Santander customer transaction prediction

Submitted by

Mahesh.M

Table of Contents

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.

Data Details -

Provided with an anonymized dataset containing 200 numeric feature variables, the binary target column, and a string ID_code column.

Problem Analysis -

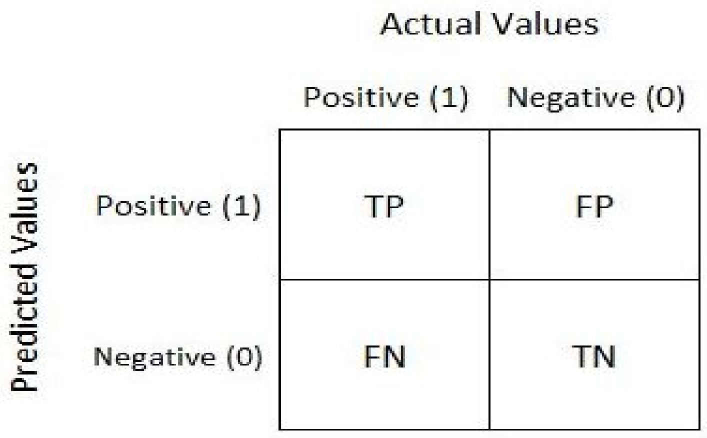
This is a binary classification problem under supervised machine learning algorithm. The task is to predict the value of target column in the test set.

Metrics -

This is a classification problem and we need to understand confusion matrix for getting evaluation metrics.

It is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.

It is extremely useful for measuring Recall, Precision, Specificity, Accuracy and most importantly AUC-ROC Curve.



Actual Values

Predicted Values	Positive (1)	Negative (0)
Positive (1)	TP	FP
Negative (0)	FN	TN

True Positive: You predicted positive and it's true.

True Negative: You predicted negative and it's true.

False Positive: (Type 1 Error) You predicted positive and it's false.

False Negative: (Type 2 Error) You predicted negative and it's false.

Based on confusion matrix we have following evaluation metrics

Recall- Out of all the positive classes, how much we predicted correctly. It should be high as possible.

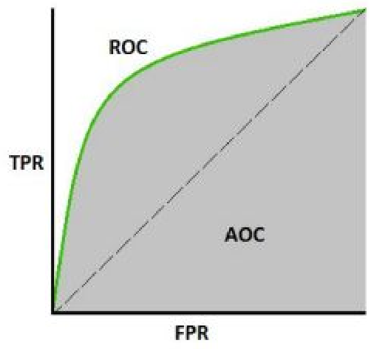
Precision- Out of all the classes, how much we predicted correctly. It should be high as possible.

F-measure- It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score.F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

Another important measure is AUC-ROC Score.

It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as **AUC-ROC** (Area Under the Receiver Operating Characteristics)

AUC-ROC curve is a performance measurement for classification problem at various thresholds settings.ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s.The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.



$$\text{TPR /Recall / Sensitivity} = \frac{TP}{TP + FN} \qquad \text{Specificity} = \frac{TN}{TN + FP} \qquad \text{FPR} = 1 - \text{Specificity} = \frac{FP}{TN + FP}$$

An excellent model has AUC near the 1 which means it has good measure of separability. A poor model has AUC near the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s. And when AUC is 0.5, it means model has no class separation capacity.

Exploratory Data Analysis

Check for variable data types in train and test data.

Id_code is object type, target is int type and 200 anonymous variables of float type.

Check for missing values

No missing values.

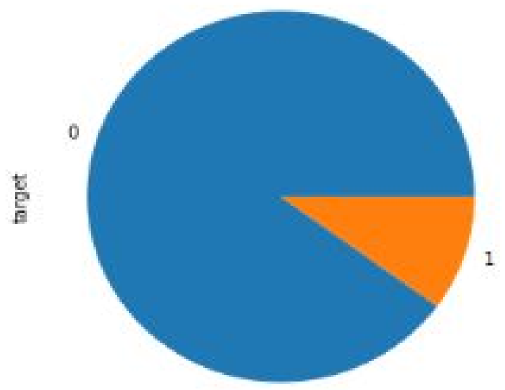
Shape of data 200000 observations with 202 columns in train data and 200000 observations with 201 columns in test data.

Check Balance of target column

0 179902

1 20098

Imbalanced Dataset.



Visualizations

Distribution of columns

See image (attached with submission) distribution of **columns.png**

Almost all features follow normalised distribution.

Distribution of all numerical features per each class.

See image (attached with submission) Distribution of columns per each class.

We can observe that there is a considerable number of features with significant different distribution for the two target values. For example, var_0, var_1, var_2, var_5, var_9, var_13, var_106, var_109, var_139 and many others. Also some features, like var_2, var_13, var_26, var_55, var_175, var_184, var_196 shows a distribution that resembles a bivariate distribution.

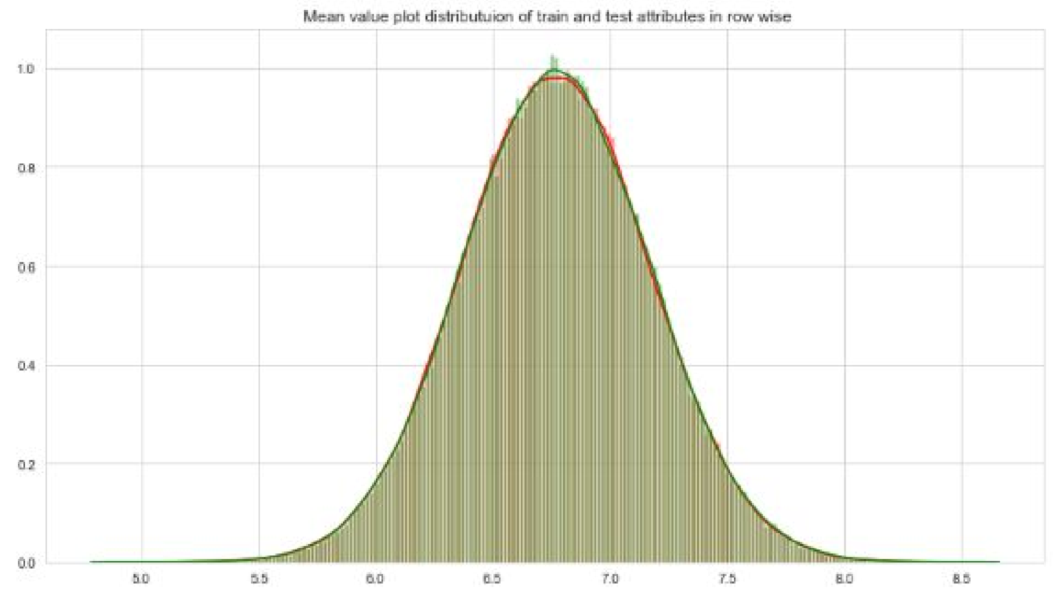
Distribution of numerical variables in train and test data.

See image (attached with submission) distribution of train and test **data.png**

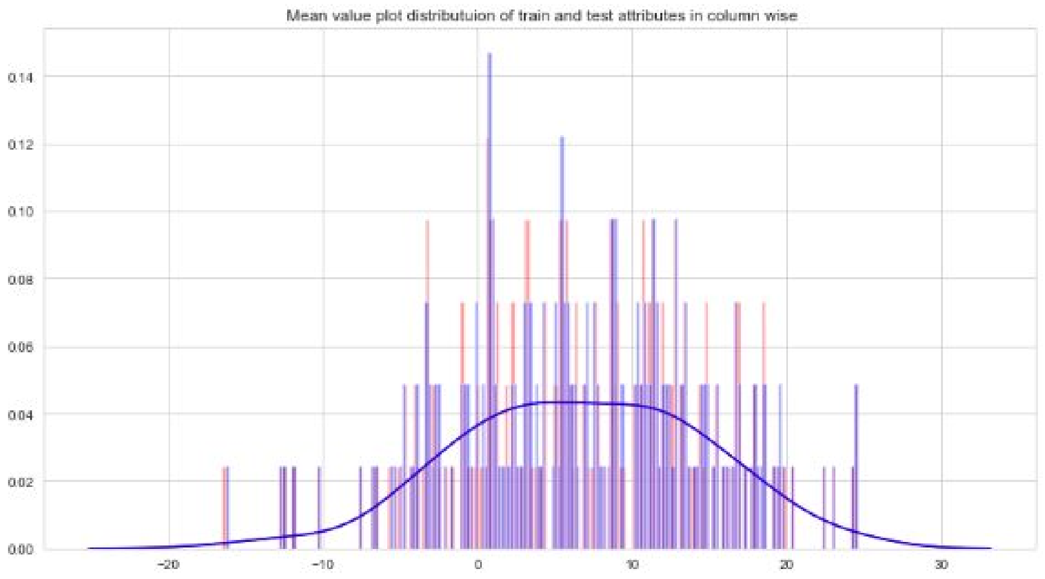
The train and test seems to be well balanced with respect to distribution of the numeric variables.

Mean value distribution of train and test attributes in dataset

Find the plot of mean value distribution train and test attributes at rowwise.

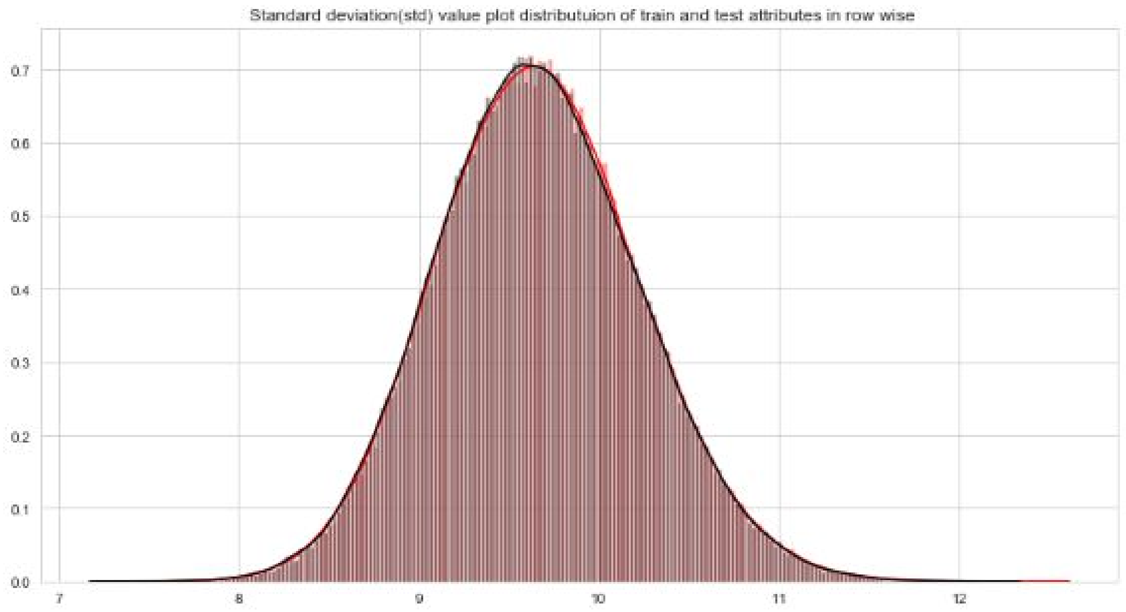


Find the plot of mean value distribution train and test attributes at column wise.

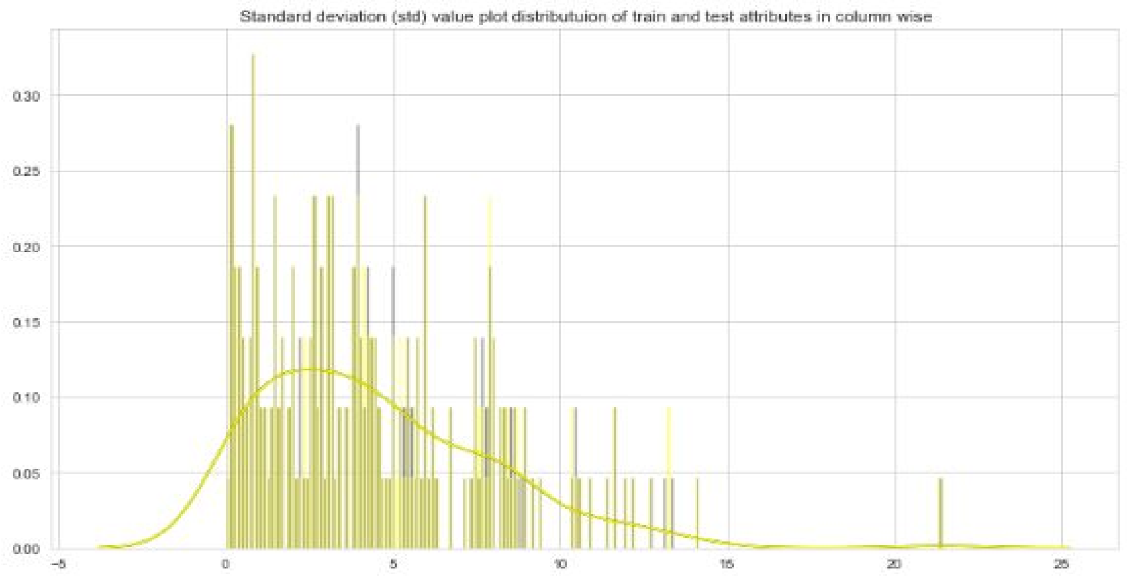


Standard deviation distribution of train and test attributes in dataset

Find the plot of (sd) distribution train and test attributes at rowwise.

