**Santander Customer Transaction Prediction**



**By**

**Jerin Joseph**

Table of Contents

[**Chapter 1: Introduction** 3](#_Toc59019163)

[**1.1 Background:** 3](#_Toc59019164)

[**1.2 Problem Statement** 4](#_Toc59019165)

[**1.3 Data Details** 4](#_Toc59019166)

[**1.4 Problem Analysis** 4](#_Toc59019167)

[**1.5 Load Data** 4](#_Toc59019168)

[**Chapter 2: Methodology** 5](#_Toc59019169)

[**2.1 Exploratory Data Analysis (EDA)** 5](#_Toc59019170)

[**2.1.1 Target Classes Distribution** 6](#_Toc59019171)

[**2.1.2 Missing Value Analysis** 6](#_Toc59019172)

[**2.1.3 Basic Features** 9](#_Toc59019173)

[**2.1.4 Data Visualizations** 10](#_Toc59019174)

[**2.1.5 Outlier Analysis** 20](#_Toc59019175)

[**2.1.6 Correlation Analysis** 29](#_Toc59019176)

[**2.1.7 Principal component analysis (PCA)** 30](#_Toc59019177)

[**2.2 Modelling** 31](#_Toc59019178)

[**2.2.1 Evaluation Metric** 31](#_Toc59019179)

[**2.2.2 Model Selection** 34](#_Toc59019180)

[**2.2.3 Logistic Regression** 35](#_Toc59019181)

[**2.2.4 Decision Tree** 42](#_Toc59019182)

[**2.2.5 Light GBM** 49](#_Toc59019183)

[**Chapter 3: Conclusion** 55](#_Toc59019184)

[**3.1 Model Selection** 55](#_Toc59019185)

[**Appendix A – Extra Figures** 56](#_Toc59019186)

[**Appendix B – Complete Python and R Code** 63](#_Toc59019187)

**Chapter 1: Introduction**

## **1.1 Background:**

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

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

products and services might help them achieve their monetary goals.

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

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

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

such as:

* is a customer satisfied?
* Will a customer buy this product?
* Can a customer pay this loan?

## **1.2 Problem Statement**

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

## **1.3 Data Details**

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

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

• ID\_code (string)

• Target (0 or 1)

• 200 numerical variables, named from var\_0 to var\_199

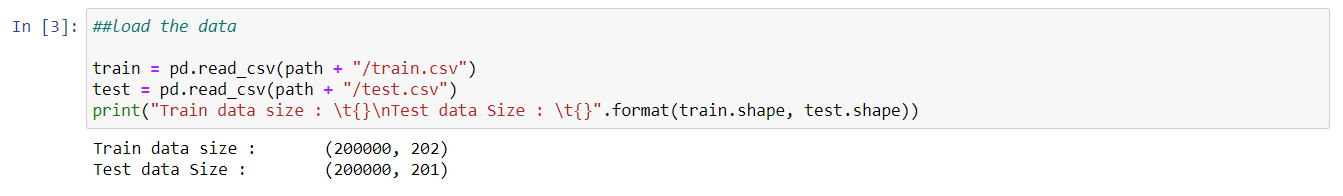
## **1.4 Problem Analysis**

* Supervised: A target variable is included in the training data and the goal is to train a

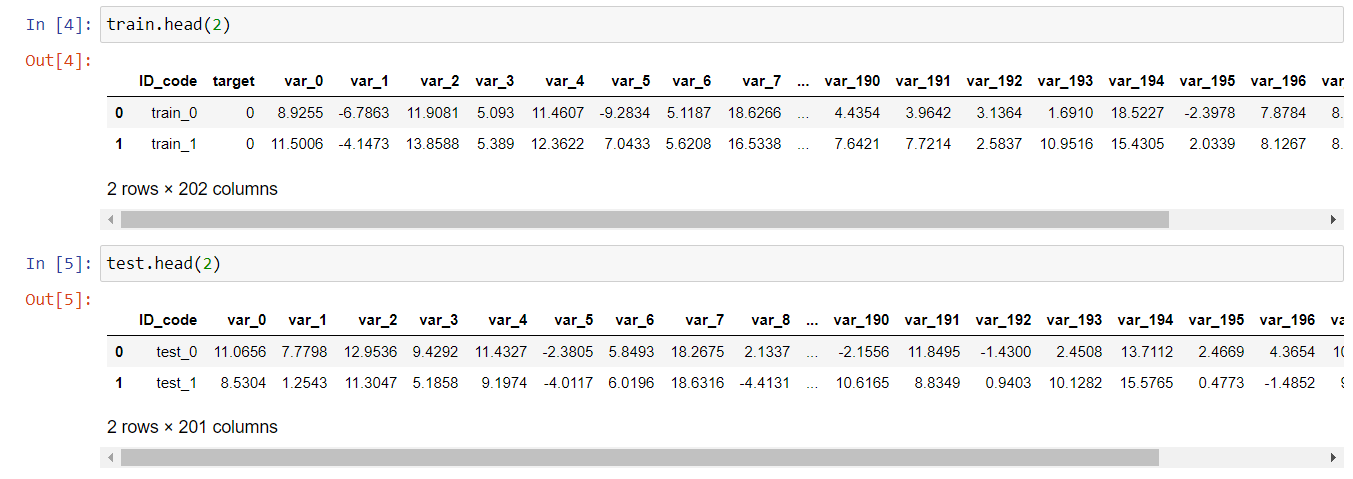
model to learn to predict target values for the test set from the features

* Classification: The target label is a binary variable, 0 (will not make a specific transaction in the future), 1 (will make a specific transaction in the future)

## **1.5 Load Data**



We can see that the train Dataset has 202 columns while the test Dataset has 201 Columns. The extra column in the Train Dataset is the target data set which is not present in the Test Dataset



The data obtained is entirely masked so with even domain knowledge we will not be able to find out any significant features. We can try with basic features like mean, standard deviation, counts, median, etc. We will do feature engineering later.

# **Chapter 2: Methodology**

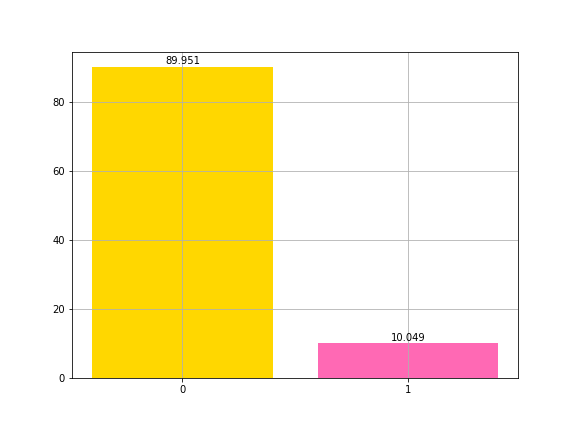
## **2.1 Exploratory Data Analysis (EDA)**

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

Exploratory data analysis is one of the most important steps 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 must clean the data otherwise it will affect performance of the model. Now we are going to explain one by one as follows.

### **2.1.1 Target Classes Distribution**





We can see from the above generated fig that nearly 90% of the Target value is 0(we assume that 0 stands for Customer didn’t make a transaction) and only 10% is 1(we assume 1 stands for Customer made a Transaction).

This makes the data significantly imbalanced!

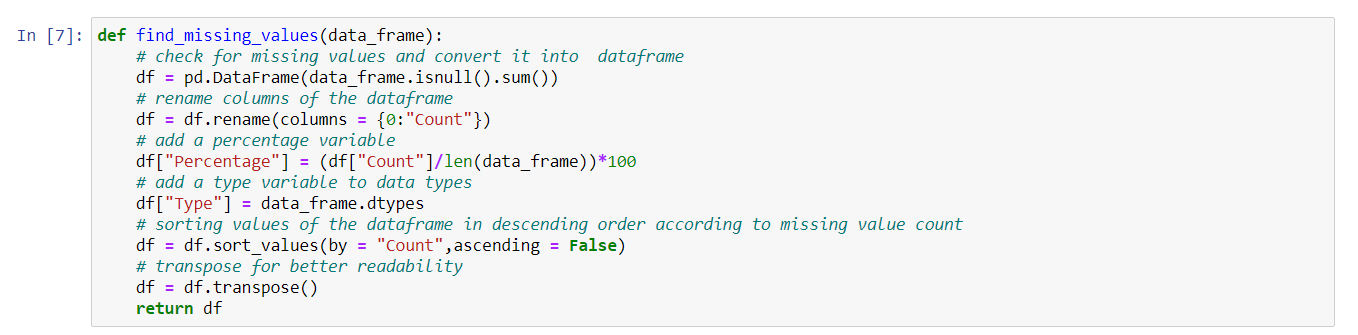
### **2.1.2 Missing Value Analysis**

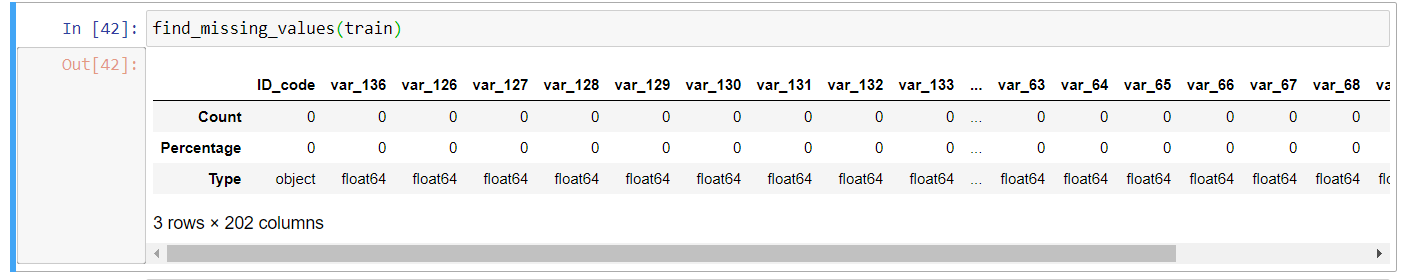
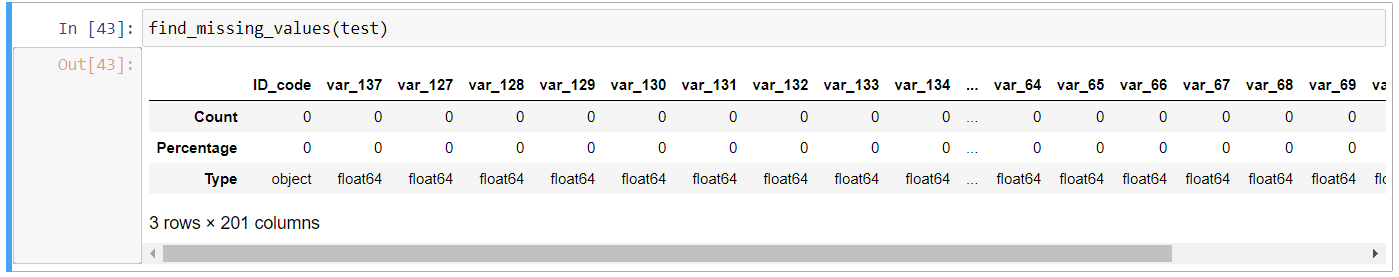
In this, we have to find out if any missing values are present in dataset. If it’s present then we can perform two actions either delete or impute the values. For any variable in our given dataset, theoretically 30% is the maximum percentage of missing values allowed, beyond which we might want to drop the variable from our analysis. The second option is to impute the values using central tendencies like mean, median and mode or we can try out KNN imputation method to impute the missing values.

