info\_plots import scikitplot as skplt

from scikitplot.metrics import plot\_confusion\_matrix,plot\_precision\_recall\_curve

from scipy.stats import randint as sp\_randint import warnings warnings.filterwarnings('ignore') random\_state=42

np.random.seed(random\_state)

#importing the train dataset train\_df=pd.read\_csv('../input/train.csv') train\_df.head()

#Shape of the train dataset train\_df.shape

#Summary of the dataset train\_df.describe()

**Target classes count**

%%time

#target classes count target\_class=train\_df['target'].value\_counts() print('Count of target classes :\n',target\_class) #Percentage of target classes count

per\_target\_class=train\_df['target'].value\_counts()/len(train\_df)\*100 print('percentage of count of target classes :\n',per\_target\_class)

#Countplot and violin plot for target classes fig,ax=plt.subplots(1,2,figsize=(20,5)) sns.countplot(train\_df.target.values,ax=ax[0],palette='husl') sns.violinplot(x=train\_df.target.values,y=train\_df.index.values,ax=ax[1], palette='husl') sns.stripplot(x=train\_df.target.values,y=train\_df.index.values,jitter=True, color=' black',linewidth=0.5,size=0.5,alpha=0.5,ax=ax[1],palette='husl') ax[0].set\_xlabel('Target')

ax[1].set\_xlabel('Target') ax[1].set\_ylabel('Index')

**Distribution of train attributes**

%%time

def plot\_train\_attribute\_distribution(t0,t1,label1,label2,train\_attributes): i=0

sns.set\_style('whitegrid')

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

for attribute in train\_attributes: i+=1

plt.subplot(10,10,i) sns.distplot(t0[attribute],hist=False,label=label1) sns.distplot(t1[attribute],hist=False,label=label2) plt.legend()

plt.xlabel('Attribute',)

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

Let us see first 100 train attributes

%%time t0=train\_df[train\_df.target.values==0] t1=train\_df[train\_df.target.values==1]

train\_attributes=train\_df.columns.values[2:102] plot\_train\_attribute\_distribution(t0,t1,'0','1',train\_attributes)

Next 100 train attributes train\_attributes=train\_df.columns.values[102:203] plot\_train\_attribute\_distribution(t0,t1,'0','1',train\_attributes)

**Distribution of test attributes**

#importing the test dataset test\_df=pd.read\_csv('../input/test.csv') test\_df.head()

#Shape of the test dataset test\_df.shape

def plot\_test\_attribute\_distribution(test\_attributes): i=0

sns.set\_style('whitegrid')

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(test\_df[attribute],hist=False) plt.xlabel('Attribute',)

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

#Let us see first 100 test attributes test\_attributes=test\_df.columns.values[1:101] plot\_test\_attribute\_distribution(test\_attributes)

#Next 100 test attributes

test\_attributes=test\_df.columns.values[101:202] plot\_test\_attribute\_distribution(test\_attributes)

**Distribution of mean values in train and test dataset**

%%time

#Distribution of mean values per column in train and test dataset plt.figure(figsize=(16,8))

#train attributes train\_attributes=train\_df.columns.values[2:202] #test attributes test\_attributes=test\_df.columns.values[1:201]

#Distribution plot for mean values per column in train attributes sns.distplot(train\_df[train\_attributes].mean(axis=0),color='blue',kde=True,bins=15 0,label='train')

#Distribution plot for mean values per column in test attributes sns.distplot(test\_df[test\_attributes].mean(axis=0),color='green',kde=True,bins=150

,label='test')

plt.title('Distribution of mean values per column in train and test dataset') plt.legend()

plt.show()

#Distribution of mean values per row in train and test dataset plt.figure(figsize=(16,8))

#Distribution plot for mean values per row in train attributes sns.distplot(train\_df[train\_attributes].mean(axis=1),color='blue',kde=True,bins=15 0,label='train')

#Distribution plot for mean values per row in test attributes sns.distplot(test\_df[test\_attributes].mean(axis=1),color='green',kde=True, bins=15 0, label='test')

plt.title('Distribution of mean values per row in train and test dataset') plt.legend()

plt.show()