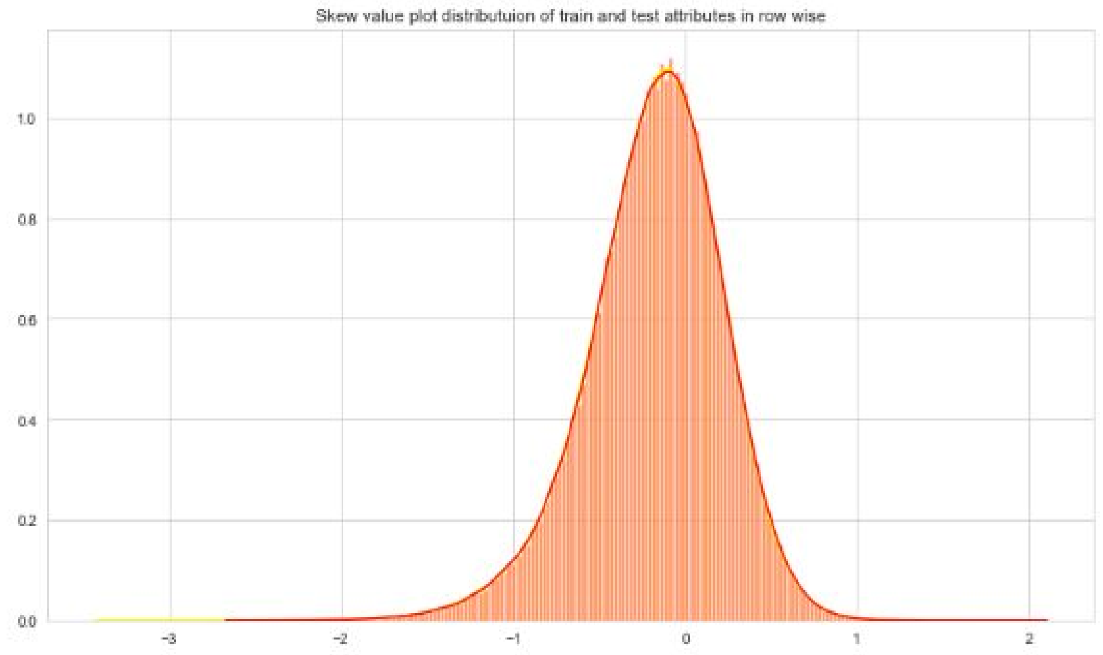


Find the plot of (sd) distribution train and test attributes at column wise.

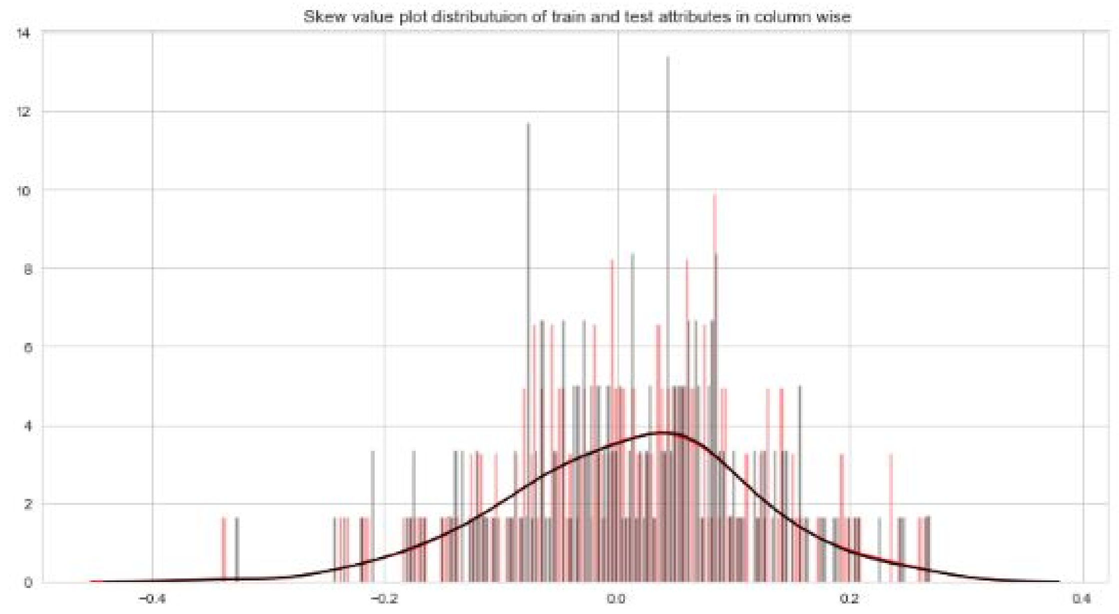


Skewness distribution of train and test attributes in dataset

Find the plot of skew distribution train and test attributes at rowwise.

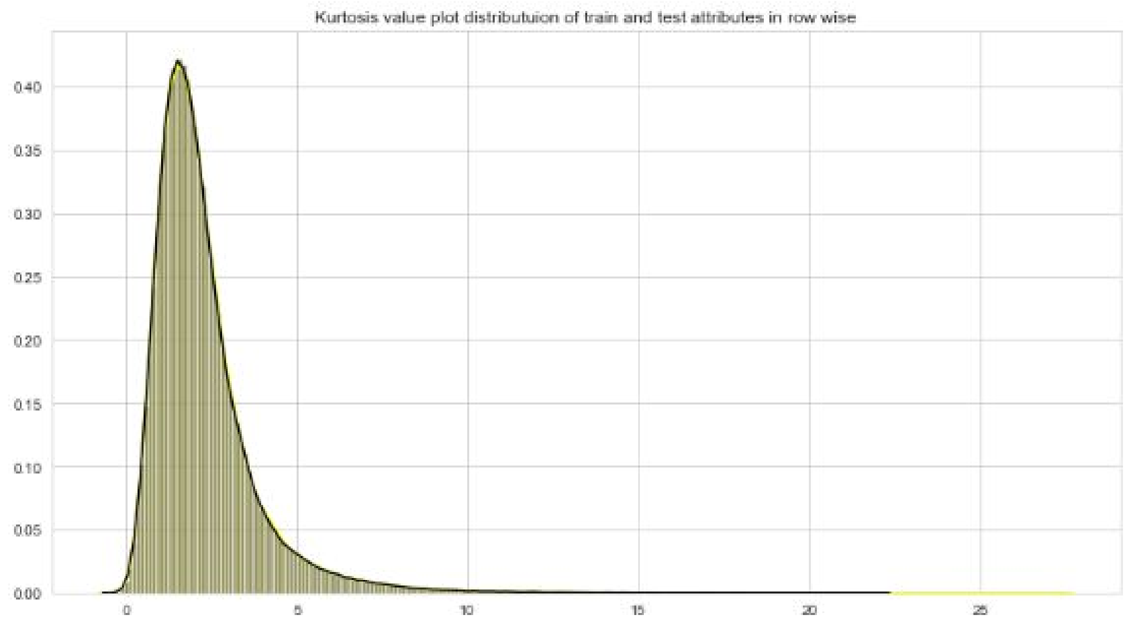


Find the plot of skew distribution train and test attributes at column wise.

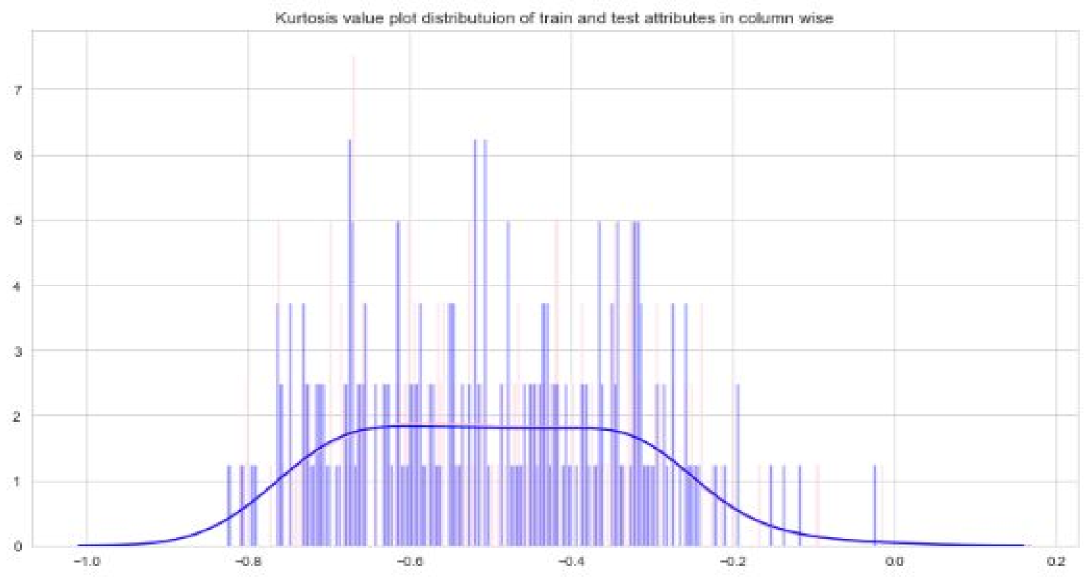


Kurtosis distribution of train and test attributes in dataset

Find the plot of kurtosis distribution train and test attributes at rowwise.



Find the plot of kurtosis distribution train and test attributes at column wise.



Outlier analysis:

I can see from the above plots, that both the training and test sets are very similar to one another and that the distributions do not have any anomalies.So i haven't performed outlier analysis due to the data is imbalanced and also it is not required for imbalanced data.

Feature Selection:

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

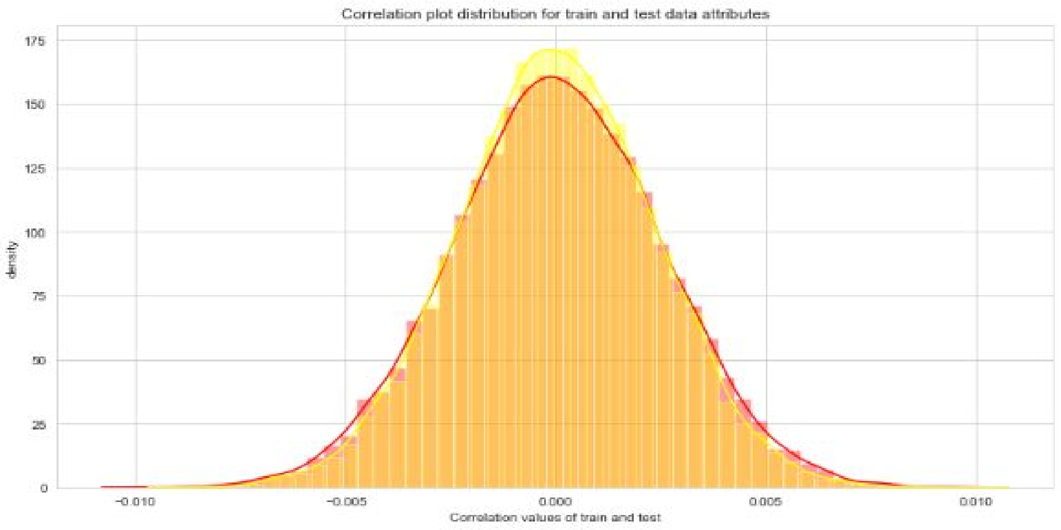
Correlation matrix

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

Correlation among numerical variables:

Minimum correlation among variables is -0.009844361358419677
Maximum correlation between variables is 0.009713658349534146

Correlation distribution plot of train and test attributes in dataset



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

Feature engineering

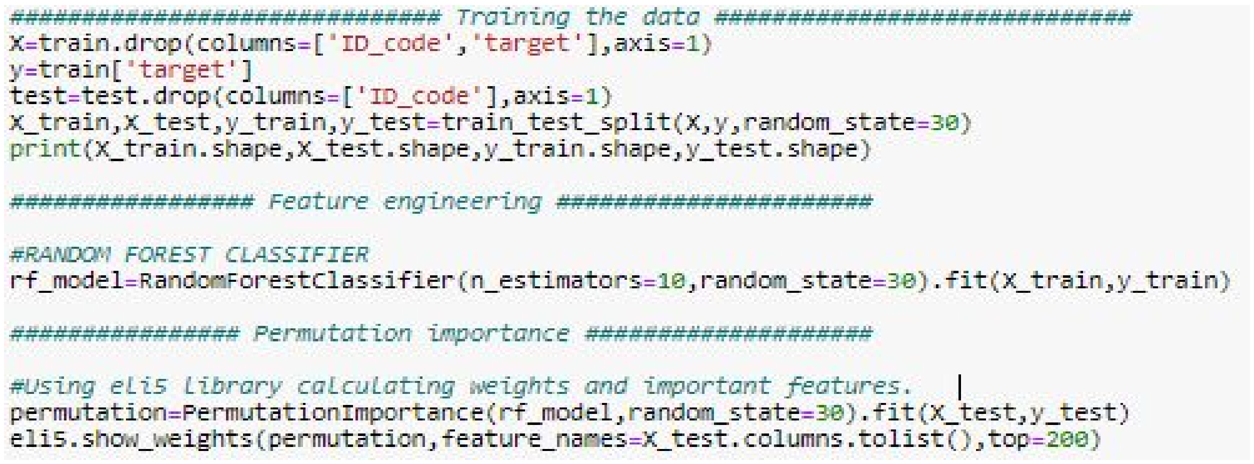
Here,I have proceeded with feature engineering by using

- Permutation importance
- Partial dependence plots

Permutation importance

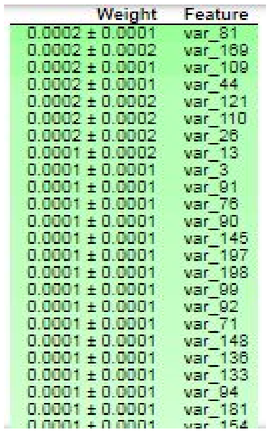
Permutation variable importance measure in a random forest for classification and regression. The variables which are mostly contributed to predict the model.