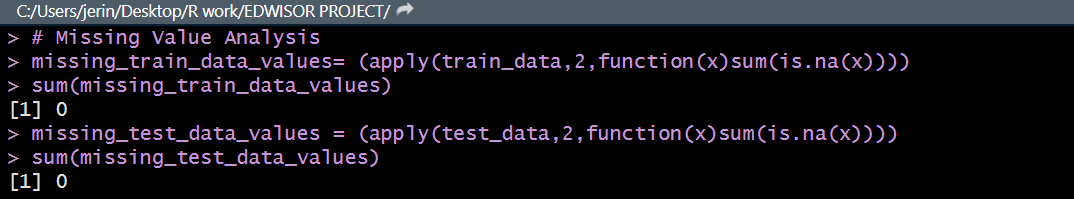
Let’s check if there is any missing data.

Python Code

I defined a function called find\_missing\_values and applied it on the train and test dataset

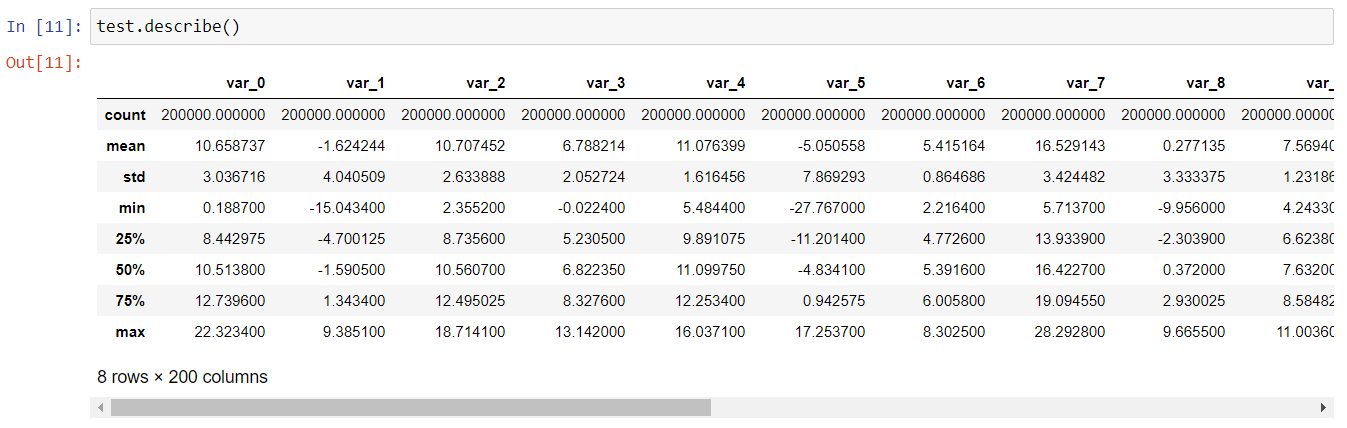


R Code



* We can notice that there are no missing values in both the Train and the Test Dataset.

### **2.1.3 Basic Features**



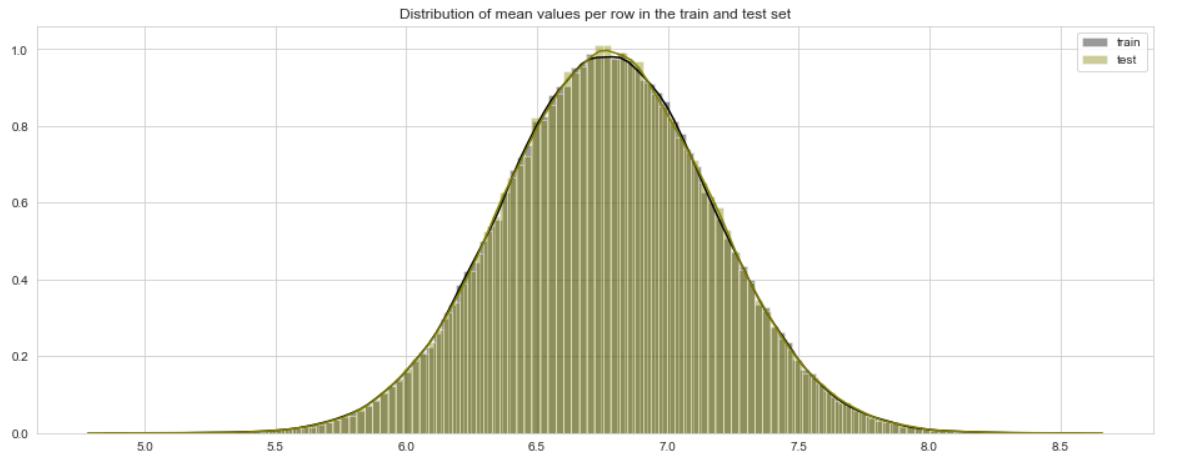
We can propose the following observations:

* standard deviation is relatively large for both train and test variable data
* min, max, mean, std values for train and test data looks quite close
* mean values are distributed over a large range.
* The number of values in train and test set is the same.

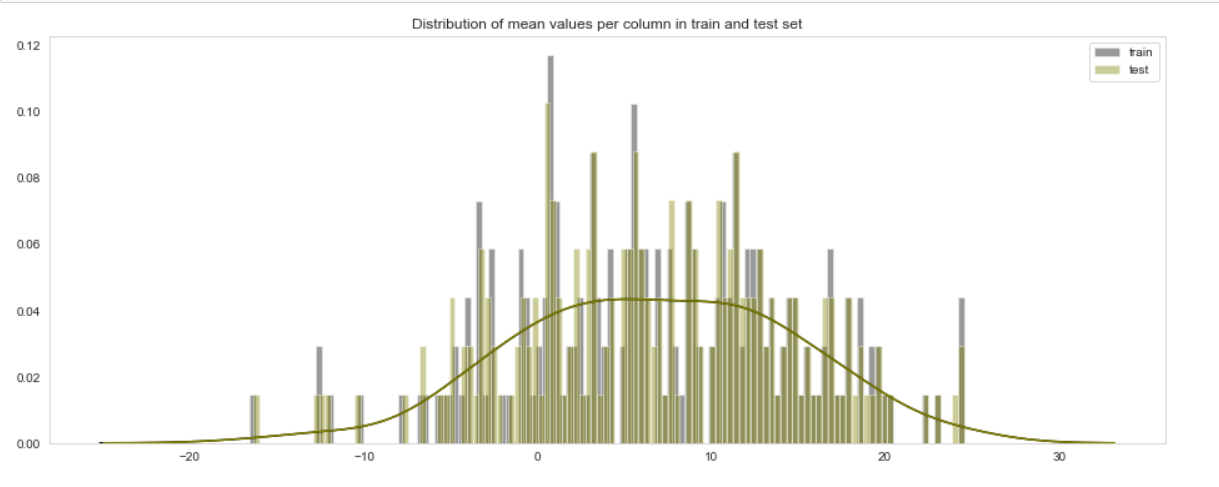
### **2.1.4 Data Visualizations**

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

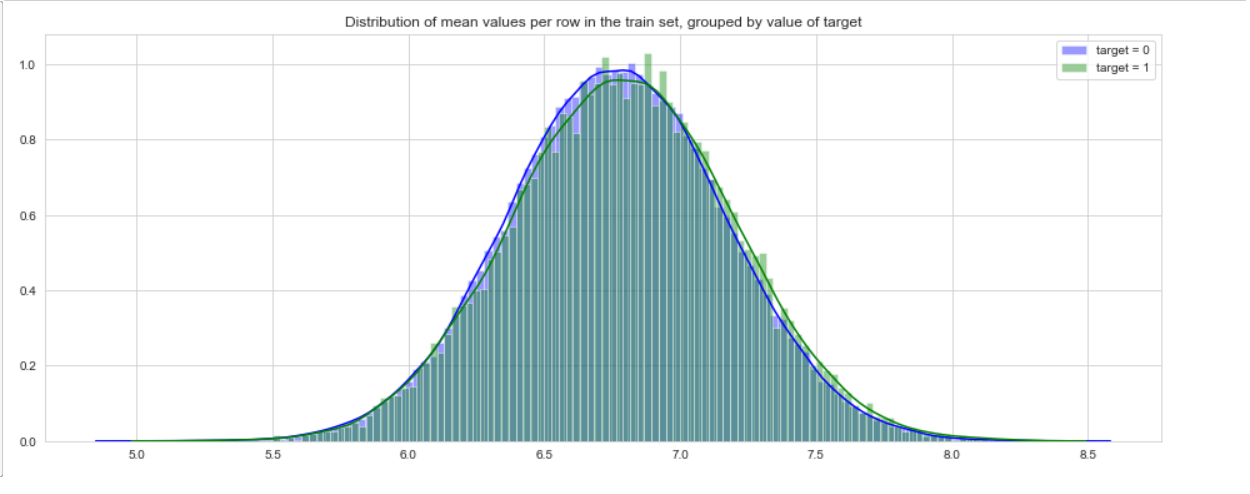
Let us look distribution of mean values per row in train and test dataset



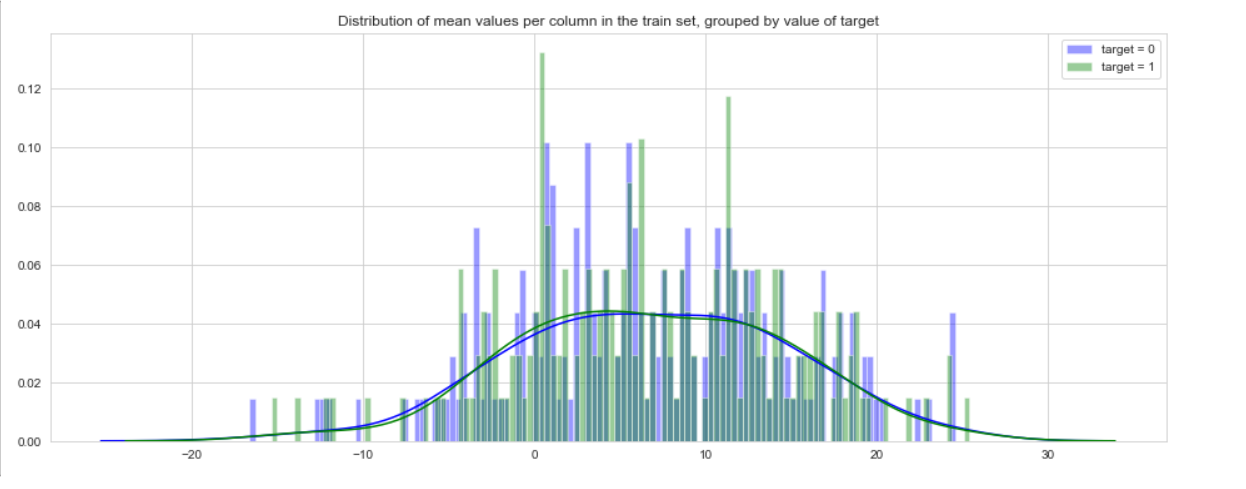
Let us look distribution of mean values per column in train and test dataset



Let us look distribution of mean values per row in train dataset, grouped by value of target



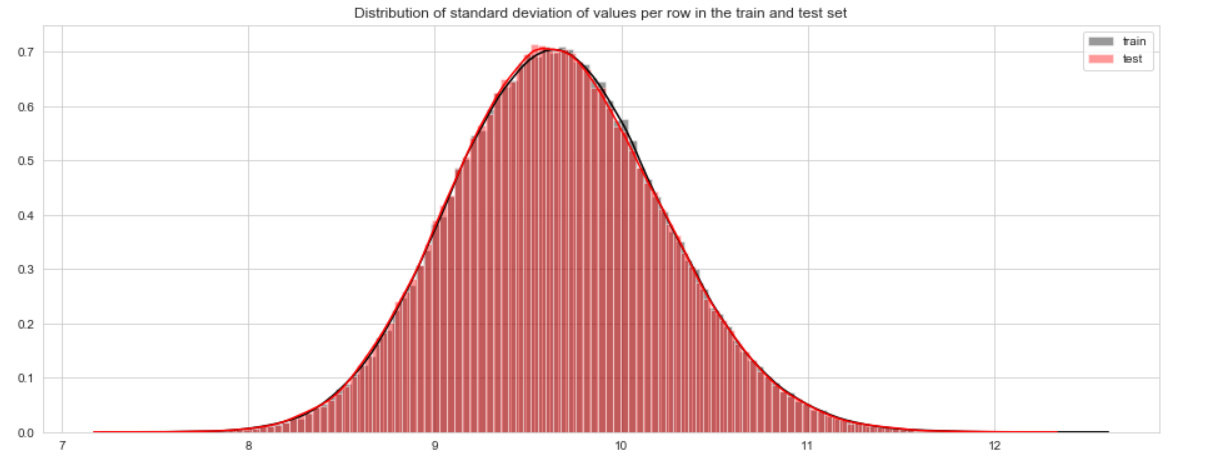
Let us look distribution of mean values per column in train dataset, grouped by value of target