**Distribution of standard deviation (std) in train and test dataset**

%%time

#Distribution of std values per column in train and test dataset plt.figure (figsize= (16, 8))

#train attributes train\_attributes=train\_df.columns.values [2:202] #test attributes test\_attributes=test\_df.columns.values [1:201]

#Distribution plot for std values per column in train attributes sns.distplot(train\_df[train\_attributes].std(axis=0),color='red',kde=True, bins=150,label='train')

#Distribution plot for std values per column in test attributes sns.distplot (test\_df [test\_attributes].std(axis=0),color='blue',kde=True,bins=150, label='test')

plt.title('Distribution of std values per column in train and test dataset') plt.legend()

plt.show()

#Distribution of std values per row in train and test dataset plt.figure(figsize=(16,8))

#Distribution plot for std values per row in train attributes sns.distplot(train\_df[train\_attributes].std(axis=1),color='red',kde=True,bins=150

, label='train')

#Distribution plot for std values per row in test attributes sns.distplot(test\_df[test\_attributes].std(axis=1),color='blue',kde=True, bins=150

, label='test')

plt.title('Distribution of std values per row in train and test dataset') plt.legend()

**Distribution of skewness in train and test dataset**

%%time

#Distribution of skew values per column in train and test dataset plt.figure(figsize=(16,8))

#train attributes train\_attributes=train\_df.columns.values[2:202] #test attributes test\_attributes=test\_df.columns.values[1:201]

#Distribution plot for skew values per column in train attributes sns.distplot(train\_df[train\_attributes].skew(axis=0),color='green',kde=True,

bins=150,label='train')

#Distribution plot for skew values per column in test attributes sns.distplot(test\_df[test\_attributes].skew(axis=0),color='blue',kde=True,bins=150

,label='test')

plt.title('Distribution of skewness values per column in train and test dataset') plt.legend()

plt.show()

#Distribution of skew values per row in train and test dataset plt.figure(figsize=(16,8))

#Distribution plot for skew values per row in train attributes sns.distplot(train\_df[train\_attributes].skew(axis=1),color='green',kde=True, bins=150,label='train')

#Distribution plot for skew values per row in test attributes sns.distplot(test\_df[test\_attributes].skew(axis=1),color='blue',kde=True, bins=150, label='test')

plt.title('Distribution of skewness values per row in train and test dataset') plt.legend()

plt.show()

**Distribution of kurtosis values in train and test dataset**

%%time

#Distribution of kurtosis values per column in train and test dataset plt.figure(figsize=(16,8))

#train attributes train\_attributes=train\_df.columns.values[2:202] #test attributes test\_attributes=test\_df.columns.values[1:201]

#Distribution plot for kurtosis values per column in train attributes sns.distplot(train\_df[train\_attributes].kurtosis(axis=0),color='blue',kde=True, bins=150,label='train')

#Distribution plot for kurtosis values per column in test attributes sns.distplot(test\_df[test\_attributes].kurtosis(axis=0),color='red',kde=True, bins=150,label='test')

plt.title('Distribution of kurtosis values per column in train and test dataset') plt.legend()

plt.show()

#Distribution of kutosis values per row in train and test dataset plt.figure(figsize=(16,8))

#Distribution plot for kurtosis values per row in train attributes sns.distplot(train\_df[train\_attributes].kurtosis(axis=1),color='blue',kde=True, bins=150,label='train')

#Distribution plot for kurtosis values per row in test attributes sns.distplot(test\_df[test\_attributes].kurtosis(axis=1),color='red',kde=True, bins=150, label='test')

plt.title('Distribution of kurtosis values per row in train and test dataset') plt.legend()

plt.show()

**Missing value analysis and Correlations**

%%time

#Finding the missing values in train and test data train\_missing=train\_df.isnull().sum().sum() test\_missing=test\_df.isnull().sum().sum() print('Missing values in train data :',train\_missing) print('Missing values in test data :',test\_missing