Python:



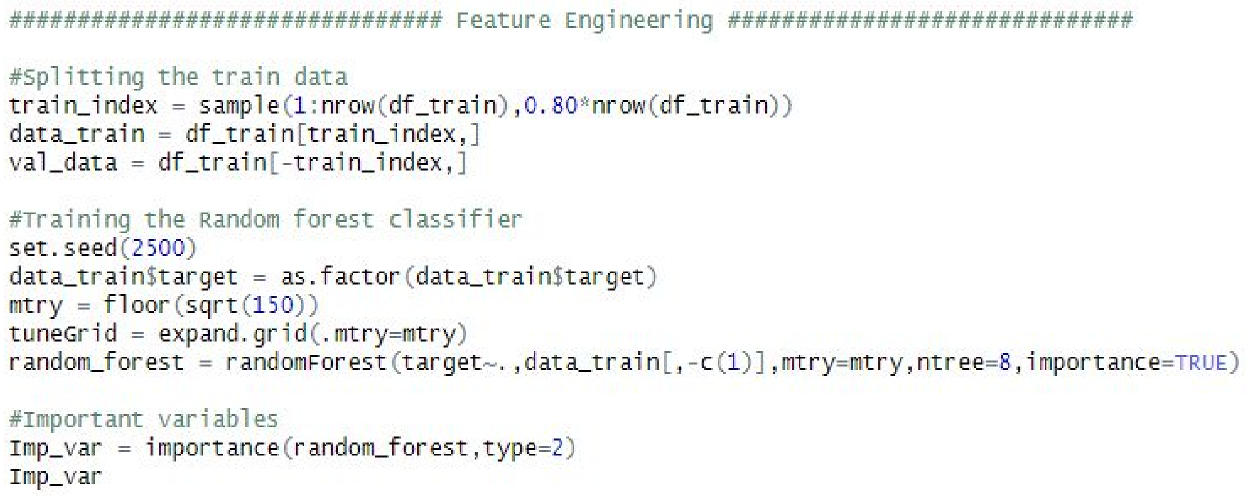
```
################################# Training the data ###############################
X=train.drop(columns=['ID_code','target'],axis=1)
y=train['target']
test=test.drop(columns=['ID_code'],axis=1)
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=30)
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)

################## Feature engineering ########################

#RANDOM FOREST CLASSIFIER
rf_model=RandomForestClassifier(n_estimators=10,random_state=30).fit(X_train,y_train)

################# Permutation importance ######################

#Using eli5 library calculating weights and important features.
permutation=PermutationImportance(rf_model,random_state=30).fit(X_test,y_test)
eli5.show_weights(permutation,feature_names=X_test.columns.tolist(),top=200)
```

Important variables



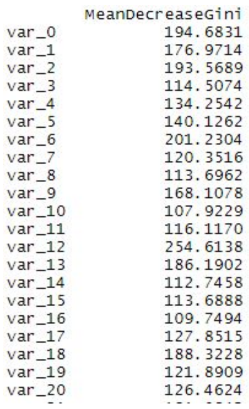
Weight	Feature
0.0002 ± 0.0001	var_81
0.0002 ± 0.0002	var_169
0.0002 ± 0.0001	var_109
0.0002 ± 0.0001	var_44
0.0002 ± 0.0002	var_121
0.0002 ± 0.0002	var_110
0.0002 ± 0.0002	var_26
0.0001 ± 0.0002	var_13
0.0001 ± 0.0001	var_3
0.0001 ± 0.0001	var_91
0.0001 ± 0.0001	var_76
0.0001 ± 0.0001	var_90
0.0001 ± 0.0001	var_145
0.0001 ± 0.0001	var_197
0.0001 ± 0.0001	var_198
0.0001 ± 0.0001	var_99
0.0001 ± 0.0001	var_92
0.0001 ± 0.0001	var_71
0.0001 ± 0.0001	var_148
0.0001 ± 0.0001	var_136
0.0001 ± 0.0001	var_133
0.0001 ± 0.0001	var_94
0.0001 ± 0.0001	var_181
0.0001 ± 0.0001	var_154

R:



```
################################# Feature Engineering ###############################

#Splitting the train data
train_index = sample(1:nrow(df_train),0.80*nrow(df_train))
data_train = df_train[train_index,]
val_data = df_train[-train_index,]

#Training the Random forest classifier
set.seed(2500)
data_train$target = as.factor(data_train$target)
mtry = floor(sqrt(150))
tuneGrid = expand.grid(.mtry=mtry)
random_forest = randomForest(target~.,data_train[,-c(1)],mtry=mtry,ntree=8,importance=TRUE)

#Important variables
Imp_var = importance(random_forest,type=2)
Imp_var
```

Variable importance based on Mean Decrease Gini

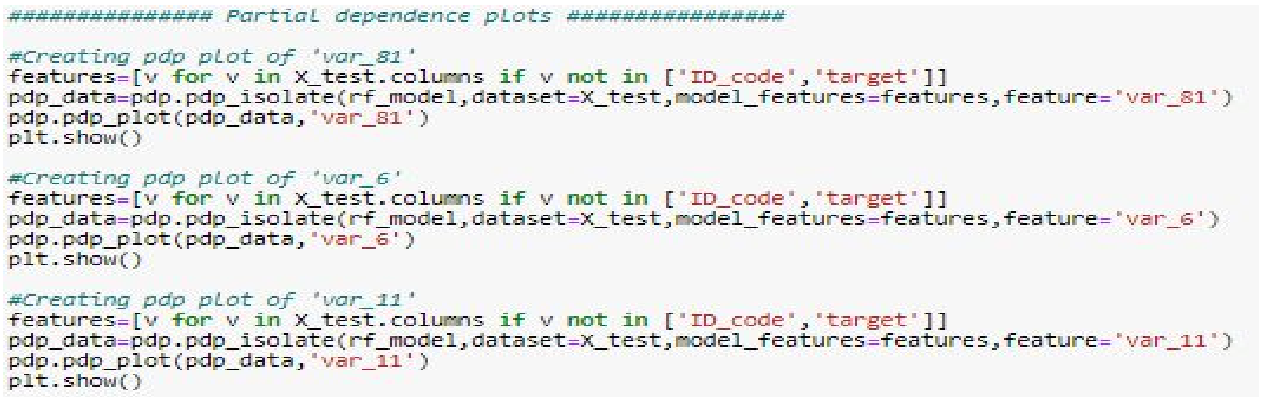


```
       MeanDecreaseGini
var_0          194.6831
var_1          176.9714
var_2          193.5689
var_3          114.5074
var_4          134.2542
var_5          140.1262
var_6          201.2304
var_7          120.3516
var_8          113.6962
var_9          168.1078
var_10         107.9229
var_11         116.1170
var_12         254.6138
var_13         186.1902
var_14         112.7458
var_15         113.6888
var_16         109.7494
var_17         127.8515
var_18         188.3228
var_19         121.8909
var_20         126.4624
```

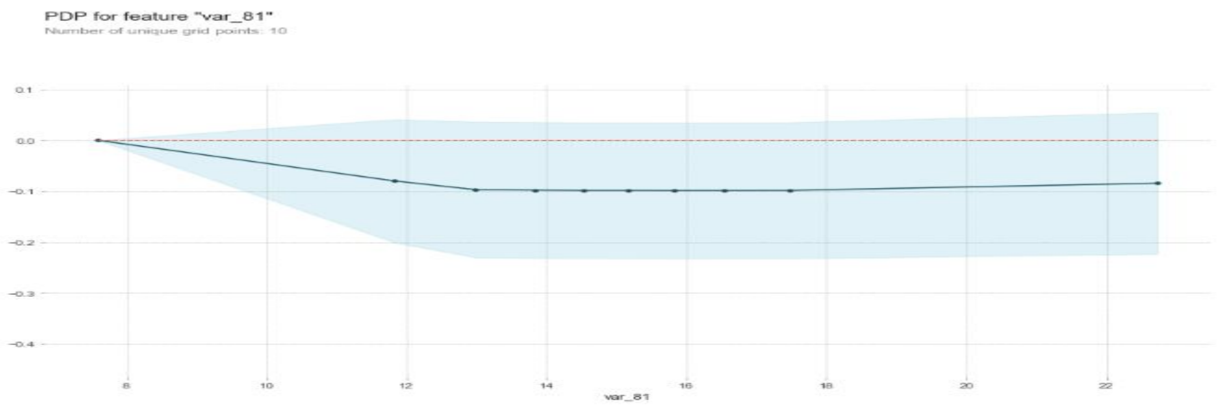
Partial dependence plots

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

Python:

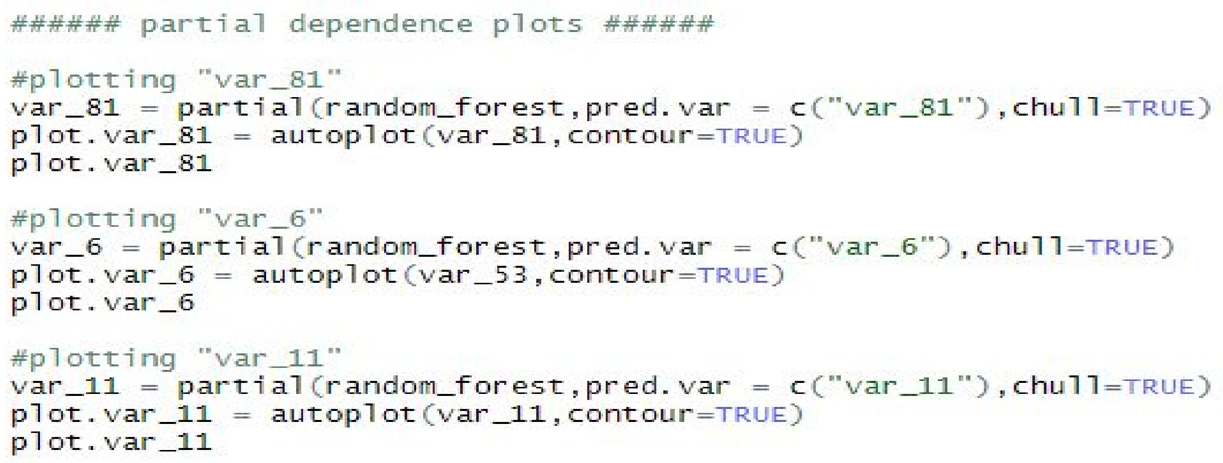


```python
################ Partial dependence plots #################

#Creating pdp plot of 'var_81'
features=[v for v in X_test.columns if v not in ['ID_code','target']]
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_test,model_features=features,feature='var_81')
pdp.pdp_plot(pdp_data,'var_81')
plt.show()

#Creating pdp plot of 'var_6'
features=[v for v in X_test.columns if v not in ['ID_code','target']]
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_test,model_features=features,feature='var_6')
pdp.pdp_plot(pdp_data,'var_6')
plt.show()

#Creating pdp plot of 'var_11'
features=[v for v in X_test.columns if v not in ['ID_code','target']]
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_test,model_features=features,feature='var_11')
pdp.pdp_plot(pdp_data,'var_11')
plt.show()
```

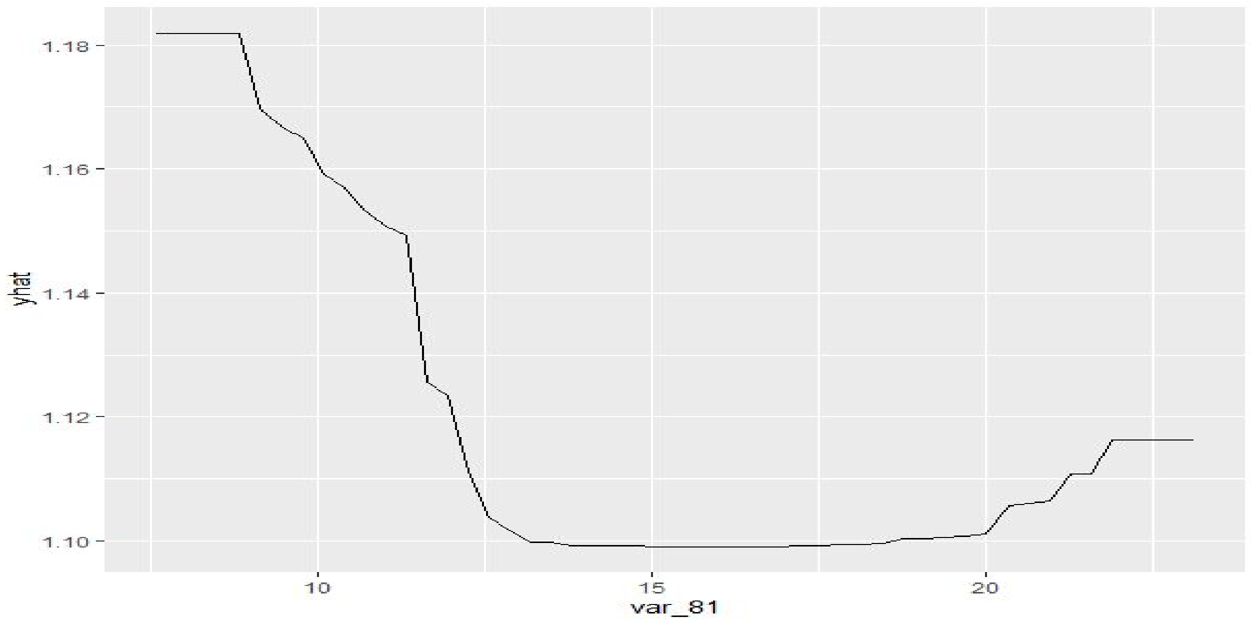


- The blue shaded area indicates the level of confidence of 'var_81'.
- On y-axis,if our value is +ve it means for that particular value of predictor variable,it is less likely to predict the correct class.While,having a positive value it shows that value has a positive impact on predicting the correct class.
- The y_axis does not show the predictor value instead it shows how the value changes with the change in given predictor variable.