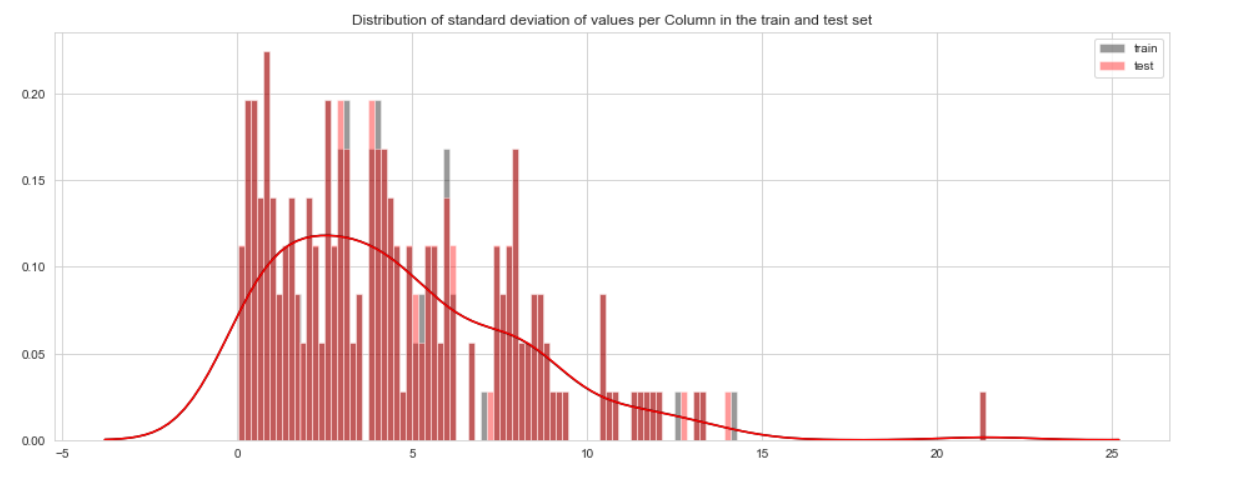
* Mean values of train and test set looks quite close
* Mean values are distributed over a large range.

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

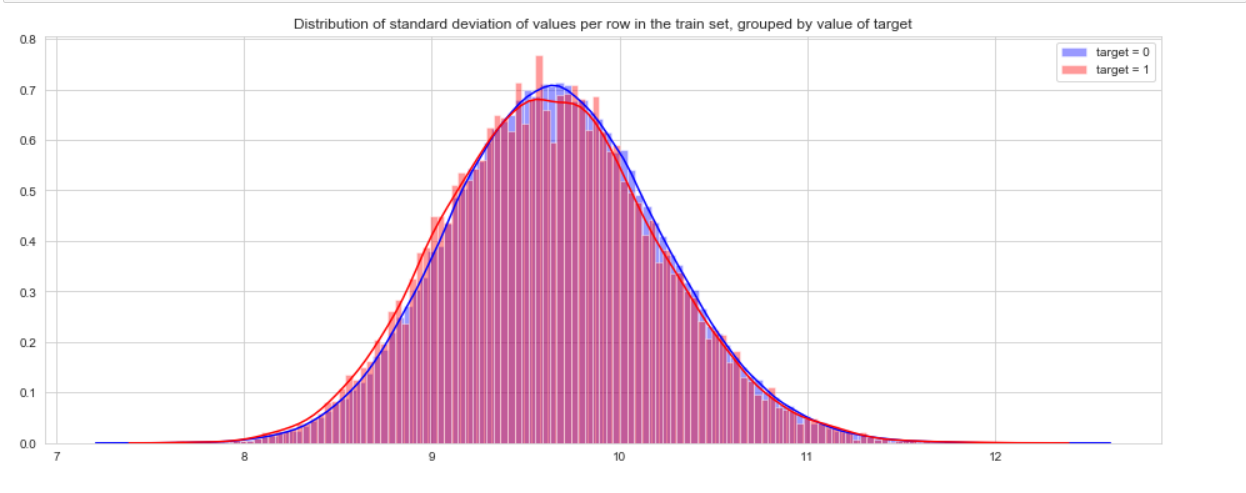
Let us look distribution of standard deviation per row in train and test dataset



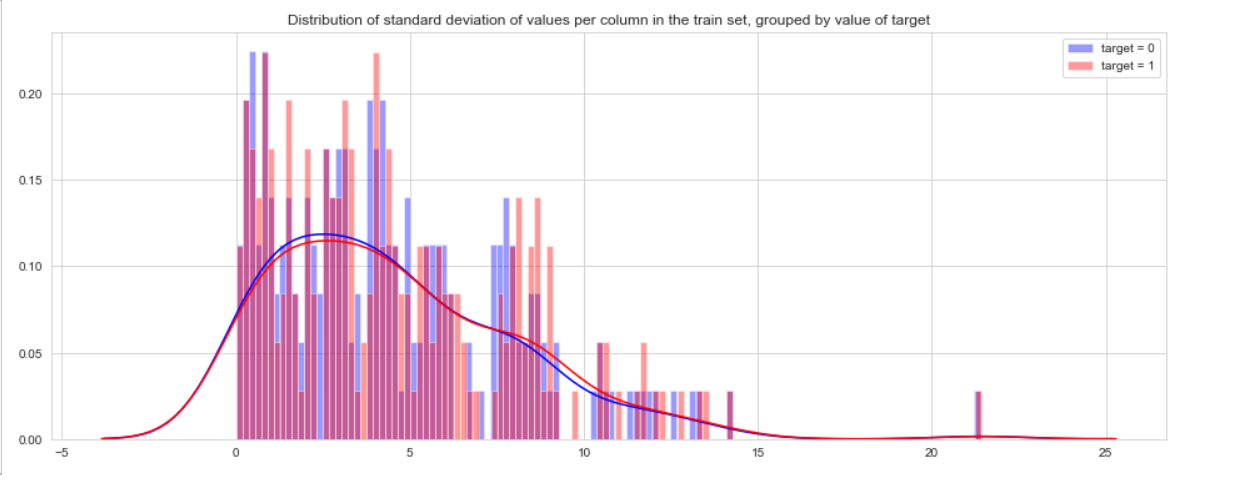
Let us look distribution of standard deviation per column in train and test dataset



Let us look distribution of standard deviation(std) values per row in train dataset, grouped by value of target



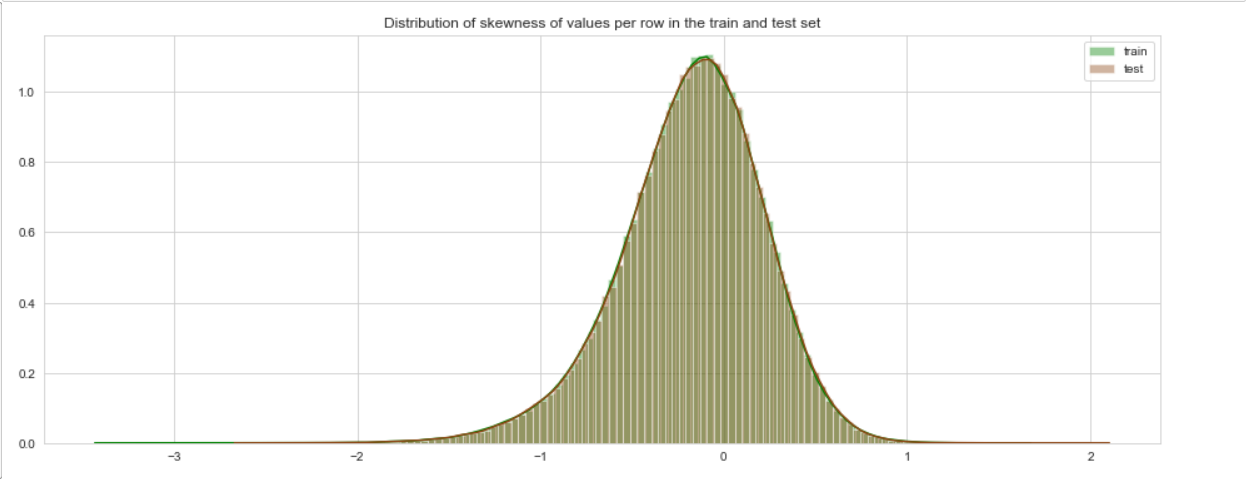
Let us look distribution of standard deviation(std) values per column in train dataset, grouped by value of target



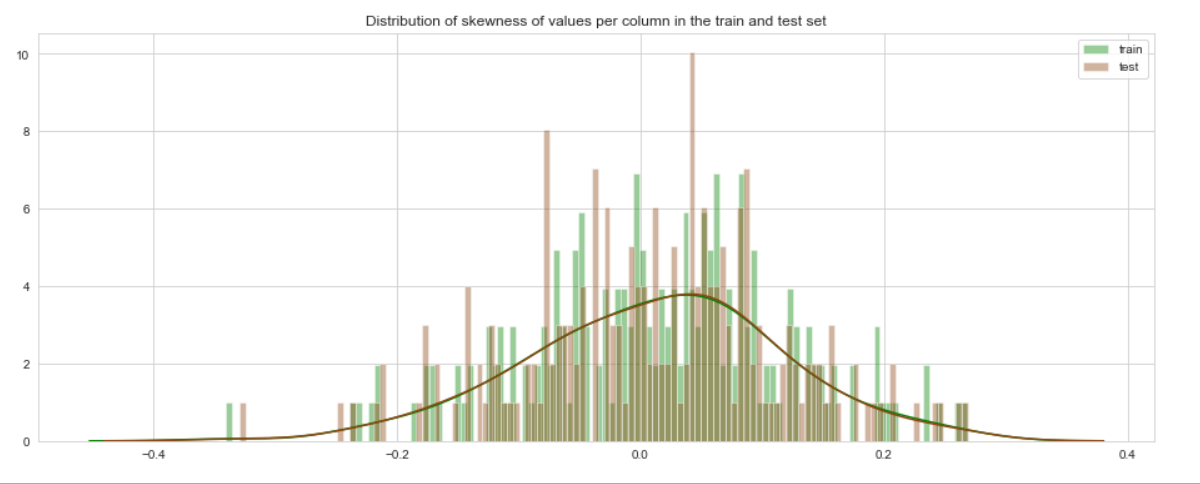
* Standard deviation is relatively large for both train and test variable data.
* Standard deviation values for train and test data looks similar