%%time

#Correlations in train attributes train\_attributes=train\_df.columns.values[2:202]

train\_correlations=train\_df[train\_attributes].corr().abs().unstack().sort\_values(k ind='quicksort').reset\_index() train\_correlations=train\_correlations[train\_correlations['level\_0']!=train\_correla tions['level\_1']]

print(train\_correlations.head(10)) print(train\_correlations.tail(10))

%%time

#Correlations in test attributes test\_attributes=test\_df.columns.values[1:201]

test\_correlations=test\_df[test\_attributes].corr().abs().unstack().sort\_values(kind

='quicksort').reset\_index() test\_correlations=test\_correlations[test\_correlations['level\_0']!=test\_correlation s['level\_1']]

print(test\_correlations.head(10)) print(test\_correlations.tail(10))

#Correlation plot

%%time

#Correlations in train data train\_correlations=train\_df[train\_attributes].corr() train\_correlations=train\_correlations.values.flatten() train\_correlations=train\_correlations[train\_correlations!=1] test\_correlations=test\_df[test\_attributes].corr() #Correlations in test data test\_correlations=test\_correlations.values.flatten() test\_correlations=test\_correlations[test\_correlations!=1]

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

#Distribution plot for correlations in train data sns.distplot(train\_correlations, color="Red", label="train") #Distribution plot for correlations in test data sns.distplot(test\_correlations, color="Blue", label="test") plt.xlabel("Correlation values found in train and test") plt.ylabel("Density")

plt.title("Correlation distribution plot for train and test attributes") plt.legend()

Feature engineering

#training data X=train\_df.drop(columns=['ID\_code','target'],axis=1) test=test\_df.drop(columns=['ID\_code'],axis=1) y=train\_df['target']

#Split the training data X\_train,X\_valid,y\_train,y\_valid=train\_test\_split(X,y,random\_state=42)

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)

%%time

#Random forest classifier rf\_model=RandomForestClassifier(n\_estimators=10,random\_state=42) #fitting the model

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

#Permutation importance

%%time

from eli5.sklearn import PermutationImportance perm\_imp=PermutationImportance(rf\_model,random\_state=42) #fitting the model

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

%%time

#Important features eli5.show\_weights(perm\_imp,feature\_names=X\_valid.columns.tolist(),top=200)

#partial dependence plots

%%time

#Create the data we will plot 'var\_81'

features=[v for v in X\_valid.columns if v not in ['ID\_code','target']] pdp\_data=pdp.pdp\_isolate(rf\_model,dataset=X\_valid,model\_features=features,feature= 'var\_81')

#plot feature "var\_81" pdp.pdp\_plot(pdp\_data,'var\_81') plt.show()

%%time

#Create the data we will plot pdp\_data=pdp.pdp\_isolate(rf\_model,dataset=X\_valid,model\_features=features,feature= ' var\_109')

#plot feature "var\_109" pdp.pdp\_plot(pdp\_data,'var\_109') plt.show()

Handling of imbalanced data

#Training data X=train\_df.drop(['ID\_code','target'],axis=1) Y=train\_df['target']

#StratifiedKFold cross validator cv=StratifiedKFold(n\_splits=5,random\_state=42,shuffle=True) for train\_index,valid\_index **in** cv.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)

%%time

#Logistic regression model lr\_model=LogisticRegression(random\_state=42) #fitting the lr model lr\_model.fit(X\_train,y\_train)

#Accuracy of the model lr\_score=lr\_model.score(X\_train,y\_train) print('Accuracy of the lr\_model :',lr\_score)

%%time

#Cross validation prediction 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))

#Confusion matrix cm=confusion\_matrix(y\_valid,cv\_predict) #Plot the confusion matrix

plot\_confusion\_matrix(y\_valid,cv\_predict,normalize=False,figsize=(15,8))

#ROC\_AUC score roc\_score=roc\_auc\_score(y\_valid,cv\_predict) print('ROC score :',roc\_score)

#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)

#Classification report scores=classification\_report(y\_valid,cv\_predict) print(scores)

%%time

#Predicting the model X\_test=test\_df.drop(['ID\_code'],axis=1) lr\_pred=lr\_model.predict(X\_test) print(lr\_pred)