R:



```
###### partial dependence plots ######

#plotting "var_81"
var_81 = partial(random_forest,pred.var = c("var_81"),chull=TRUE)
plot.var_81 = autoplot(var_81,contour=TRUE)
plot.var_81

#plotting "var_6"
var_6 = partial(random_forest,pred.var = c("var_6"),chull=TRUE)
plot.var_6 = autoplot(var_53,contour=TRUE)
plot.var_6

#plotting "var_11"
var_11 = partial(random_forest,pred.var = c("var_11"),chull=TRUE)
plot.var_11 = autoplot(var_11,contour=TRUE)
plot.var_11
```

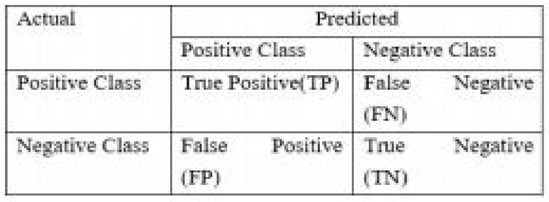


Dealing Imbalanced dataset

Before modelling for this dataset let us understand how to deal with imbalanced dataset for classification problem.

Traditional Machine Learning algorithms tend to produce unsatisfactory classifiers when faced with imbalanced datasets.For any imbalanced data set, if the event to be predicted belongs to the minority class and the event rate is less than 10%, it is usually referred to as a rare event. The conventional model evaluation methods do not accurately measure model performance when faced with imbalanced datasets. Standard classifier algorithms like Decision Tree,have a bias towards classes which have large number of instances.They tend to only predict the majority class data.

The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class. Evaluation of a classification algorithm performance is measured by the Confusion Matrix which contains information about the actual and the predicted class.



Actual	Predicted	
	Positive Class	Negative Class
Positive Class	True Positive(TP)	False Negative (FN)
Negative Class	False Positive (FP)	True Negative (TN)

Accuracy of a model = (TP+TN) / (TP+FN+FP+TN)
However,while working in an imbalanced domain accuracy is not an appropriate measure to evaluate model performance.Hence we need evaluation metrics such as Recall, Precision, F1_score, AUC-ROC score along with Accuracy.

How to deal with these imbalanced datasets?

- Changing the performance metric.
- Oversampling the minority class.
- Under sampling the majority class.
- Synthetic Minority Oversampling Technique(SMOTE) in Python and Random Oversampling Examples(ROSE) in R.
- Changing the algorithm.

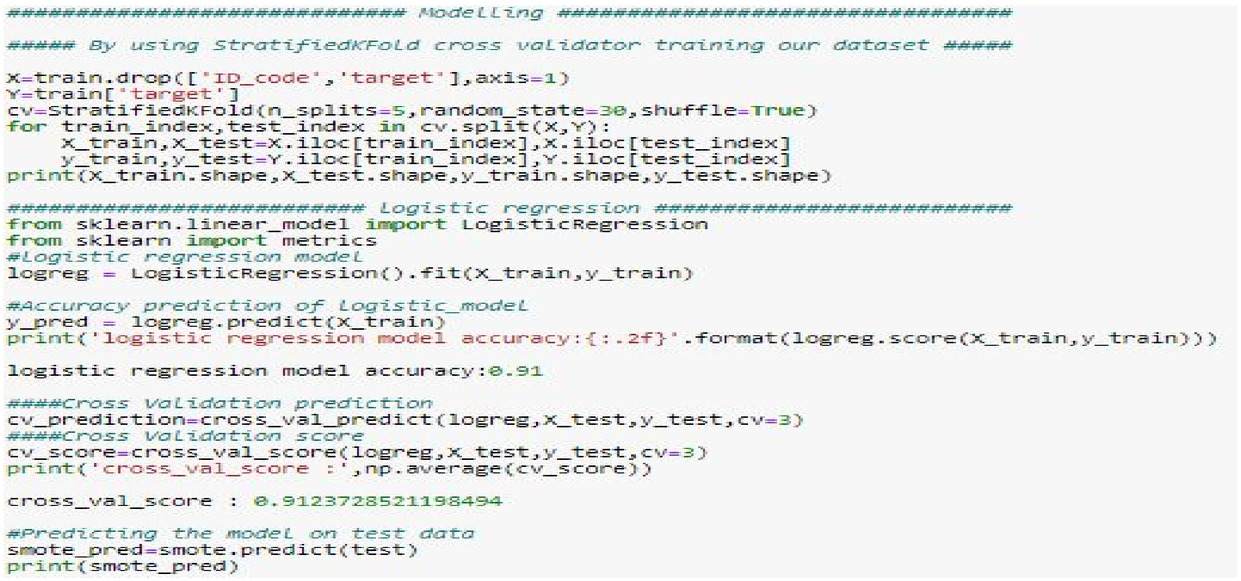
Logistic Regression:

➢ Using Logistic Regression to predict the values of our target variable.

This is the classification problem.We can use logistic regression for this problem.

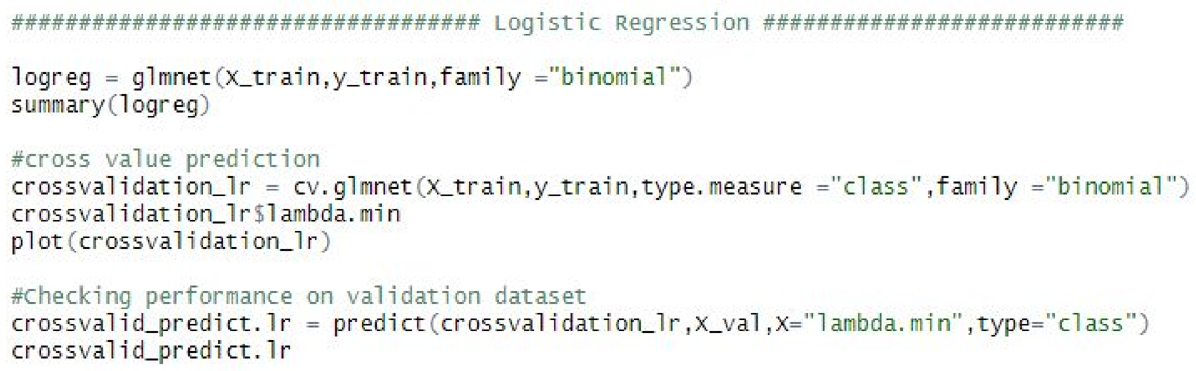
Logistic Regression is used when the dependent variable(target) is categorical. It has a bias towards classes which have large number of instances. It tends to only predict the majority class data.The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of misclassification of the minority class as compared to the majority class

Python:



```
################################# Modelling ####################################

##### By using StratifiedKFold cross validator training our dataset #####

X=train.drop(['ID_code','target'],axis=1)
Y=train['target']
cv=StratifiedKFold(n_splits=5,random_state=30,shuffle=True)
for train_index,test_index in cv.split(X,Y):
    X_train,X_test=X.iloc[train_index],X.iloc[test_index]
    y_train,y_test=Y.iloc[train_index],Y.iloc[test_index]
print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)

############################# Logistic regression ##############################
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
#Logistic regression model
logreg = LogisticRegression().fit(X_train,y_train)

#Accuracy prediction of Logistic_model
y_pred = logreg.predict(X_train)
print('logistic regression model accuracy:{:.2f}'.format(logreg.score(X_train,y_train)))

logistic regression model accuracy:0.91

####Cross Validation prediction
cv_prediction=cross_val_predict(logreg,X_test,y_test,cv=3)
####Cross Validation score
cv_score=cross_val_score(logreg,X_test,y_test,cv=3)
print('cross_val_score :',np.average(cv_score))

cross_val_score : 0.9123728521198494

#Predicting the model on test data
smote_pred=smote.predict(test)
print(smote_pred)
```

R:



```
################################### Logistic Regression ###########################

logreg = glmnet(X_train,y_train,family ="binomial")
summary(logreg)

#cross value prediction
crossvalidation_lr = cv.glmnet(X_train,y_train,type.measure ="class",family ="binomial")
crossvalidation_lr$lambda.min
plot(crossvalidation_lr)

#Checking performance on validation dataset
crossvalid_predict.lr = predict(crossvalidation_lr,X_val,X="lambda.min",type="class")
crossvalid_predict.lr
```

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

Oversampling the minority class

Over-Sampling increases the number of instances in the minority class by randomly replicating them in order to present a higher representation of the minority class in the sample.

➢ **Advantages**
- Unlike under sampling this method leads to no information loss.
- Outperforms under sampling

➢ **Disadvantages**
- It increases the likelihood of overfitting since it replicates the minority class events.

Under sampling the majority class

Undersampling aims to balance class distribution by randomly eliminating majority class examples. This is done until the majority and minority class instances are balanced out.

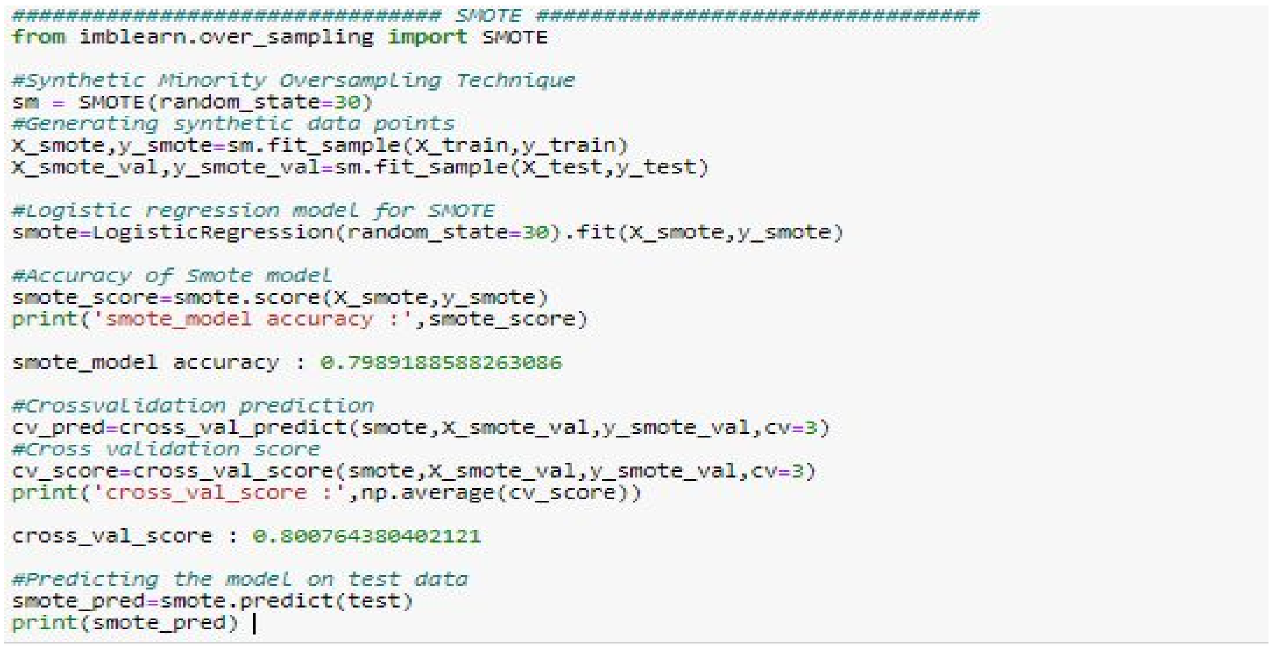
➢ **Advantages**
- It can help improve run time and storage problems by reducing the number of training data samples when the training data set is huge.

➢ **Disadvantages**
- It can discard potentially useful information which could be important for building rule classifiers.
- The sample chosen by random under sampling may be a biased sample. And it will not be an accurate representation of the population. Thereby, resulting in inaccurate results with the actual test data set.

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:

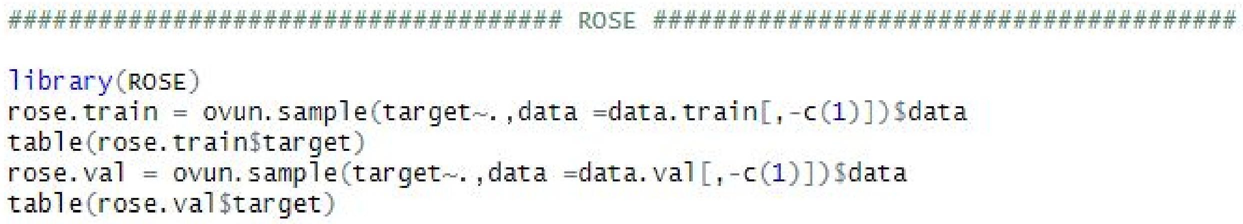


```
################################## SMOTE ##################################
from imblearn.over_sampling import SMOTE

#Synthetic Minority Oversampling Technique
sm = SMOTE(random_state=30)
#Generating synthetic data points
X_smote,y_smote=sm.fit_sample(X_train,y_train)
X_smote_val,y_smote_val=sm.fit_sample(X_test,y_test)

#Logistic regression model for SMOTE
smote=LogisticRegression(random_state=30).fit(X_smote,y_smote)

#Accuracy of Smote model
smote_score=smote.score(X_smote,y_smote)
print('smote_model accuracy :',smote_score)

smote_model accuracy : 0.7989188588263086

#Crossvalidation prediction
cv_pred=cross_val_predict(smote,X_smote_val,y_smote_val,cv=3)
#Cross validation score
cv_score=cross_val_score(smote,X_smote_val,y_smote_val,cv=3)
print('cross_val_score :',np.average(cv_score))

cross_val_score : 0.800764380402121

#Predicting the model on test data
smote_pred=smote.predict(test)
print(smote_pred)
```