**Distribution of skewness of values in both train and test dataset: -**

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



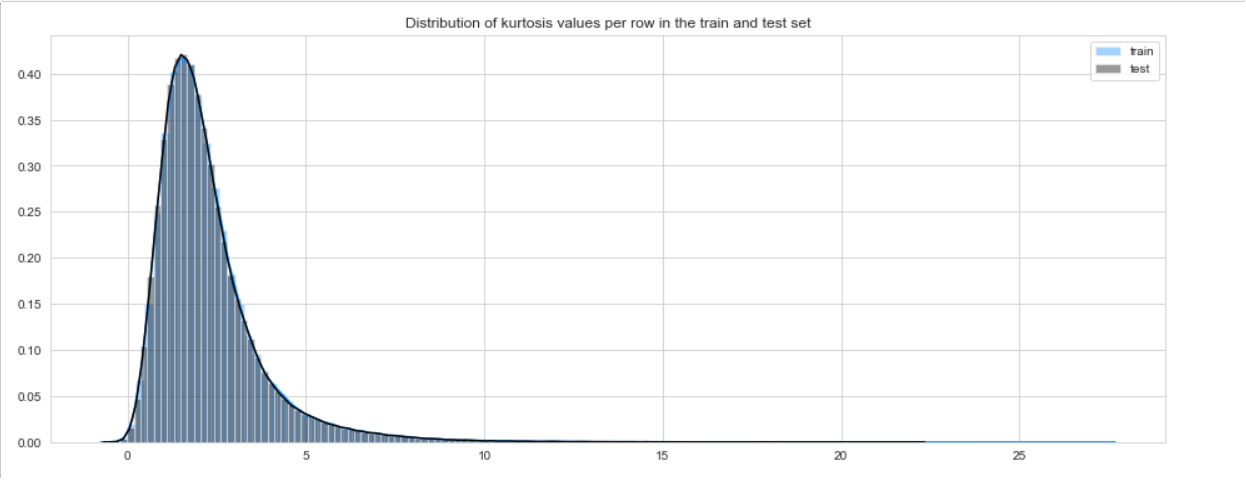
Let us look distribution of skewness of values per column in train and test dataset



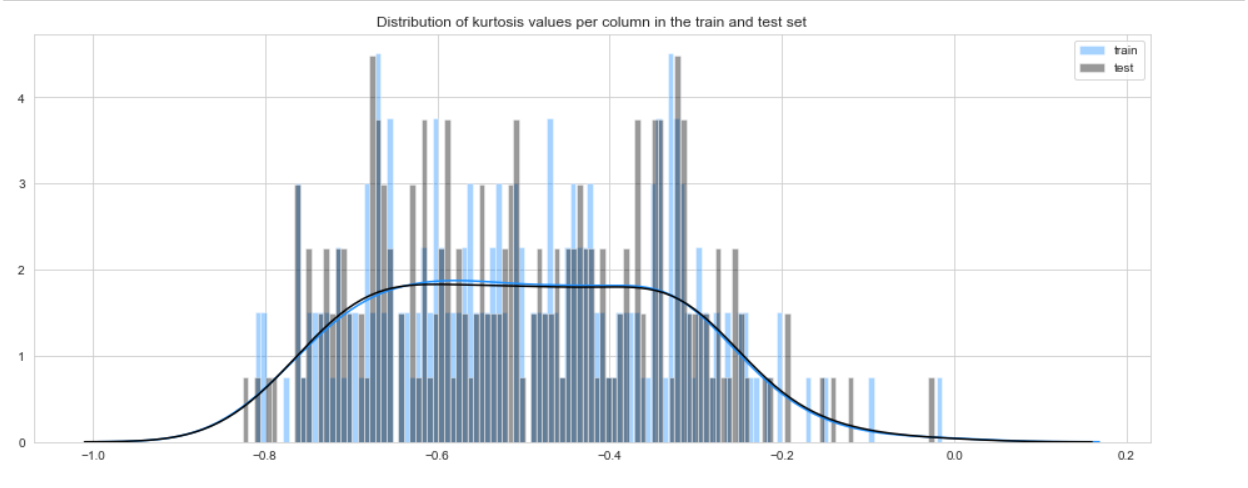
* We found the distribution is left skewed

**Distribution of kurtosis in both train and test dataset: -**

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



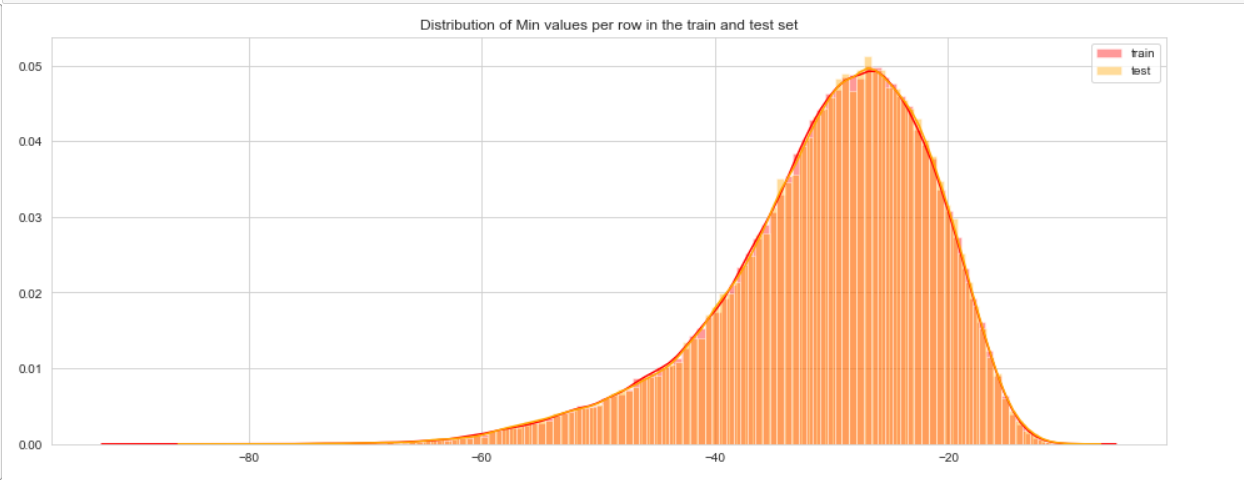
Let us look distribution of kurtosis per column in train and test dataset



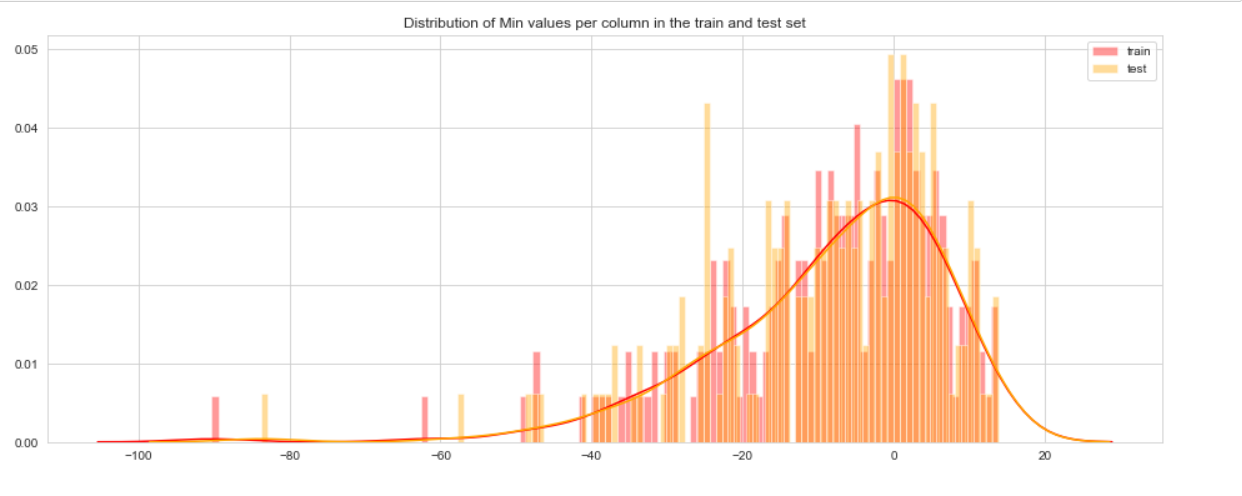
* We found the distribution to be Leptokurtic which means heavy tails on either side indicating large outliers