#Synthetic Minority Oversampling Technique sm = SMOTE(random\_state=42, ratio=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=42) #fitting the smote model smote.fit(X\_smote,y\_smote)

#Accuracy of the model smote\_score=smote.score(X\_smote,y\_smote) print('Accuracy of the smote\_model :',smote\_score)

#Cross validation prediction 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\_val\_score :',np.average(cv\_score))

#Confusion matrix cm=confusion\_matrix(y\_smote\_v,cv\_pred) #Plot the confusion matrix

plot\_confusion\_matrix(y\_smote\_v,cv\_pred,normalize=False,figsize=(15,8))

#ROC\_AUC score roc\_score=roc\_auc\_score(y\_smote\_v,cv\_pred) print('ROC score :',roc\_score)

#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)

%%time

#Predicting the model X\_test=test\_df.drop(['ID\_code'],axis=1) smote\_pred=smote.predict(X\_test) print(smote\_pred)

LightGBM

#Training the model #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': 8, 'metric':'auc', 'num\_leaves': 100,

'learning\_rate': 0.03,

'max\_bin': 950, #default 255

'subsample\_for\_bin': 200,

'subsample': 1,

'subsample\_freq': 1,

'colsample\_bytree': 0.8,

'reg\_alpha': 1.2, #L1 regularization(>0) 'reg\_lambda': 1.2,#L2 regularization(>0) 'min\_split\_gain': 0.5, #>0

'min\_child\_weight': 1,

'min\_child\_samples': 5, 'is\_unbalance':True,

}

num\_rounds=3000

lgbm= lgb.train(params,lgb\_train,num\_rounds,valid\_sets=[lgb\_train,lgb\_valid], verbose\_eval=100,early\_stopping\_rounds = 1000)

X\_test=test\_df.drop(['ID\_code'],axis=1) #predict the model

#probability predictions lgbm\_predict\_prob=lgbm.predict(X\_test,random\_state=42,num\_iteration=lgbm.best\_iter ation)

#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)

#plot the important features lgb.plot\_importance(lgbm,max\_num\_features=150,importance\_type="split",figsize=(20, 50))

#final submission sub\_df=pd.DataFrame({'ID\_code':test\_df['ID\_code'].values}) sub\_df['lgbm\_predict\_prob']=lgbm\_predict\_prob sub\_df['lgbm\_predict']=lgbm\_predict sub\_df['smote\_predict']=smote\_pred sub\_df.to\_csv('submission.csv',index=False)

sub\_df.head()

# R Code

Exploratory Data Analysis

#Load the libraries library(tidyverse) library(moments) library(DataExplorer) library(caret) library(Matrix) library(mlbench) library(caTools) library(randomForest) library(glmnet) library(mlr) library(unbalanced) library(vita)

library(rBayesianOptimization) library(lightgbm) library(boot)

library(pROC) library(DMwR) library(ROSE) library(yardstick)

#loading the train data

train\_df<-read.csv('../input/train.csv') head(train\_df)

#Dimension of train data dim(train\_df)

#Summary of the dataset str(train\_df)

#convert to factor

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

**Target classes count in train data**

require(gridExtra) #Count of target classes table(train\_df$target)

#Percenatge counts of target classes table(train\_df$target)/length(train\_df$target)\*100 #Bar plot for count of target classes

plot1<-ggplot(train\_df,aes(target))+theme\_bw()+geom\_bar(stat='count',fill='lig htgreen')

#Violin with jitter plots for target classes

plot2<-ggplot(train\_df,aes(x=target,y=1:nrow(train\_df)))+theme\_bw()+geom\_violi n(fill='lightblue')+

facet\_grid(train\_df$target)+geom\_jitter(width=0.02)+labs(y='Index') grid.arrange(plot1,plot2, ncol=2)

**Distribution of train attributes**

#Distribution of train attributes from 3 to 102 for (var in names(train\_df)[c(3:102)]){

target<-train\_df$target

plot<-ggplot(train\_df, aes(x=train\_df[[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(train\_df)[c(103:202)]){

target<-train\_df$target

plot<-ggplot(train\_df, aes(x=train\_df[[var]], fill=target)) + geom\_density(kernel='gaussian') + ggtitle(var)+theme\_classic() print(plot)