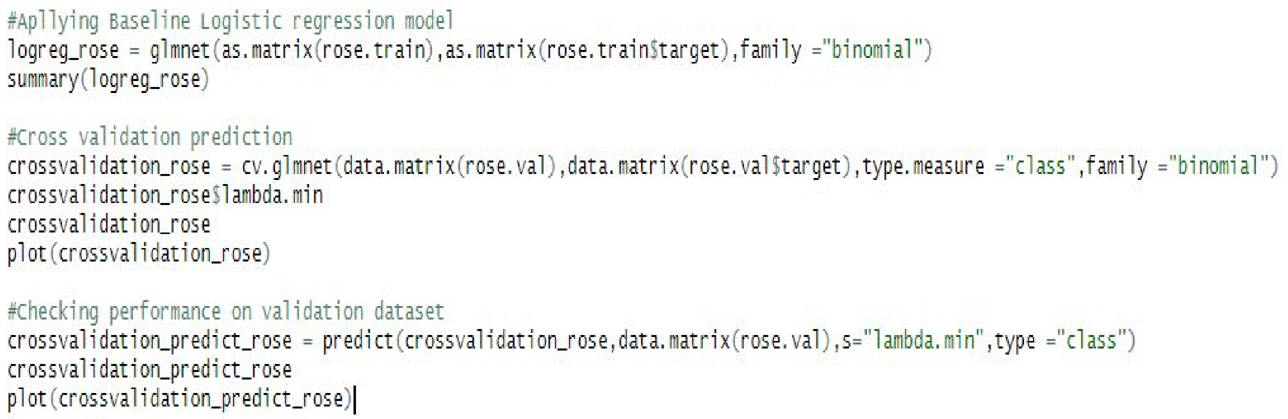
R:

Random Oversampling Examples (ROSE)

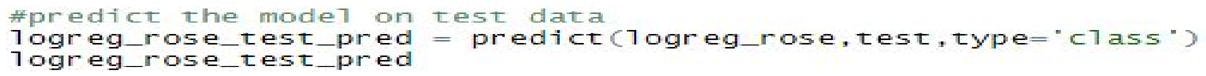
It creates a sample of synthetic data by enlarging the features space of minority and majority class examples. In order to balance imbalanced data we are going to use SMOTE sampling method.



```
##################################### ROSE #####################################

library(ROSE)
rose.train = ovun.sample(target~.,data =data.train[,-c(1)])$data
table(rose.train$target)
rose.val = ovun.sample(target~.,data =data.val[,-c(1)])$data
table(rose.val$target)
```



```
#Apllying Baseline Logistic regression model
logreg_rose = glmnet(as.matrix(rose.train),as.matrix(rose.train$target),family ="binomial")
summary(logreg_rose)

#Cross validation prediction
crossvalidation_rose = cv.glmnet(data.matrix(rose.val),data.matrix(rose.val$target),type.measure ="class",family ="binomial")
crossvalidation_rose$lambda.min
crossvalidation_rose
plot(crossvalidation_rose)

#Checking performance on validation dataset
crossvalidation_predict_rose = predict(crossvalidation_rose,data.matrix(rose.val),s="lambda.min",type ="class")
crossvalidation_predict_rose
plot(crossvalidation_predict_rose)
```

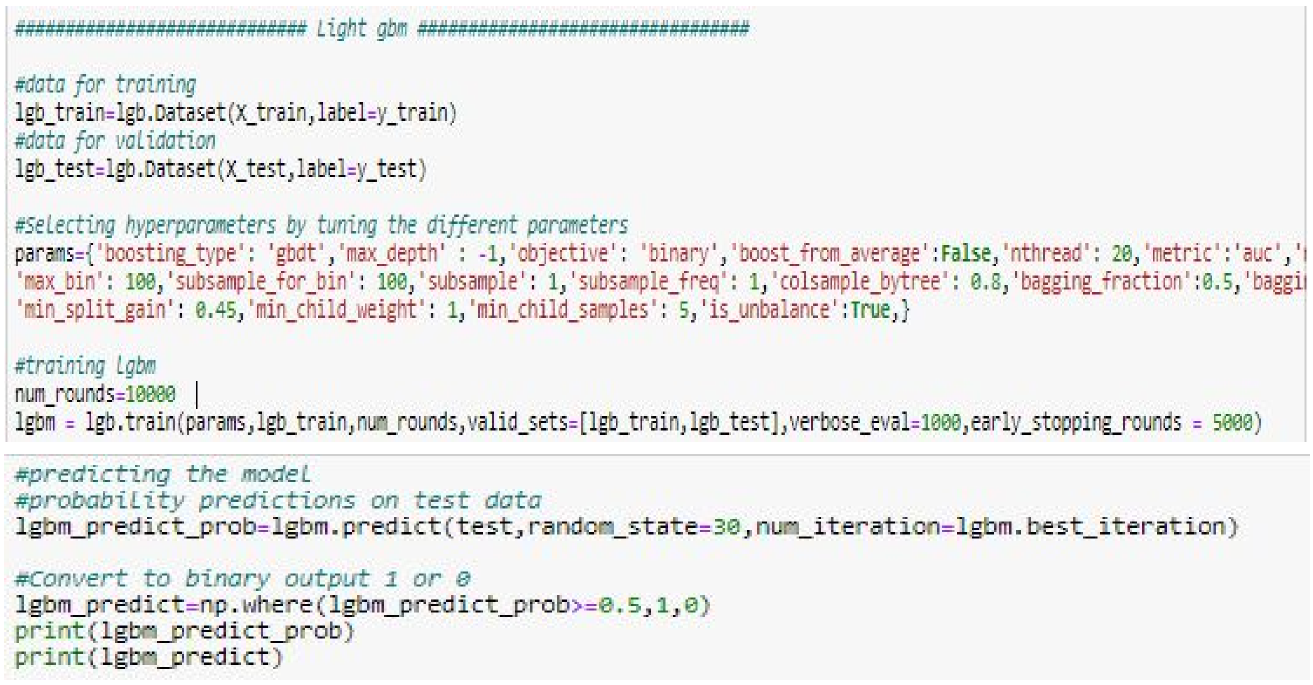


```
#predict the model on test data
logreg_rose_test_pred = predict(logreg_rose,test,type='class')
logreg_rose_test_pred
```

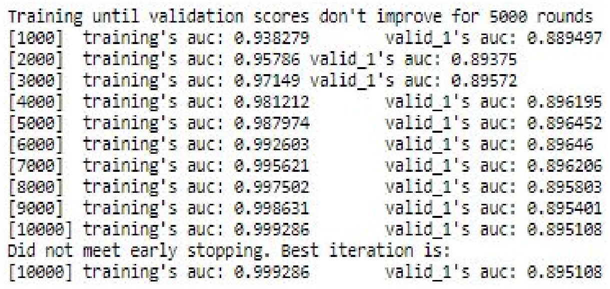
LightGBM

LightGBM is Gradient Boosting ensemble model which is faster in speed and accuracy as compared to bagging and adaptive boosting. It is capable of performing equally well with large datasets with a significant reduction in training time as compared to XGBOOST. But parameter tuning in LightGBM should be done carefully

Python:



```
############################### Light gbm ###################################

#data for training
lgb_train=lgb.Dataset(X_train,label=y_train)
#data for validation
lgb_test=lgb.Dataset(X_test,label=y_test)

#Selecting hyperparameters by tuning the different parameters
params={'boosting_type': 'gbdt','max_depth' : -1,'objective': 'binary','boost_from_average':False,'nthread': 20,'metric':'auc','
'max_bin': 100,'subsample_for_bin': 100,'subsample': 1,'subsample_freq': 1,'colsample_bytree': 0.8,'bagging_fraction':0.5,'baggi
'min_split_gain': 0.45,'min_child_weight': 1,'min_child_samples': 5,'is_unbalance':True,}

#training lgbm
num_rounds=10000
lgbm = lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_test],verbose_eval=1000,early_stopping_rounds = 5000)

#predicting the model
#probability predictions on test data
lgbm_predict_prob=lgbm.predict(test,random_state=30,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)
```

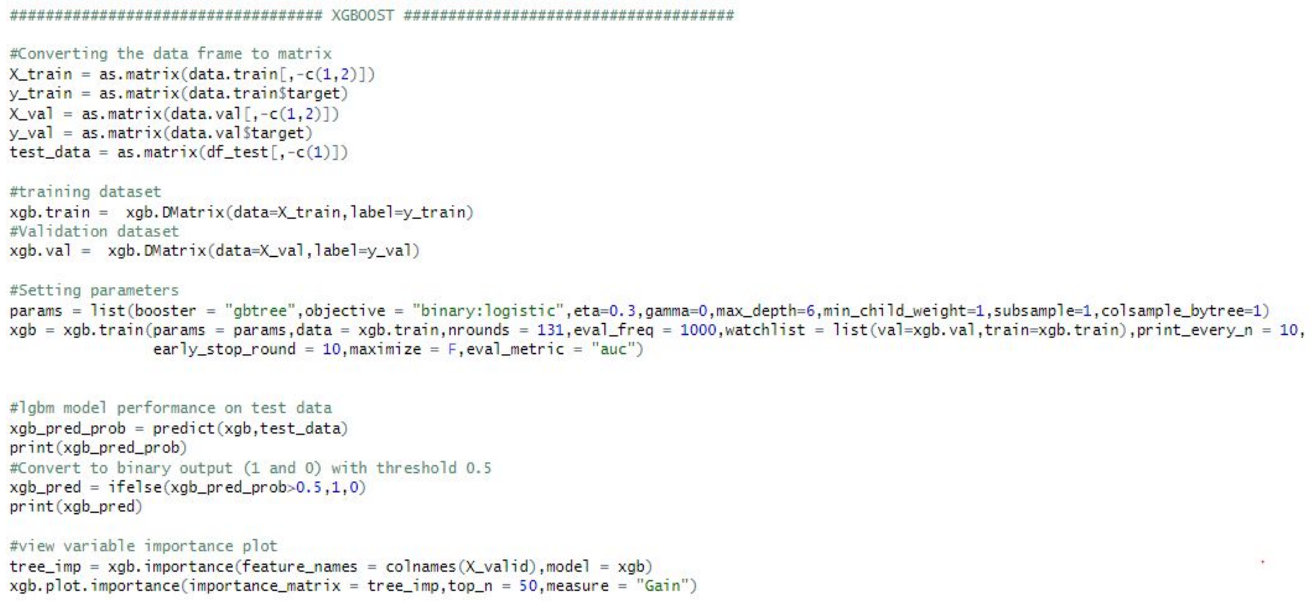


```
Training until validation scores don't improve for 5000 rounds
[1000]  training's auc: 0.938279        valid_1's auc: 0.889497
[2000]  training's auc: 0.95786 valid_1's auc: 0.89375
[3000]  training's auc: 0.97149 valid_1's auc: 0.89572
[4000]  training's auc: 0.981212        valid_1's auc: 0.896195
[5000]  training's auc: 0.987974        valid_1's auc: 0.896452
[6000]  training's auc: 0.992603        valid_1's auc: 0.89646
[7000]  training's auc: 0.995621        valid_1's auc: 0.896206
[8000]  training's auc: 0.997502        valid_1's auc: 0.895803
[9000]  training's auc: 0.998631        valid_1's auc: 0.895401
[10000] training's auc: 0.999286        valid_1's auc: 0.895108
Did not meet early stopping. Best iteration is:
[10000] training's auc: 0.999286        valid_1's auc: 0.895108
```

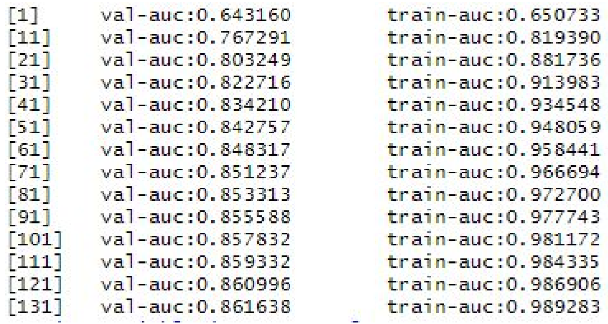
R:

XGBOOST

XGBoost is a decision-tree-based ensemble algorithm that uses a gradient boosting framework.In prediction problems involving unstructured data,tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class.



```
#################################### XGBOOST ######################################

#Converting the data frame to matrix
X_train = as.matrix(data.train[,-c(1,2)])
y_train = as.matrix(data.train$target)
X_val = as.matrix(data.val[,-c(1,2)])
y_val = as.matrix(data.val$target)
test_data = as.matrix(df_test[,-c(1)])

#training dataset
xgb.train =  xgb.DMatrix(data=X_train,label=y_train)
#Validation dataset
xgb.val =  xgb.DMatrix(data=X_val,label=y_val)

#Setting parameters
params = list(booster = "gbtree",objective = "binary:logistic",eta=0.3,gamma=0,max_depth=6,min_child_weight=1,subsample=1,colsample_bytree=1)
xgb = xgb.train(params = params,data = xgb.train,nrounds = 131,eval_freq = 1000,watchlist = list(val=xgb.val,train=xgb.train),print_every_n = 10,
               early_stop_round = 10,maximize = F,eval_metric = "auc")


#lgbm model performance on test data
xgb_pred_prob = predict(xgb,test_data)
print(xgb_pred_prob)
#Convert to binary output (1 and 0) with threshold 0.5
xgb_pred = ifelse(xgb_pred_prob>0.5,1,0)
print(xgb_pred)

#view variable importance plot
tree_imp = xgb.importance(feature_names = colnames(X_valid),model = xgb)
xgb.plot.importance(importance_matrix = tree_imp,top_n = 50,measure = "Gain")
```



```
[1]     val-auc:0.643160        train-auc:0.650733
[11]    val-auc:0.767291        train-auc:0.819390
[21]    val-auc:0.803249        train-auc:0.881736
[31]    val-auc:0.822716        train-auc:0.913983
[41]    val-auc:0.834210        train-auc:0.934548
[51]    val-auc:0.842757        train-auc:0.948059
[61]    val-auc:0.848317        train-auc:0.958441
[71]    val-auc:0.851237        train-auc:0.966694
[81]    val-auc:0.853313        train-auc:0.972700
[91]    val-auc:0.855588        train-auc:0.977743
[101]   val-auc:0.857832        train-auc:0.981172
[111]   val-auc:0.859332        train-auc:0.984335
[121]   val-auc:0.860996        train-auc:0.986906
[131]   val-auc:0.861638        train-auc:0.989283
```