**Distribution of Min Values in both train and test dataset: -**

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

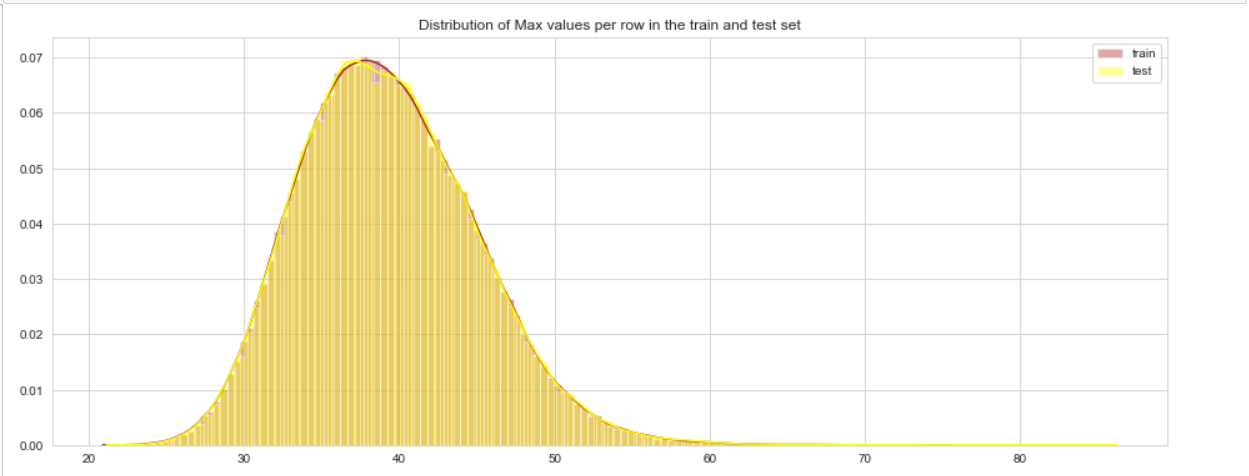


Let us look distribution of Min values per column in train and test dataset

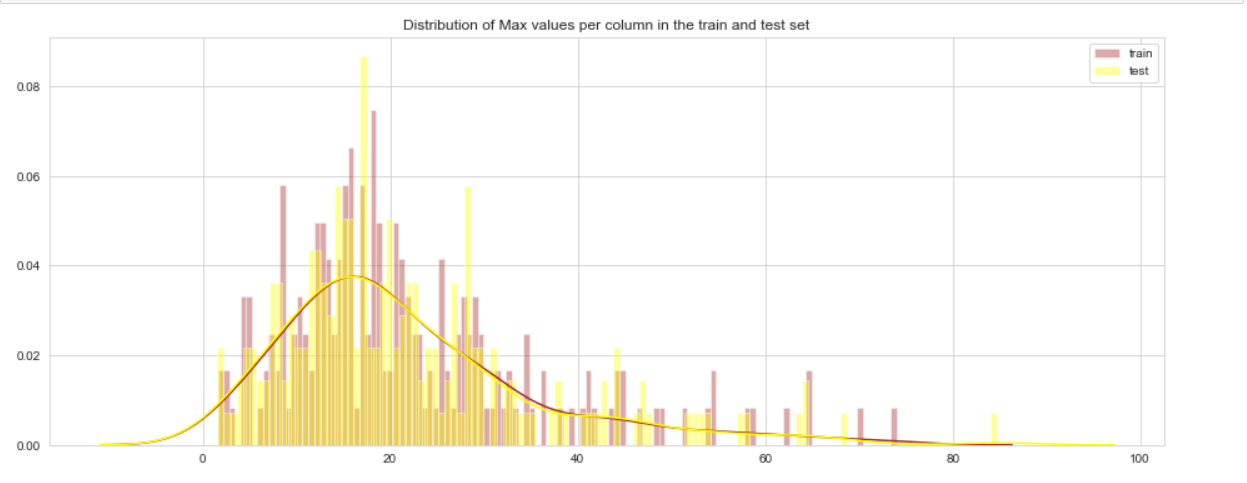


**Distribution of Max Values in both train and test dataset: -**

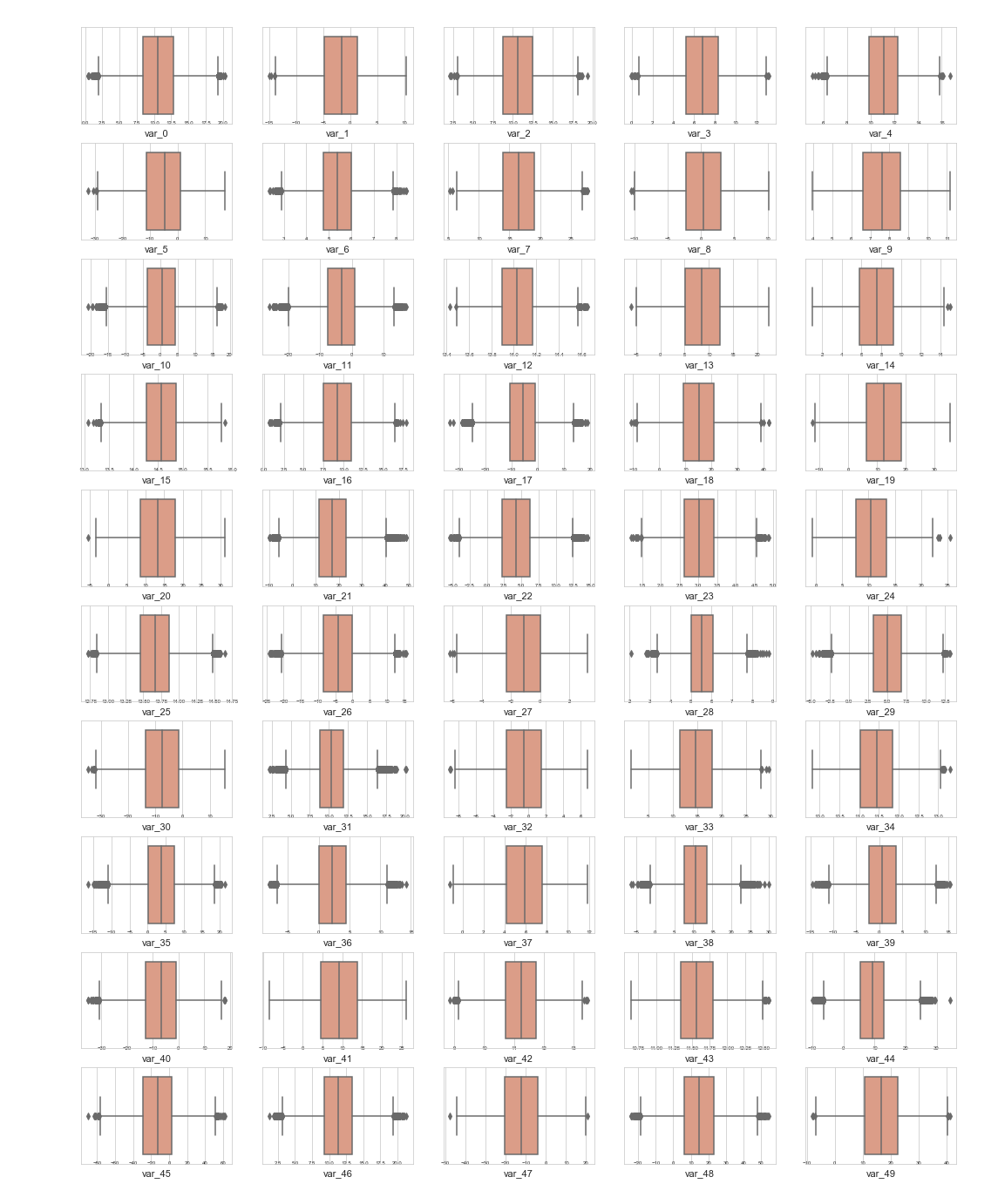
Let us look distribution of Max values per row in train and test dataset



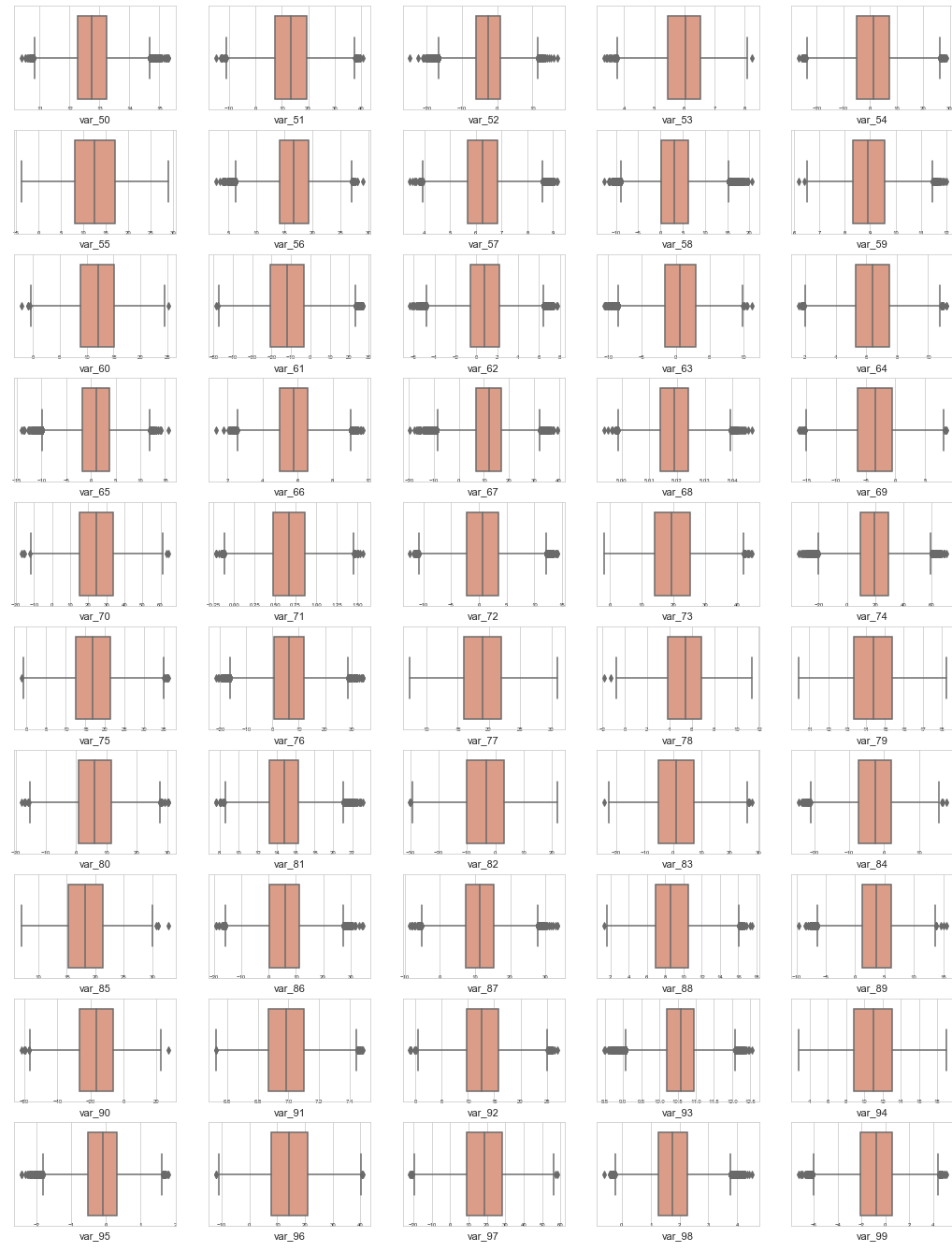
Let us look distribution of Max values per column in train and test dataset



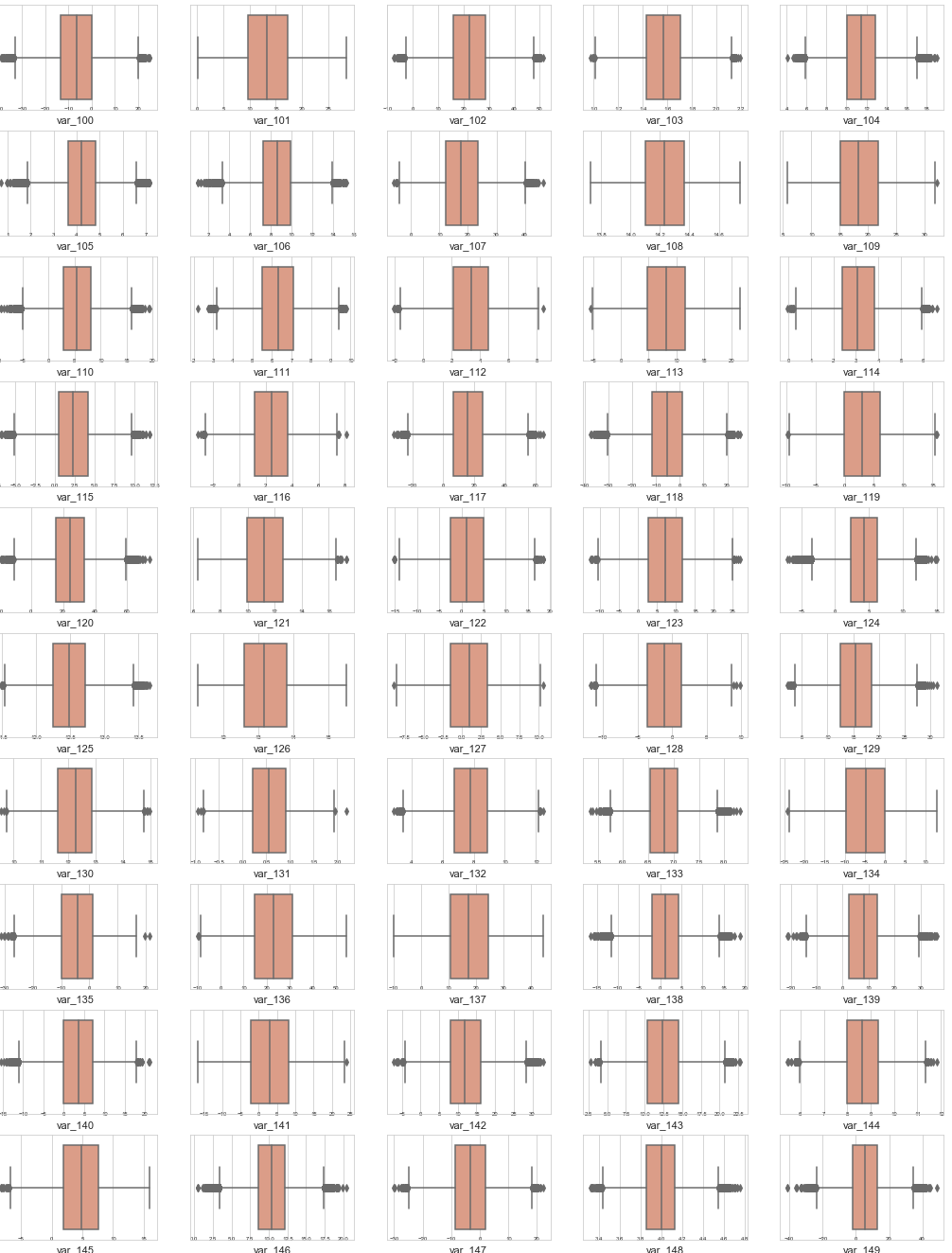
* Min and Max values for train and test dataset looks almost identical
* We can see from above that all the variables have nearly same distribution with the same scales