}

**Distribution of test attributes**

#loading test data

test\_df<-read.csv('../input/test.csv') head(test\_df)

#Dimension of test dataset dim(test\_df)

#Distribution of test attributes from 2 to 101 plot\_density(test\_df[,c(2:101)], ggtheme = theme\_classic(), geom\_density\_args = list(color='blue'))

#Distribution of test attributes from 102 to 201 plot\_density(test\_df[,c(102:201)], ggtheme = theme\_classic(), geom\_density\_args = list(color='blue'))

**Distribution of mean values in train and test dataset**

#Applying the function to find mean values per row in train and test data. train\_mean<-apply(train\_df[,-c(1,2)],MARGIN=1,FUN=mean)

test\_mean<-apply(test\_df[,-c(1)],MARGIN=1,FUN=mean) ggplot()+

#Distribution of mean values per row in train data geom\_density(data=train\_df[,-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=test\_df[,-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(train\_df[,-c(1,2)],MARGIN=2,FUN=mean)

test\_mean<-apply(test\_df[,-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 row in train and test dataset")

**Distribution of standard deviation in train and test dataset**

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

train\_sd<-apply(train\_df[,-c(1,2)],MARGIN=1,FUN=sd) test\_sd<-apply(test\_df[,-c(1)],MARGIN=1,FUN=sd) ggplot()+

#Distribution of sd values per row in train data geom\_density(data=train\_df[,-c(1,2)],aes(x=train\_sd),kernel='gaussian', show.le gend=TRUE,color='red')+theme\_classic()+

#Distribution of mean values per row in test data geom\_density(data=test\_df[,-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(train\_df[,-c(1,2)],MARGIN=2,FUN=sd)

test\_sd<-apply(test\_df[,-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")

**Distribution of skewness values in train and test dataset**

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

train\_skew<-apply(train\_df[,-c(1,2)],MARGIN=1,FUN=skewness) test\_skew<-apply(test\_df[,-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(train\_df[,-c(1,2)],MARGIN=2,FUN=skewness) test\_skew<-apply(test\_df[,-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")

**Distribution of kurtosis values in train and test dataset**

#Applying the function to find kurtosis values per row in train and test data. train\_kurtosis<-apply(train\_df[,-c(1,2)],MARGIN=1,FUN=kurtosis) test\_kurtosis<-apply(test\_df[,-c(1)],MARGIN=1,FUN=kurtosis)

ggplot()+

#Distribution of sd values per column in train data geom\_density(aes(x=train\_kurtosis),kernel='gaussian',show.legend=TRUE, color='b lue')+theme\_classic()+

#Distribution of sd values per column 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(train\_df[,-c(1,2)],MARGIN=2,FUN=kurtosis) test\_kurtosis<-apply(test\_df[,-c(1)],MARGIN=2,FUN=kurtosis) ggplot()+

#Distribution of sd values per column in train data geom\_density(aes(x=train\_kurtosis),kernel='gaussian',show.legend=TRUE, color='blue')+theme\_classic()+

#Distribution of sd 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 and Correlations**

#Finding the missing values in train data missing\_val<-data.frame(missing\_val=apply(train\_df,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(test\_df,2, function(x){sum(is.na(x))}))

missing\_val<-sum(missing\_val) missing\_val

#Correlations in train data

train\_df$target<-as.numeric(train\_df$target) train\_correlations<-cor(train\_df[,c(2:202)]) train\_correlations

#Correlations in test data test\_correlations<-cor(test\_df[,c(2:201)]) test\_correlations

**Feature Engineering**

#Split the training data

train\_index<-sample(1:nrow(train\_df),0.75\*nrow(train\_df)) train\_data<-train\_df[train\_index,]

valid\_data<-train\_df[-train\_index,] dim(train\_data)

dim(valid\_data)

#Training the Random forest classifier set.seed(2732)

train\_data$target<-as.factor(train\_data$target) mtry<-floor(sqrt(200))

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

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

#Variable importance

VarImp<-importance(rf,type=2)