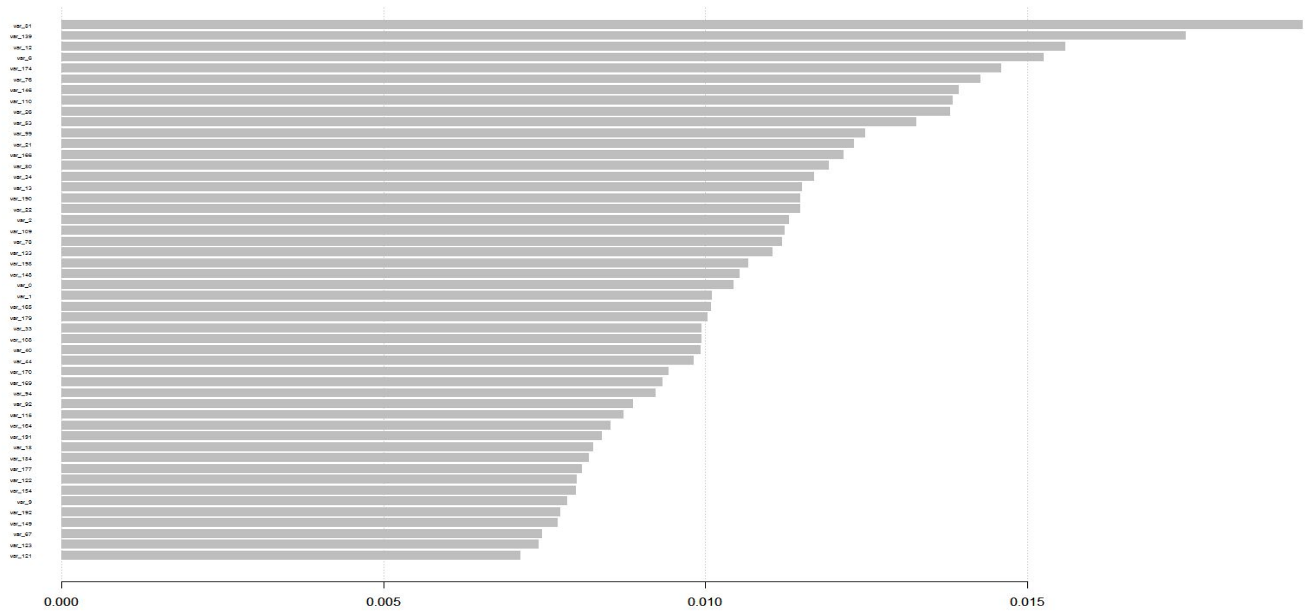
Python:

Plot of important features:

```
#plot the important features of lgbm
lgb.plot_importance(lgbm,max_num_features=50,importance_type="split",figsize=(30,50))
```

R:

```
#important features plot
tree_imp = xgb.importance(feature_names = colnames(X_train),model = xgb)
xgb.plot.importance(importance_matrix = tree_imp,top_n = 50,measure = "Gain")
```



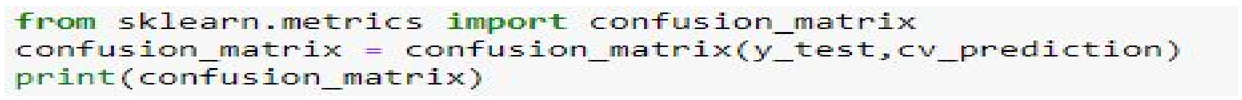
Model Evaluation

We have used 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.Here I had used two metrics for model evaluation as follows.

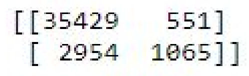
Confusion Matrix: - It is a technique for summarizing the performance of a classification algorithm.The number of correct predictions and incorrect predictions are summarized with count values and broken down by each class.
Roc_auc_score :- It is a metric that computes the area under the Roc curve and 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

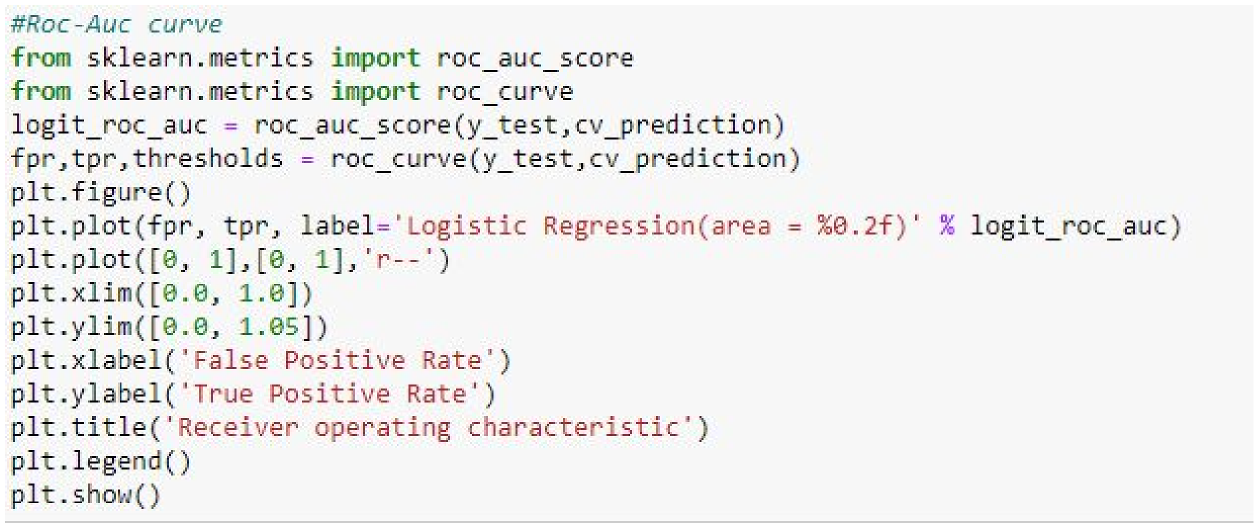
Python:



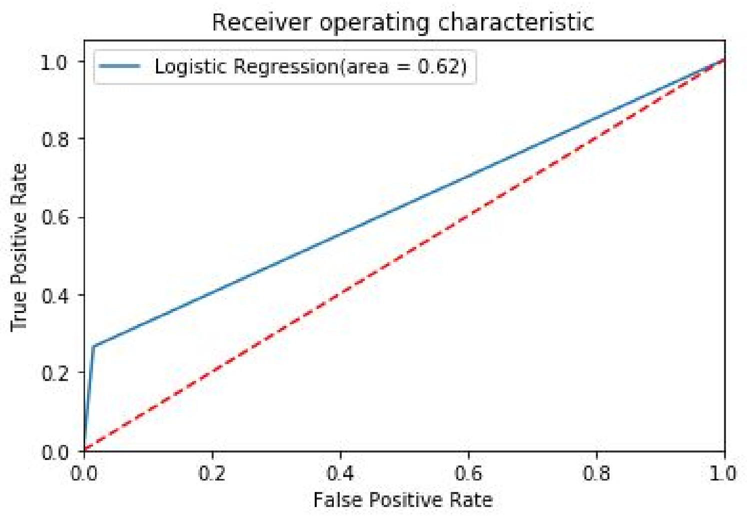
```
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_test,cv_prediction)
print(confusion_matrix)
```



```
[[35429   551]
 [ 2954  1065]]
```



```
#Roc-Auc curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test,cv_prediction)
fpr,tpr,thresholds = roc_curve(y_test,cv_prediction)
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression(area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1],[0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend()
plt.show()
```



While comparing roc_auc_score and cross validation score,we can conclude that the model is not performing well on imbalanced data.

Classification report:

```
#Classification report
from sklearn.metrics import classification_report
print(classification_report(y_test,cv_prediction))
```

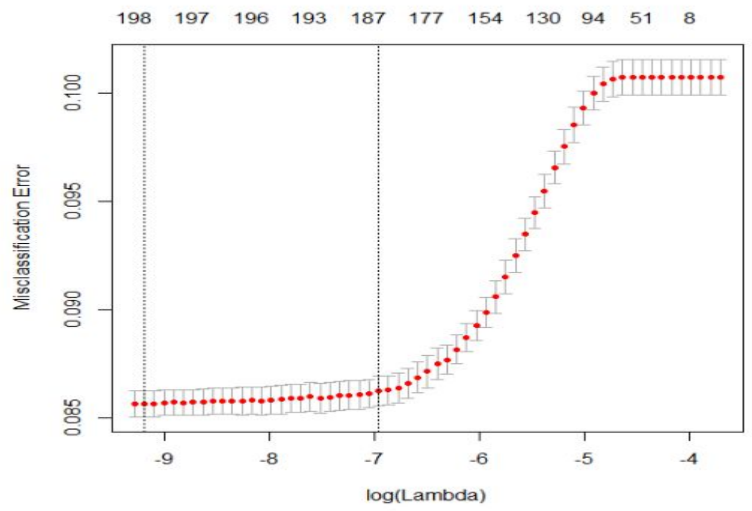
```
              precision    recall  f1-score   support

           0       0.92      0.98      0.95     35980
           1       0.66      0.26      0.38      4019

    accuracy                           0.91     39999
   macro avg       0.79      0.62      0.67     39999
weighted avg       0.90      0.91      0.90     39999
```

F1 score is high for number of customers,it shows those customers will not make a transaction then who will make a transaction.So,changing the algorithm would be a better choice.

Logistic Regression

R:

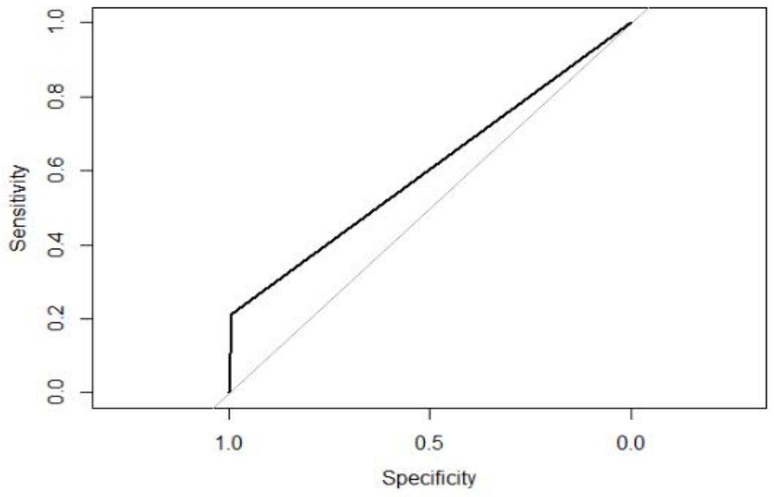
```
#cross value prediction
crossvalidation_lr = cv.glmnet(X_train,y_train,type.measure ="class",family ="binomial")
crossvalidation_lr$lambda.min
plot(crossvalidation_lr)
```



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

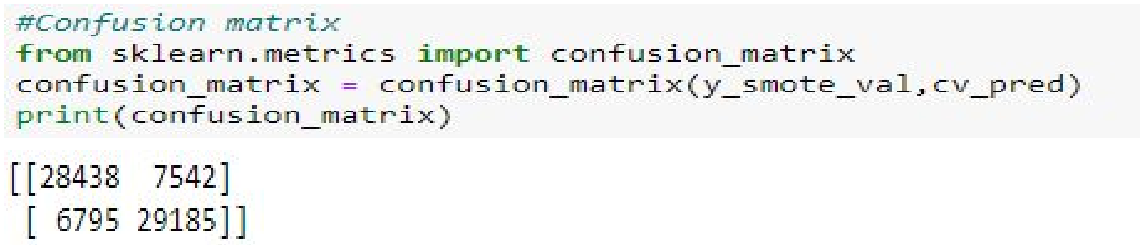
```
###### Confusion matrix ######
confusionMatrix(data=crossvalid_predict.lr,reference=target)

Confusion Matrix and Statistics

          Reference
Prediction     1     2
         1 35696  3133
         2   325   846

               Accuracy : 0.9136
                 95% CI : (0.9108, 0.9163)
    No Information Rate : 0.9005
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.2967

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9910
            Specificity : 0.2126
         Pos Pred Value : 0.9193
         Neg Pred Value : 0.7225
             Prevalence : 0.9005
         Detection Rate : 0.8924
   Detection Prevalence : 0.9707
      Balanced Accuracy : 0.6018

       'Positive' Class : 1
```

```
########## ROC_AUC ###########
crossvalid_predict.lr = as.numeric(crossvalid_predict.lr)
roc(data=data.val[,c(2)],response=target,predictor=crossvalid_predict.lr,plot=TRUE,auc=TRUE)
```