**2.1.5 Outlier Analysis**

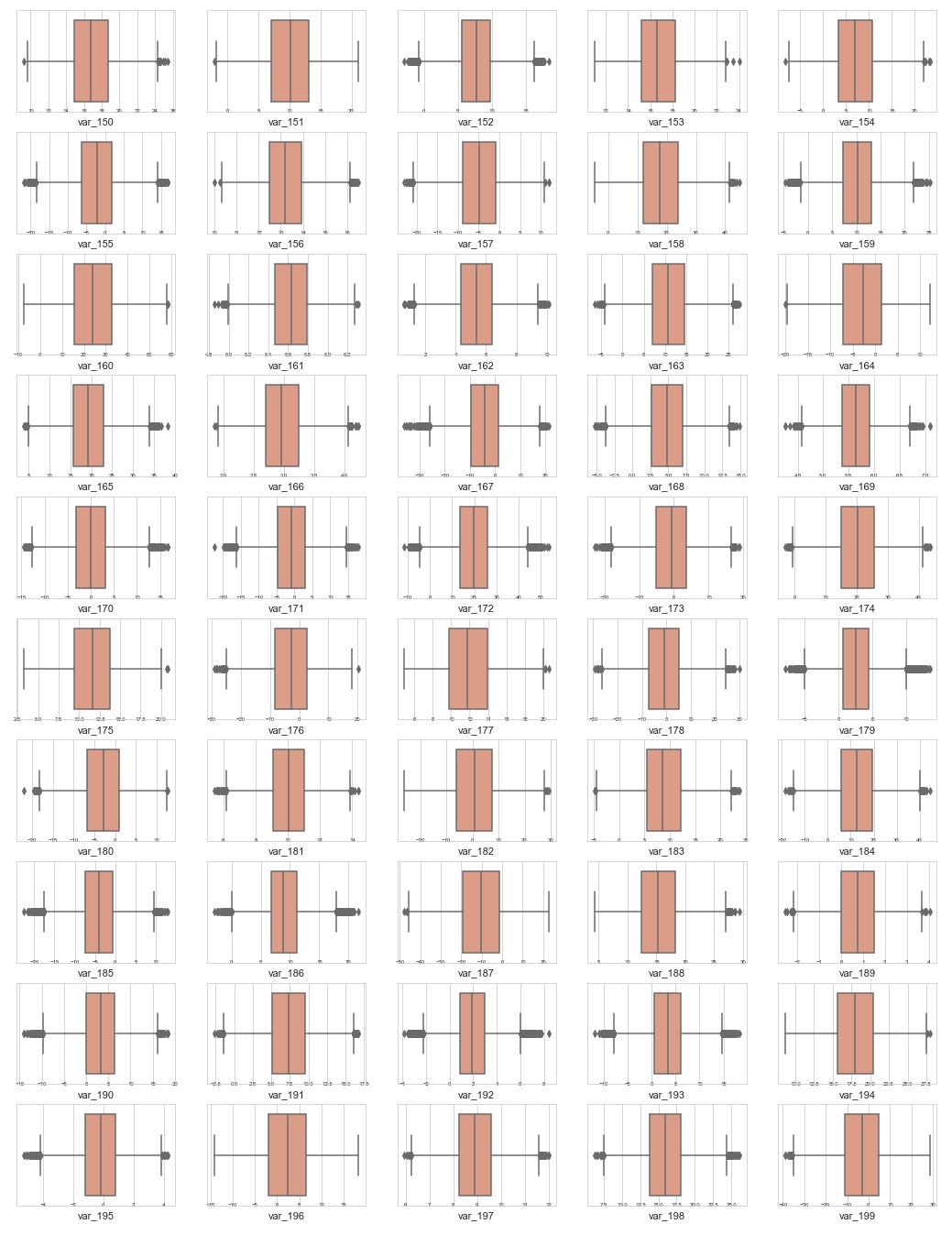
**Fig: Box plots from var\_0 to var\_49**



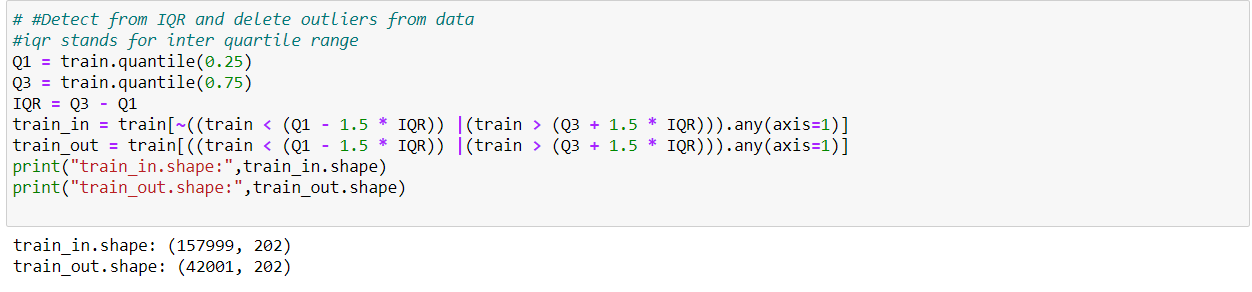
**Fig: Box plots from var\_50 to var\_99**

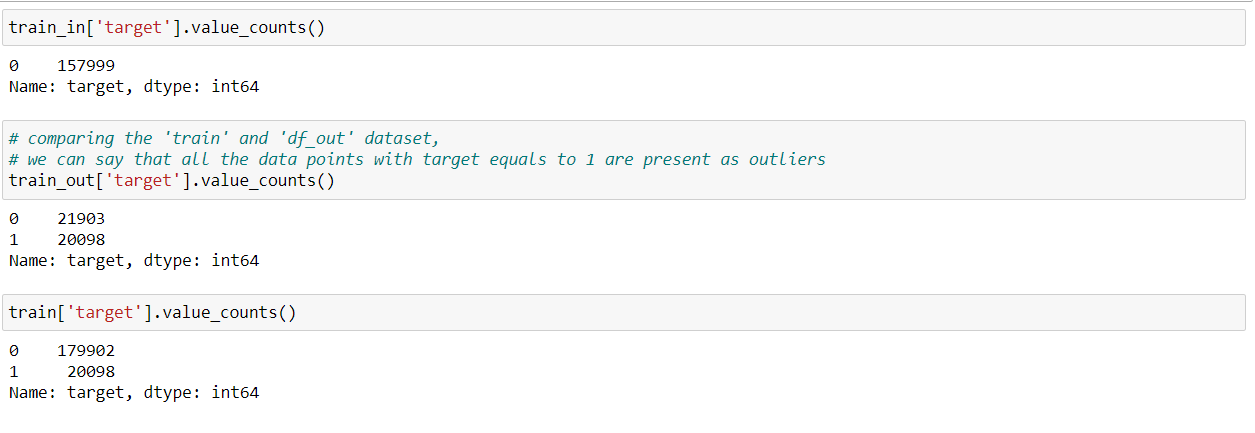
****

**Fig: Box plots from var\_100 to var\_149**

****

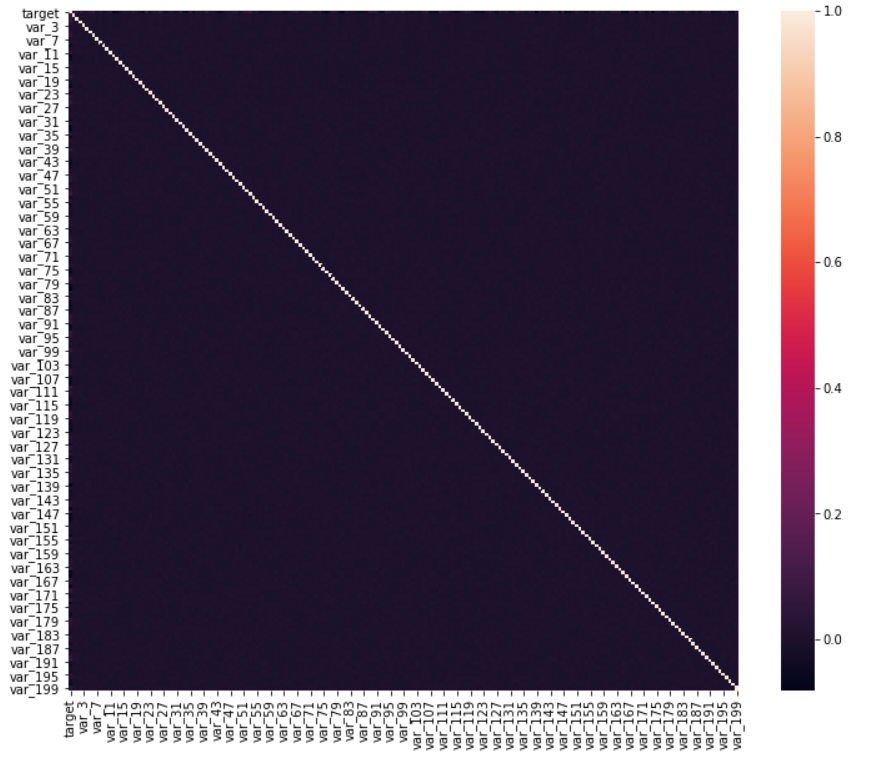
**Fig: Box plots from var\_150 to var\_199**





* Almost all variables have outliers present from the box plots above.
* After separating outliers and inliers with IQR method we found that all the target variables with label as one are outliers.
* Outliers present in our data, are meaningful and thus can’t be removed.

### **2.1.6 Correlation Analysis**



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

From correlation 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 and are not correlated to each other.

### **2.1.7 Principal component analysis (PCA)**

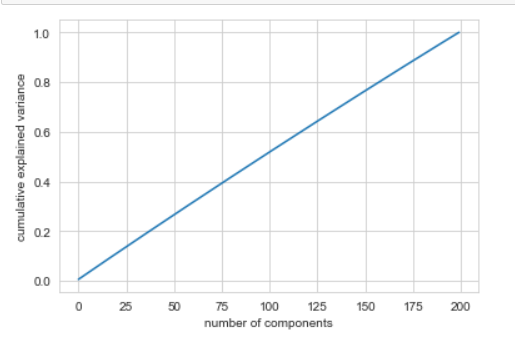
PCA is a dimensionality reduction technique that reduces less-informative 'noise' features.

But PCA is sensitive to variance and different scales, so standardizing will help PCA

perform better.

However, since we found that the correlation between different features in the training

dataset is not that significant, so using PCA might not be meaningful.



* the line of cumulative sums of explained variance ratio when you PCA the data set, it is indicative of a dataset that has already undergone PCA (get straight line i.e. y=x).
* Since PCA hasn't been useful, we decided to proceed with the existing dataset with all 200 variables.

## **2.2 Modelling**

### **2.2.1 Evaluation Metric**

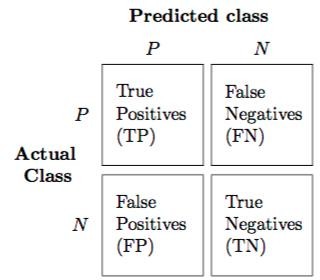
Now, we will be using three models for predicting the target variable, but we need to decide which model better for this project. There are many metrics used for model evaluation. Classification accuracy may be misleading if we have an imbalanced dataset or if we have more than two classes in dataset.

For classification problems, the confusion matrix used for evaluation. But, in our case the data is imbalanced.

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

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

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



Accuracy: - The ratio of correct predictions to total predictions

Accuracy = TP+TN / (Total Predictions)

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

Error rate = FN + FP/ (Total Predictions)

Accuracy=1-Error rate

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

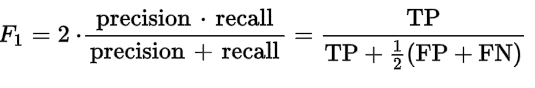
Precision = TP/(TP+FP)

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

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

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

F-measure**- ​**It is difficult to compare two models with low precision and high recall orvice 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.



**ROC\_AUC Scores:**

It can be more flexible to predict probabilities of an observation belonging to each class in a classification problem rather than predicting classes directly.