VarImp

#Partial dependence plot #We will plot "var\_81"

partialPlot(rf,valid\_data[,-c(1,2)],valid\_data$var\_81,xlab='var\_81

#We will plot "var\_12"

partialPlot(rf,valid\_data[,-c(1,2)],valid\_data$var\_12,xlab='var\_12')

**Handling of imbalanced data**

#Split the data using CreateDataPartition set.seed(689)

train.index<-createDataPartition(train\_df$target,p=0.8,list=FALSE) train.data<-train\_df[train.index,]

valid.data<-train\_df[-train.index,] dim(train.data)

dim(valid.data)

#training dataset set.seed(682)

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 data

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

#Logistic regression model set.seed(667)

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

#Cross validation prediction set.seed(8909)

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

#Minimum lambda cv\_lr$lambda.min

#plot the auc score vs log(lambda) plot(cv\_lr)

#Model performance on validation dataset set.seed(5363)

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

#Confusion matrix set.seed(689)

#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)

#ROC\_AUC score and curve set.seed(892)

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)

#predict the model set.seed(763)

lr\_pred<-predict(lr\_model,test,type='class') lr\_pred

#Random Oversampling Examples(ROSE) set.seed(699)

#train.data$target<-as.factor(train.data$target)

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

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

#Logistic regression model set.seed(462)

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

summary(lr\_rose)

#Cross validation prediction set.seed(473)

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

cv\_rose

#Minimum lambda cv\_rose$lambda.min

#plot the auc score vs log(lambda) plot(cv\_rose)

#Model performance on validation dataset set.seed(442)

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

cv\_predict.rose

#Confusion matrix set.seed(478)

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

#ROC\_AUC score and curve set.seed(843)

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)

#predict the model set.seed(6543)

rose\_pred<-predict(lr\_rose,test,type='class') rose\_pred

#Convert data frame to matrix set.seed(5432)

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(test\_df[,-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)

set.seed(653)

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

metric = "auc", min\_sum\_hessian\_in\_leaf = 1,

feature\_fraction = 0.7,

bagging\_fraction = 0.7,

bagging\_freq = 5, learning\_rate=0.1, num\_leaves=100, num\_threads=8, min\_data = 100,

max\_bin = 200,

lambda\_l1 = 8,

lambda\_l2 = 1.3, min\_data\_in\_bin=150, min\_gain\_to\_split = 20,

min\_data\_in\_leaf = 40, is\_unbalance = TRUE)

set.seed(7663)

lgbm.model <- lgb.train(params = lgb.grid, data = lgb.train, nrounds =3000, eval\_freq =100,valids=list(val1=lgb.train,val2=lgb.valid), early\_stopping\_rounds = 1000)

#lgbm model performance on test data

set.seed(6532)

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)

set.seed(6521)

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

lgb.plot.importance(tree\_imp, top\_n = 150, measure = "Gain")

sub\_df<-data.frame(ID\_code=test\_df$ID\_code,lgb\_predict\_prob=lgbm\_pred\_prob, lgb \_predict=lgbm\_pred,smote\_predict=smote\_pred) write.csv(sub\_df,'submission.CSV',row.names=F)

head(sub\_df)

# References

[1]. [https://www.kaggle.com](https://www.kaggle.com/)

[2]. <http://rprogramming.net/>

[3]. <https://medium.com/>

[4]. **Practical Machine learning with Python** book by by [Dipanjan Sarkar](https://www.amazon.com/Dipanjan-Sarkar/e/B01M100PAR/ref%3Ddp_byline_cont_book_1) , [Raghav Bali](https://www.amazon.com/Raghav-Bali/e/B07K2PKCT9/ref%3Ddp_byline_cont_book_2) and [Tushar Sharma](https://www.amazon.com/s/ref%3Ddp_byline_sr_book_3?ie=UTF8&amp;field-author=Tushar%2BSharma&amp;search-alias=books&amp;text=Tushar%2BSharma&amp;sort=relevancerank)

[5]. <https://stackoverflow.com/>

[6]. [https://www.rdocumentation.org](https://www.rdocumentation.org/)

[7]. <https://www.analyticsvidhya.com/blog>