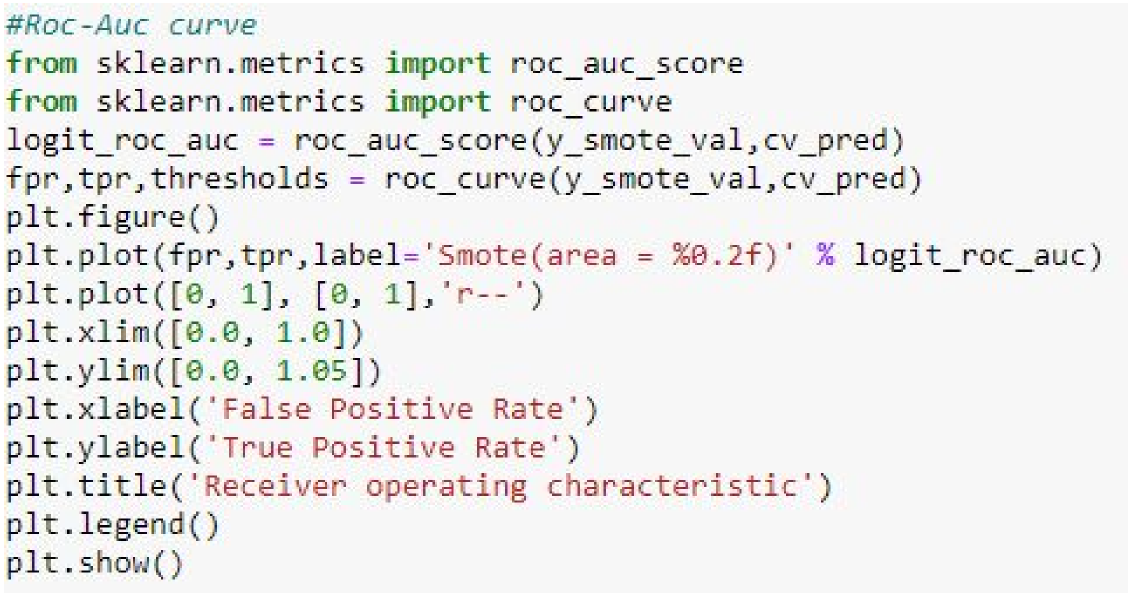
Synthetic Minority Oversampling Technique (SMOTE)

Python:

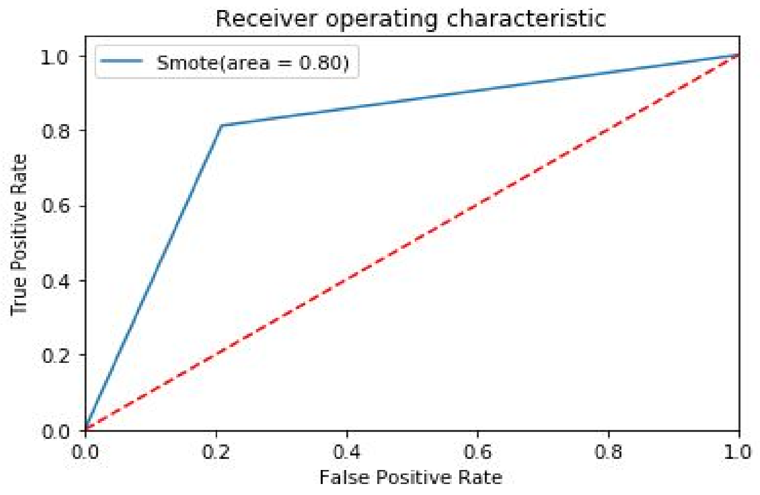


```
#Confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix = confusion_matrix(y_smote_val,cv_pred)
print(confusion_matrix)
```

```
[[28438  7542]
 [ 6795 29185]]
```



```
#Roc-Auc curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_smote_val,cv_pred)
fpr,tpr,thresholds = roc_curve(y_smote_val,cv_pred)
plt.figure()
plt.plot(fpr,tpr,label='Smote(area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend()
plt.show()
```



Classification report:

```
from sklearn.metrics import classification_report
print(classification_report(y_smote_val,cv_pred))
```

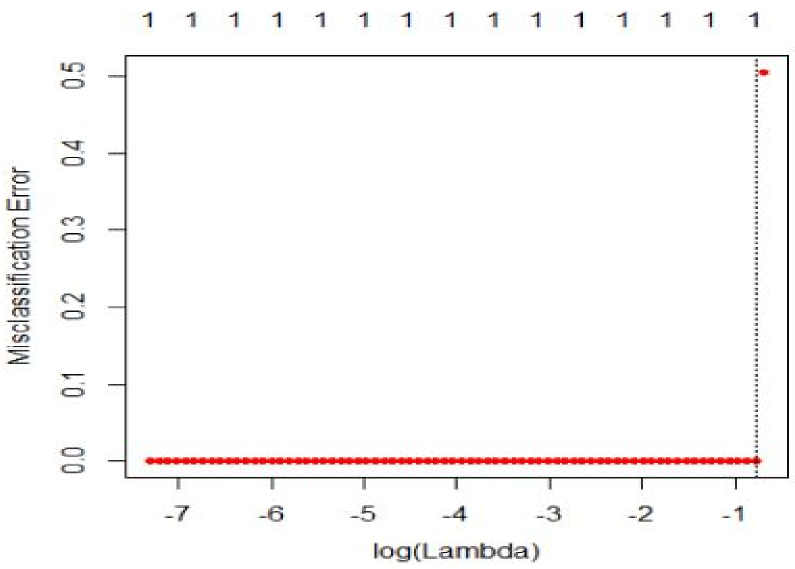
```
              precision    recall  f1-score   support

           0       0.81      0.79      0.80     35980
           1       0.79      0.81      0.80     35980

    accuracy                           0.80     71960
   macro avg       0.80      0.80      0.80     71960
weighted avg       0.80      0.80      0.80     71960
```

We can see that smote model is performing well on imbalance data compared to logistic regression.

Random Oversampling Examples (ROSE)

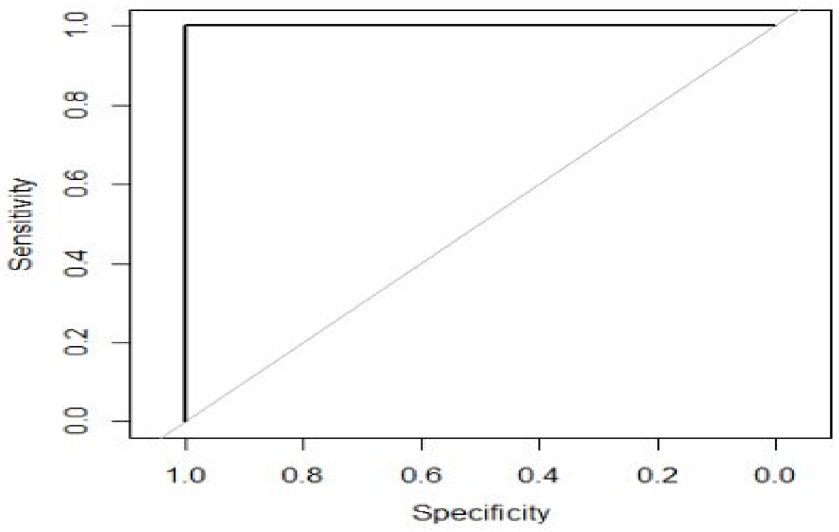
R:

```
#Cross validation prediction
crossvalidation_rose = cv.glmnet(data.matrix(rose.val),data.matrix(rose.val$target),type.measure ="class",family ="binomial")
crossvalidation_rose$lambda.min
crossvalidation_rose
plot(crossvalidation_rose)
```



```
###### Confusion matrix ######
confusionMatrix(data=crossvalidation_predict_rose,reference=target)

Confusion Matrix and Statistics

          Reference
Prediction     1     2
         1 19988     0
         2     0 20012

               Accuracy : 1
                 95% CI : (0.9999, 1)
    No Information Rate : 0.5003
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.4997
         Detection Rate : 0.4997
   Detection Prevalence : 0.4997
      Balanced Accuracy : 1.0000

       'Positive' Class : 1
```

```
########## ROC_AUC ###########
crossvalidation_predict_rose = as.numeric(crossvalidation_predict_rose)
roc(data=data.val[,c(2)],response=target,predictor=crossvalidation_predict_rose,plot=TRUE,auc=TRUE)
```



By changing count of one target class variable. Finally we got area under the ROC curve is 1 but that is not 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. LightGBM model in python performed well on imbalanced data.
4. XGBOOST in r is performed well on imbalanced data.

Finally LightGBM in Python and XGBOOST in R,is the best choice for identifying the customers who will make a specific transaction in the future, irrespective of the amount of money transacted.

Further Improvements

Further improvements in model can be done by-

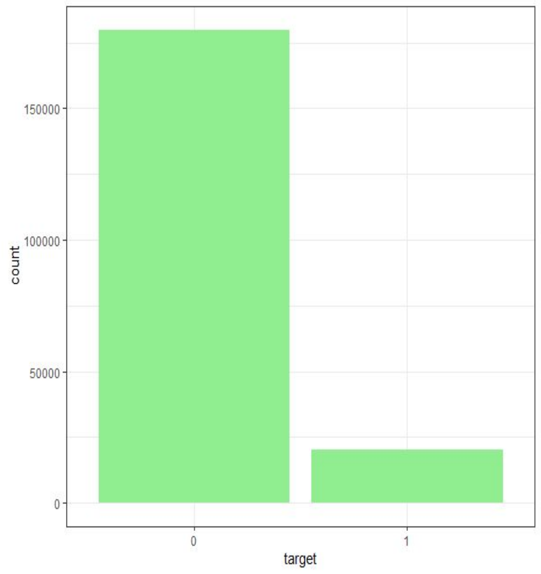
- Using Parallel Processing with LightGBM Algorithm.
- Selecting important features and then modelling them.
- Using Stratified Folding for train and test splits.
- Using Parallel Processing with LightGBM Algorithm

Conclusion

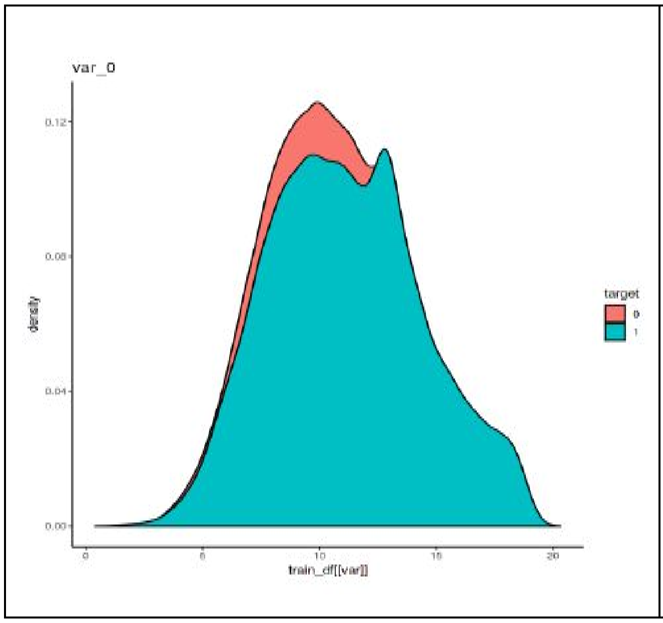
This was a classification problem on a typically unbalanced dataset with no missing values. Predictor variables are anonymous and numeric and target variable is categorical.Visualising descriptive features and finally I got to know that these variables are not correlated among themselves. After that I decided to treat imbalanced dataset and built different models with original data and choosen LightGBM as my final model then using the same model with feature engineered data we got AUC-Score of 0.8951.