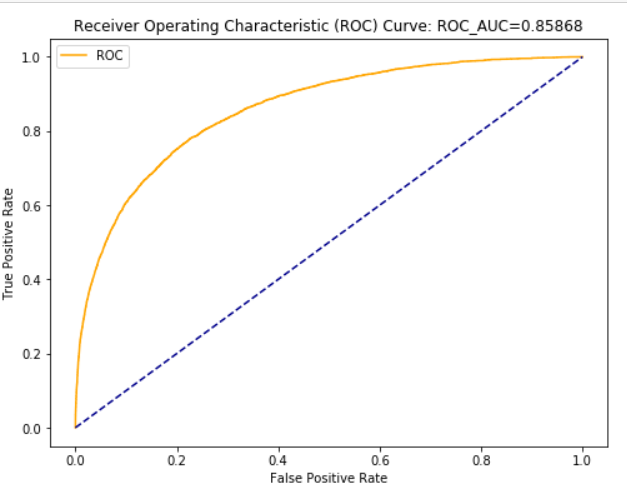
The reason for this is to provide our model the capability to choose and even calibrate the threshold for how to interpret the predicted probabilities.

There are two diagnostic tools that help in the interpretation of probabilistic forecast for binary (two-class) classification predictive modelling problems are **ROC Curves** and **Precision-Recall curves**.

**ROC** is a probability curve for different classes. ROC tells us how good the model is for distinguishing the given classes, in terms of the predicted probability.

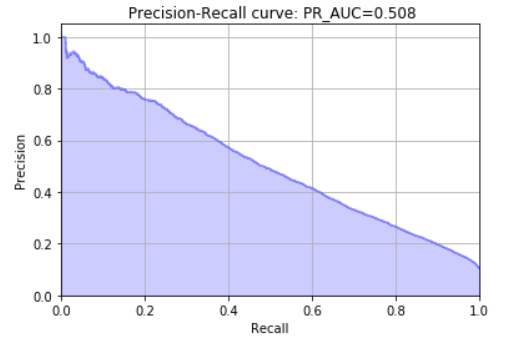
A typical ROC curve has False Positive Rate (FPR) on the X-axis and True Positive Rate (TPR) on the Y-axis.

The area covered by the curve is the area between the orange line (ROC) and the axis. This area covered is **AUC**. The bigger the area covered, the better the machine learning models is at distinguishing the given classes. Ideal value for AUC is 1.



**Precision** is a ratio of the number of true positives divided by the sum of the true positives and false positives. It describes how good a model is at predicting the positive class.

**Recall** is calculated as the ratio of the number of true positives divided by the sum of the true positives and the false negatives. Recall is the same as sensitivity.



**Precision-Recall** curves are useful in cases where there is an imbalance in the observations between the two classes. Specifically, there are many examples of no event (class 0) and only a few examples of an event (class 1).

Key to the calculation of precision and recall is that the calculations do not make use of the true negatives. It is only concerned with the correct prediction of the minority class, class 1.

A precision-recall curve is a plot of the precision (y-axis) and the recall (x-axis) for different thresholds, much like the ROC curve.

### **2.2.2 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

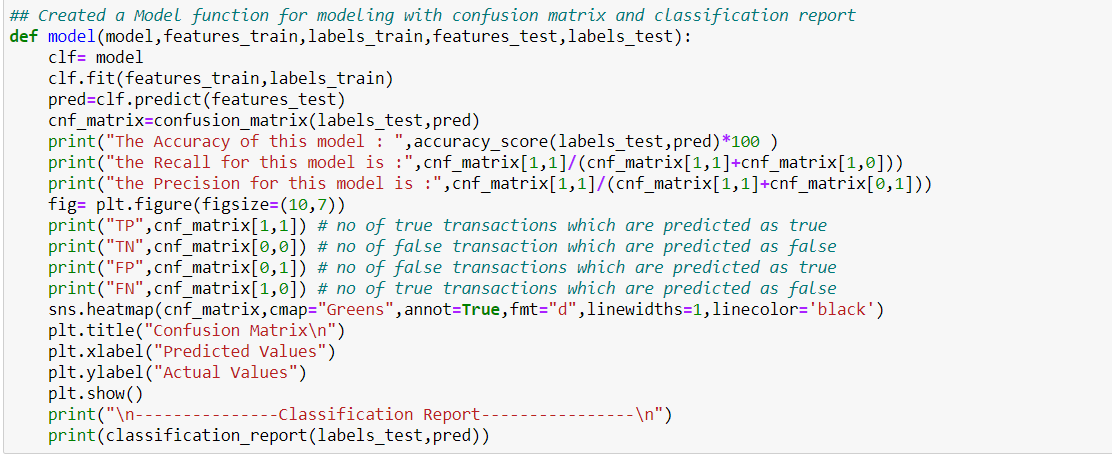
2. Ordinal

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.

For ease of process, we have created a model function which will carry out our modeling with confusion matrix and also showing classification report



### **2.2.3 Logistic Regression**

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

We will start our model building from the simplest to more complex. Therefore, we use

Simple Logistic Regression first as our base model.

Since this is an unbalanced dataset, we need to define parameter 'class\_weight = balanced' which will give equal weights to both the targets irrespective of their representation in the training dataset.

**Python Code**

|  |
| --- |
| # Splitting the train and test data  Target = train['target']  # Input dataset for Train and Test  train\_inp = train.drop(columns = ['target', 'ID\_code'])  test\_inp = test.drop(columns = ['ID\_code'])  X\_train, X\_test, y\_train, y\_test = train\_test\_split(train\_inp, Target, test\_size=0.2, random\_state = 42)  print ("X\_train: ", X\_train.shape)  print ("y\_train: ", y\_train.shape)  print("X\_test: ", X\_test.shape)  print ("y\_test: ", y\_test.shape) |

Output-

X\_train: (160000, 200)

y\_train: (160000,)

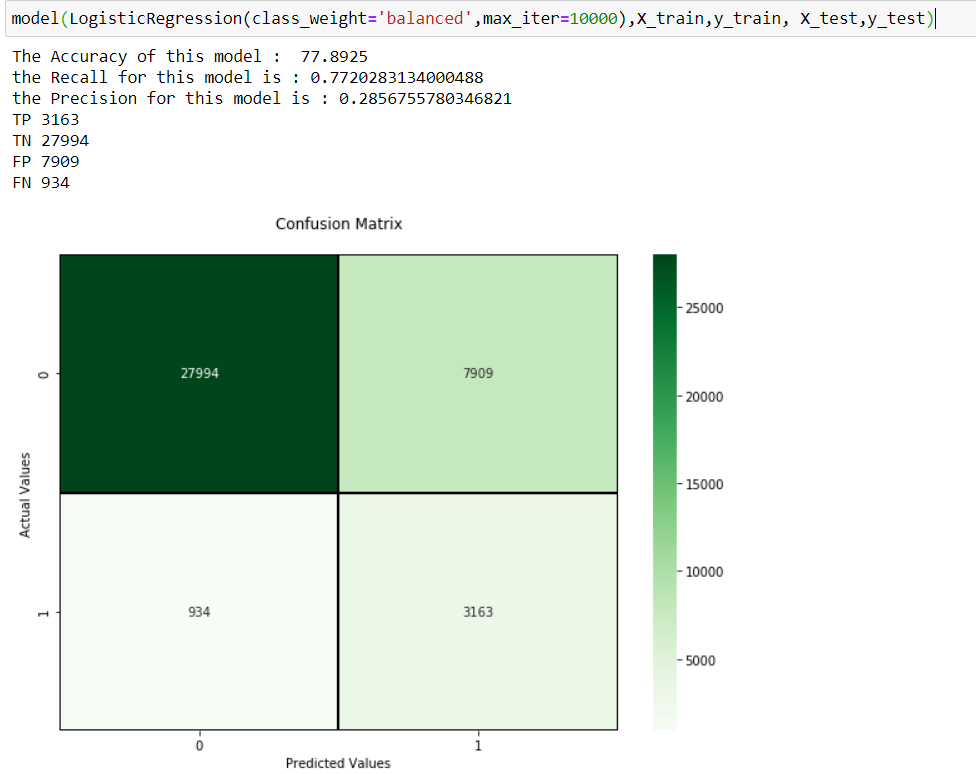
X\_test: (40000, 200)

y\_test: (40000,)

|  |
| --- |
| ## Created a Model function for modeling with confusion matrix and classification report  def model(model,features\_train,labels\_train,features\_test,labels\_test):  clf= model  clf.fit(features\_train,labels\_train)  pred=clf.predict(features\_test)  cnf\_matrix=confusion\_matrix(labels\_test,pred)  print("The Accuracy of this model : ",accuracy\_score(labels\_test,pred)\*100 )  print("the Recall for this model is :",cnf\_matrix[1,1]/(cnf\_matrix[1,1]+cnf\_matrix[1,0]))  print("the Precision for this model is :",cnf\_matrix[1,1]/(cnf\_matrix[1,1]+cnf\_matrix[0,1]))  fig= plt.figure(figsize=(10,7))  print("TP",cnf\_matrix[1,1]) # no of true transactions which are predicted as true  print("TN",cnf\_matrix[0,0]) # no of false transaction which are predicted as false  print("FP",cnf\_matrix[0,1]) # no of false transactions which are predicted as true  print("FN",cnf\_matrix[1,0]) # no of true transactions which are predicted as false  sns.heatmap(cnf\_matrix,cmap="Greens",annot=True,fmt="d",linewidths=1,linecolor='black')  plt.title("Confusion Matrix\n")  plt.xlabel("Predicted Values")  plt.ylabel("Actual Values")  plt.show()  print("\n---------------Classification Report----------------\n")  print(classification\_report(labels\_test,pred)) |

|  |
| --- |
| model(LogisticRegression(class\_weight='balanced',max\_iter=10000),X\_train,y\_train, X\_test,y\_test) |

Output-



---------------Classification Report----------------

precision recall f1-score support

0 0.97 0.78 0.86 35903

1 0.29 0.77 0.42 4097

accuracy 0.78 40000

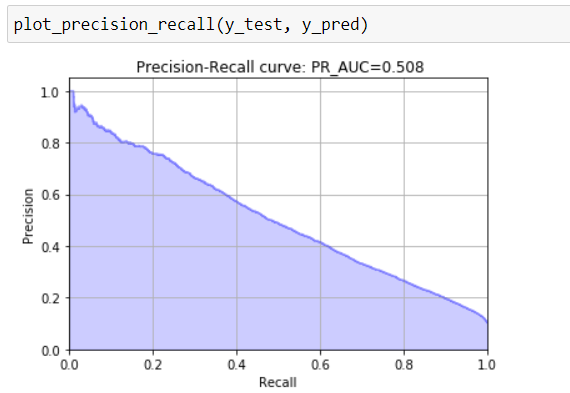
macro avg 0.63 0.78 0.64 40000

weighted avg 0.90 0.78 0.82 40000

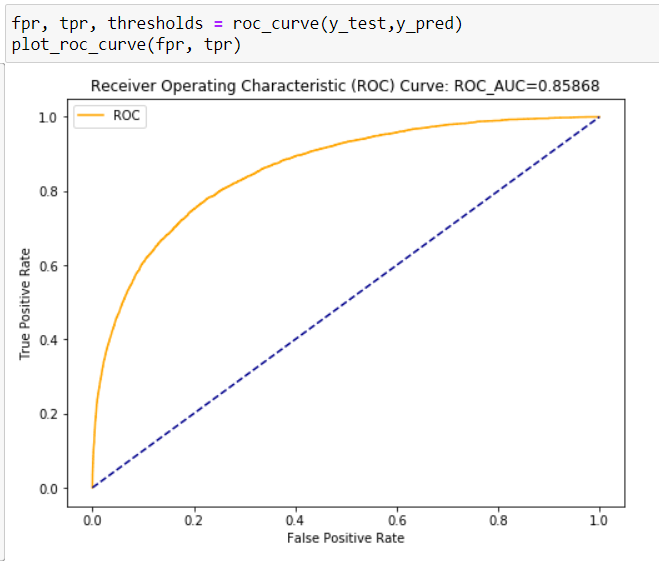
Plotting and Checking PR-AUC Score and ROC-AUC Score

|  |
| --- |
| logreg\_scaled = LogisticRegression(class\_weight='balanced',max\_iter=10000).fit(X\_train,y\_train)  y\_pred = logreg\_scaled.predict\_proba(X\_test)[:,1] |

|  |
| --- |
| # Function to plot precision-recall curve  #Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced  def plot\_precision\_recall(y\_test, y\_pred):  precision, recall, threshold = precision\_recall\_curve(y\_test, y\_pred)  plt.step(recall, precision, color='b', alpha=0.3,where='post')  plt.fill\_between(recall, precision, alpha=0.2, color='b')  plt.xlabel('Recall')  plt.ylabel('Precision')  # set the y range  plt.ylim([0.0, 1.05])  # set the x raneg  plt.xlim([0.0, 1.0])  plt.title(' Precision-Recall curve: PR\_AUC={0:0.3f}'.format( auc(recall, precision)))  plt.grid() |



|  |
| --- |
| # Function to plot ROC curve  def plot\_roc\_curve(fpr, tpr):  fig= plt.figure(figsize=(8,6))  plt.plot(fpr, tpr, color='orange', label='ROC')  plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('Receiver Operating Characteristic (ROC) Curve: ROC\_AUC={0:0.5f}'.format(roc\_auc\_score(y\_test, y\_pred)))  plt.legend()  plt.show() |