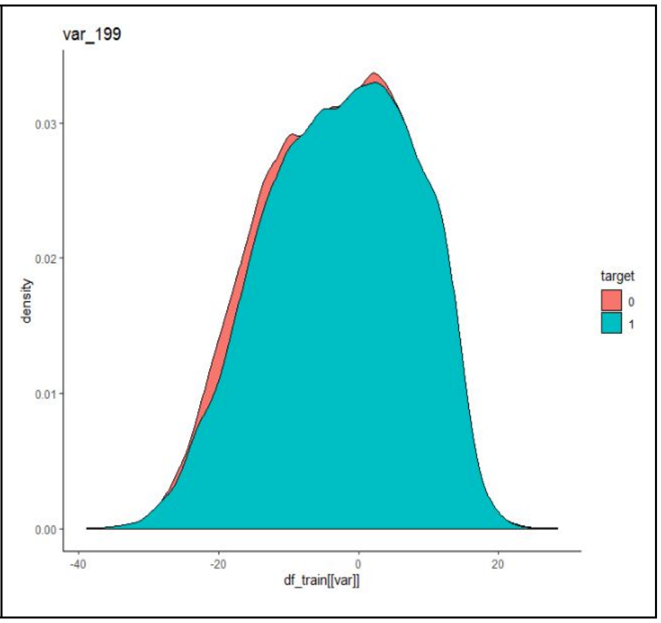
Visualizations - ggplot2 -R

Target classes count

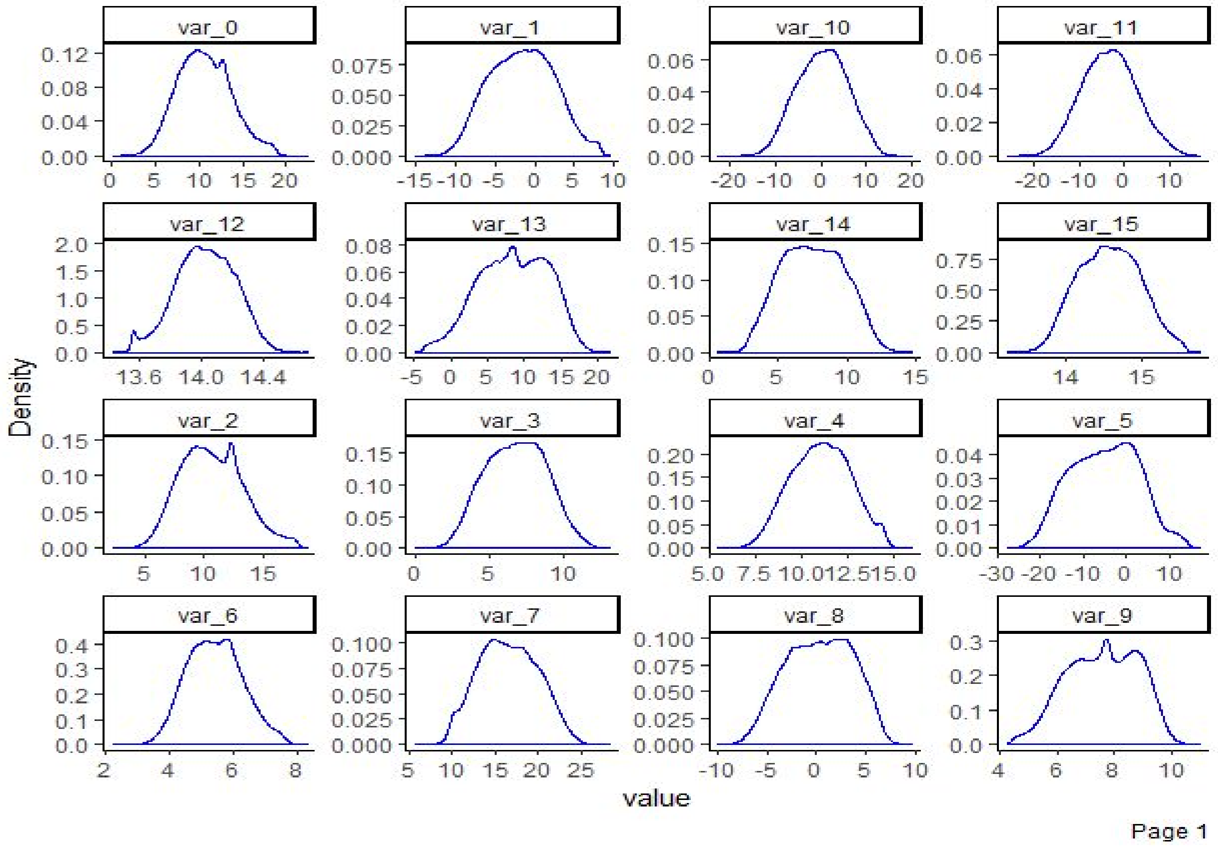


Distribution of train attributes

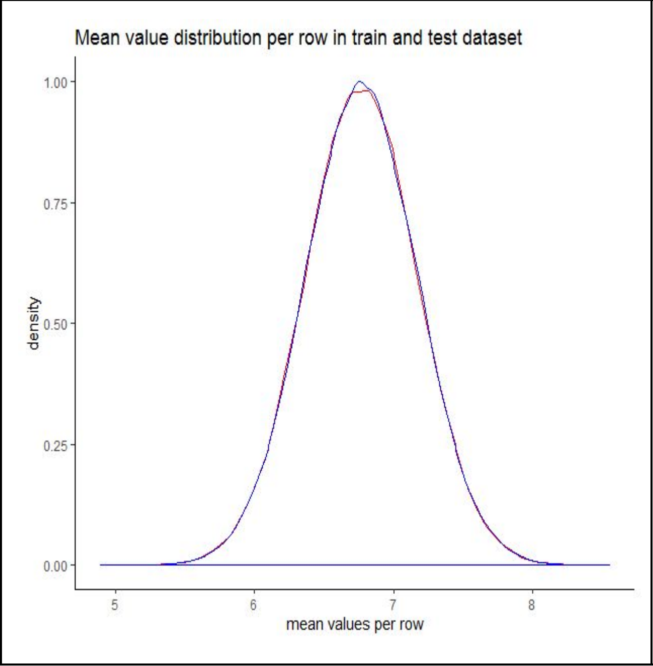


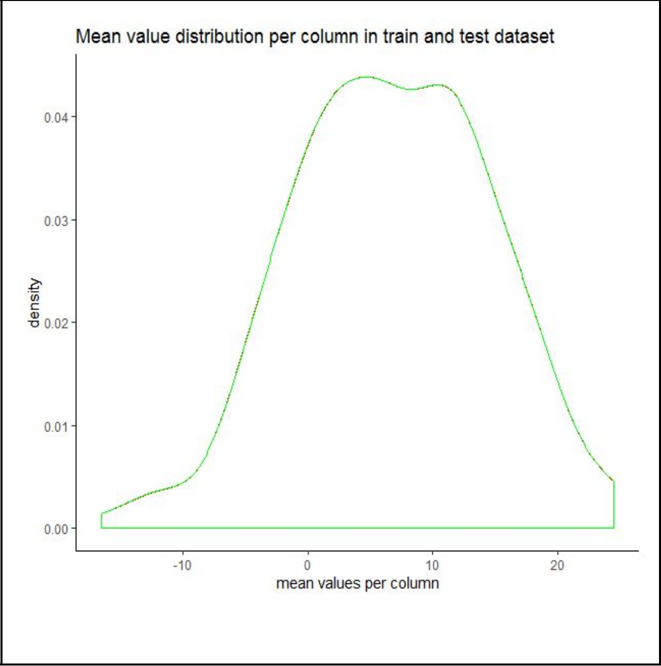


Distribution of test attributes

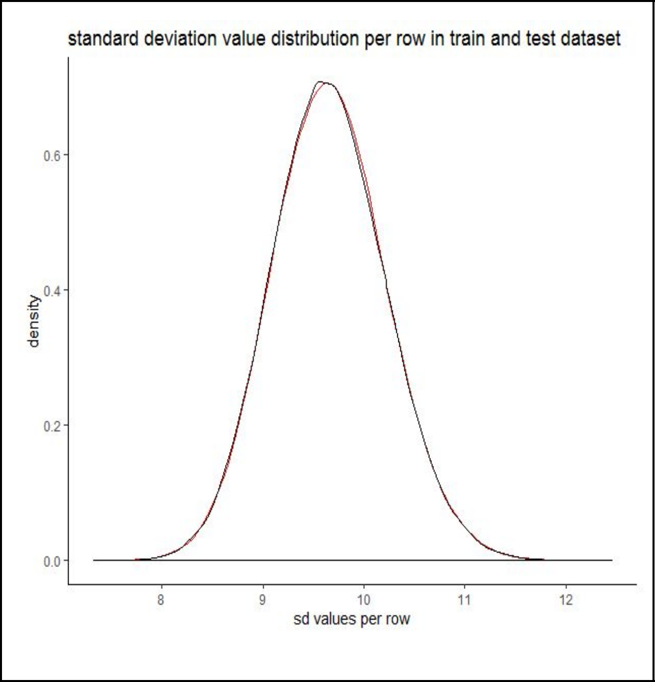


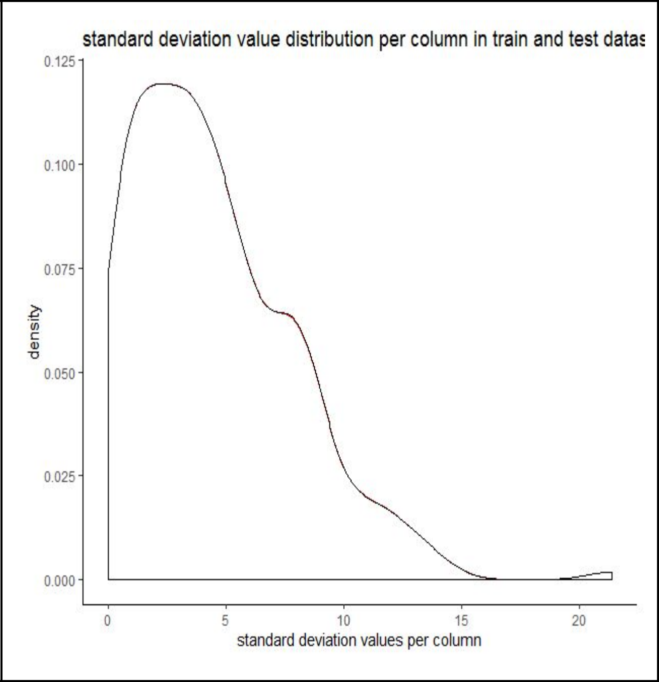
Mean value distribution of train and test attributes in dataset



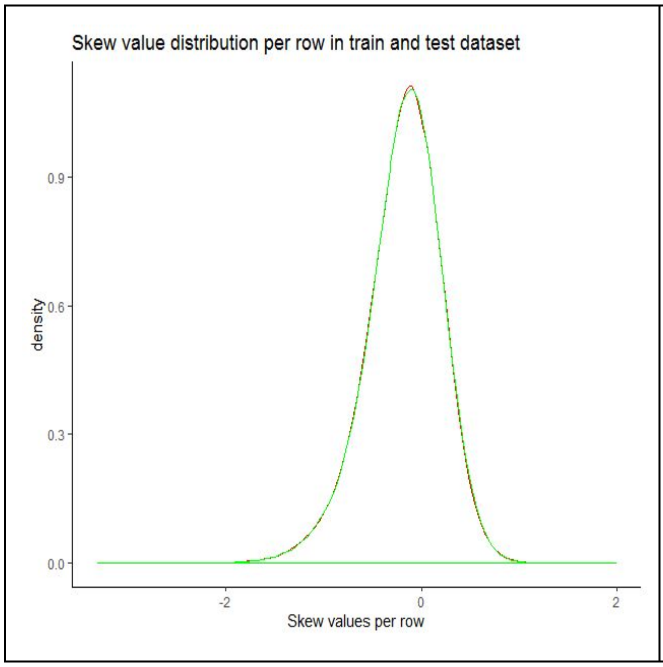


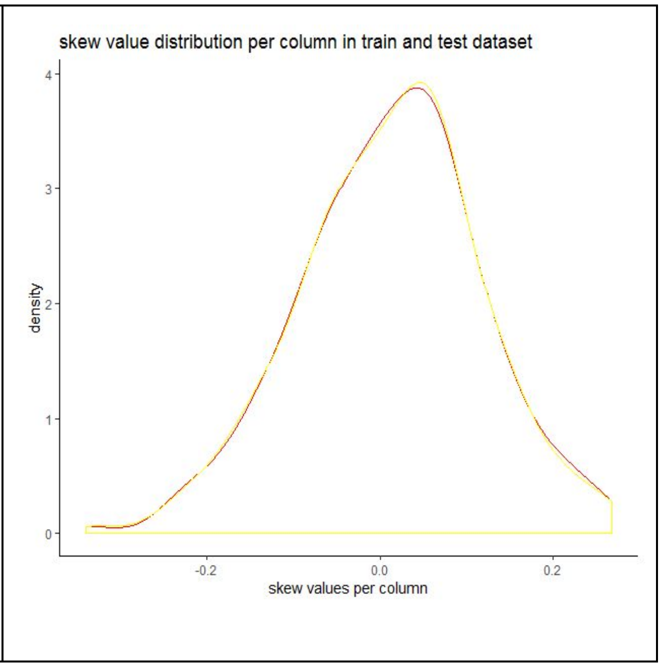
Standard Deviation value distribution of train and test attributes in dataset



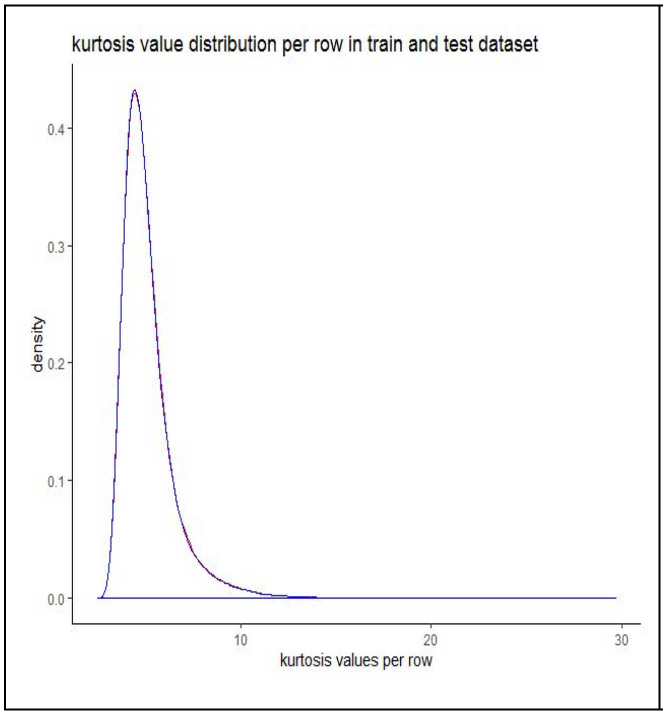


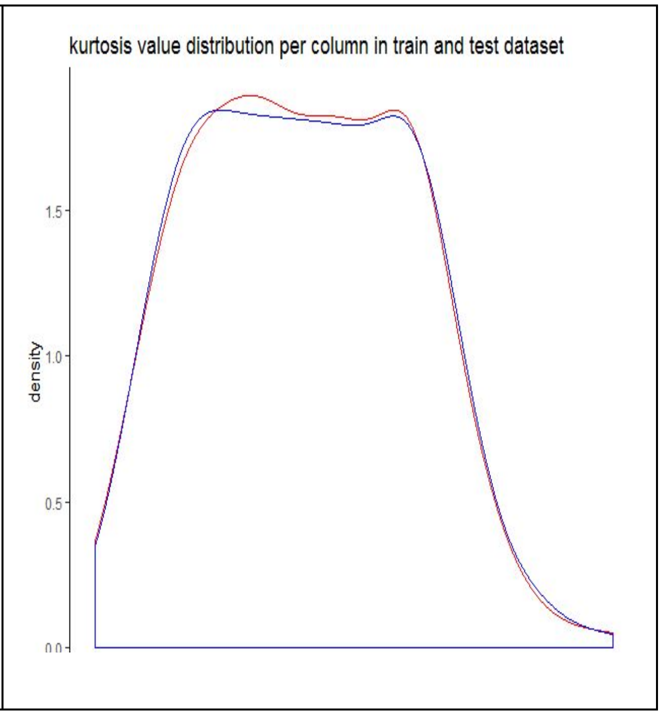
Skew value distribution of train and test attributes in dataset





Kurtosis value distribution of train and test attributes in dataset


kurtosis value distribution per row in train and test dataset
density
kurtosis values per row


kurtosis value distribution per column in train and test dataset
density