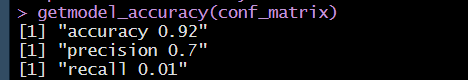


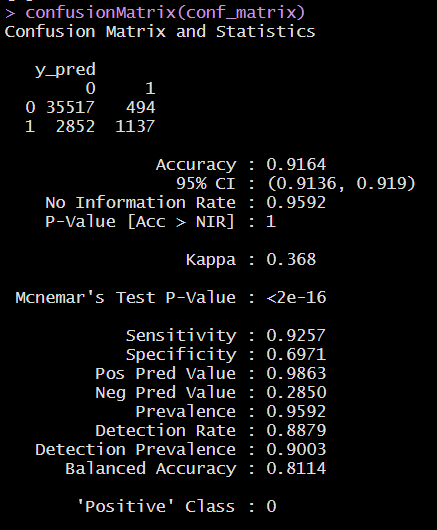
**R Code**

|  |
| --- |
| ##split the data into train and test  set.seed(1234)  require("caret")  train\_data.index = createDataPartition(train\_data$target, p = .80, list = FALSE)  train = train\_data[ train\_data.index,]  test = train\_data[-train\_data.index,]  #Function to get model accuracy  getmodel\_accuracy=function(conf\_matrix)  {  model\_parm =list()  tn =conf\_matrix[1,1]  tp =conf\_matrix[2,2]  fp =conf\_matrix[1,2]  fn =conf\_matrix[2,1]  p =(tp)/(tp+fp)  r =(fp)/(fp+tn)  f1=2\*((p\*r)/(p+r))  print(paste("accuracy",round((tp+tn)/(tp+tn+fp+fn),2)))  print(paste("precision",round(p ,2)))  } |

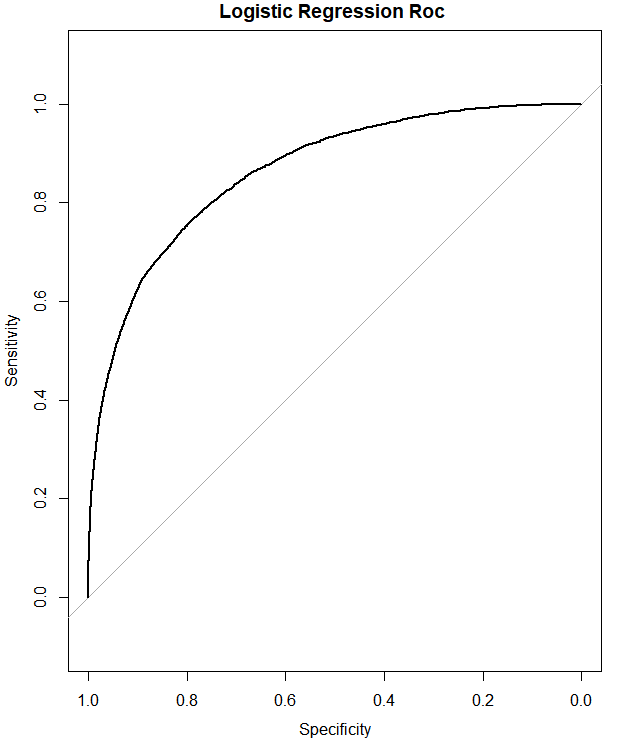
|  |
| --- |
| ##################### Logistic Regression Model ########################  logit\_model =glm(target~. ,data =train ,family='binomial')  # model summary  summary(logit\_model)  #get model predicted probality  y\_prob =predict(logit\_model , test[,-1] ,type = 'response' )  # convert probality to class according to thresshold  y\_pred = ifelse(y\_prob >0.5, 1, 0)  #create confusion matrix  conf\_matrix= table(test[,1] , y\_pred)  #print model accuracy  getmodel\_accuracy(conf\_matrix) |

Output-





|  |
| --- |
| # get auc  roc=roc(test[,1], y\_prob)  print(roc )  # plot roc \_auc plot  plot(roc ,main ="Logistic Regression Roc ") |
|  |



Area under the curve: 0.8627

### **2.2.4 Decision Tree**

Moving on to a slightly advanced algorithm, decision trees. Again, the parameters here are class\_weight to deal with unbalanced target variable, random\_state for reproducibility of same trees.

The feature max\_features and min\_sample\_leaf are used to prune the tree and avoid

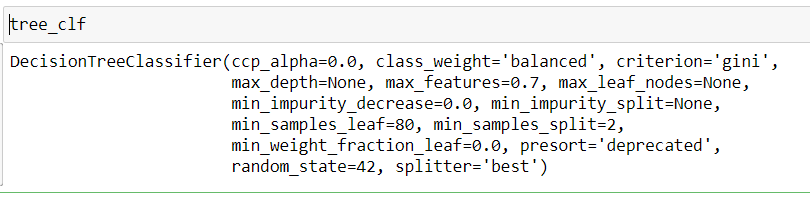
overfitting to the training data.

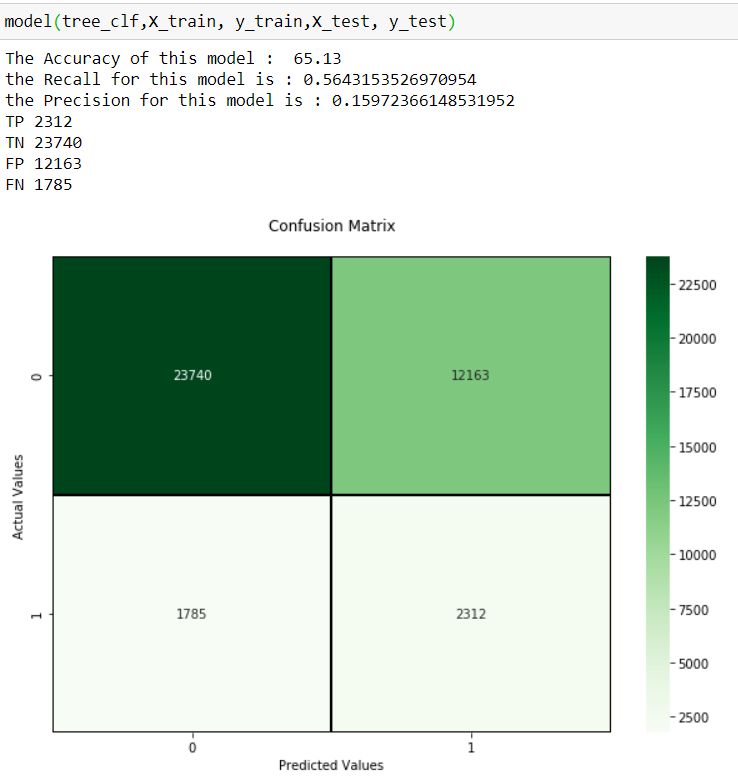
Max\_features defines what proportion of available input features will be used to create tree. Min\_sample\_leaf restricts the minimum number of samples in a leaf node, making sure none of the leaf nodes has less than 80 samples in it. By default it takes value "1". If leaf nodes have less samples it implies, we have grown the tree too much and trying to predict each sample

very precisely, thus leading to overfitting.

**Python Code**

|  |
| --- |
| tree\_clf = DecisionTreeClassifier(class\_weight='balanced', random\_state = 42,  max\_features = 0.7, min\_samples\_leaf = 80) |





---------------Classification Report----------------

precision recall f1-score support

0 0.93 0.66 0.77 35903

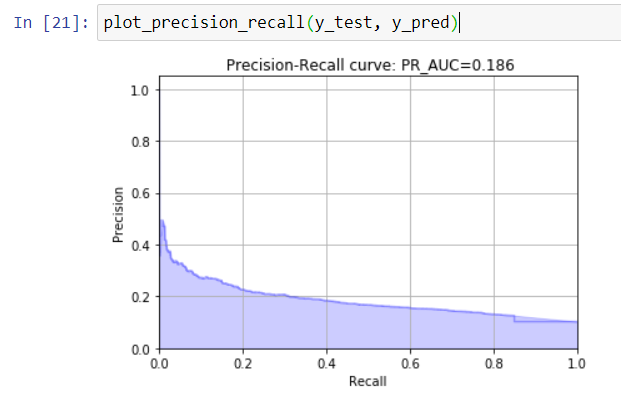
1 0.16 0.56 0.25 4097

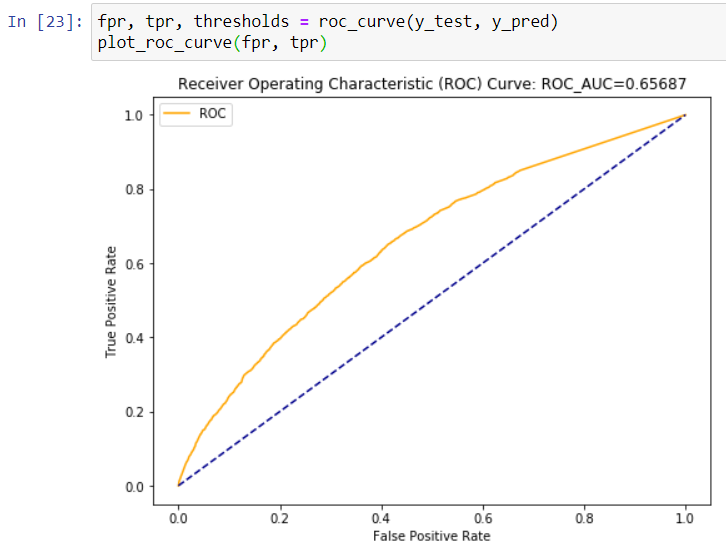
accuracy 0.65 40000

macro avg 0.54 0.61 0.51 40000

weighted avg 0.85 0.65 0.72 40000

Plotting and Checking PR-AUC Score and ROC-AUC Score

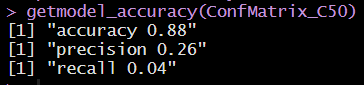


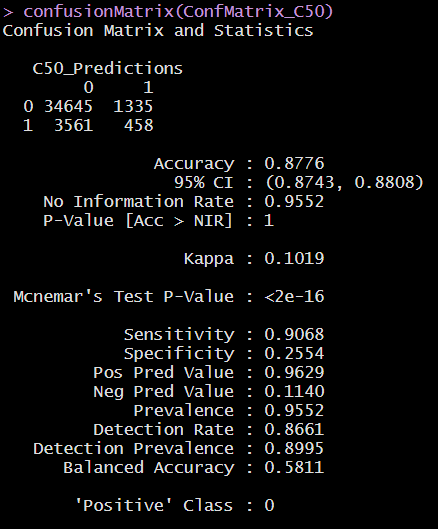


**R Code**

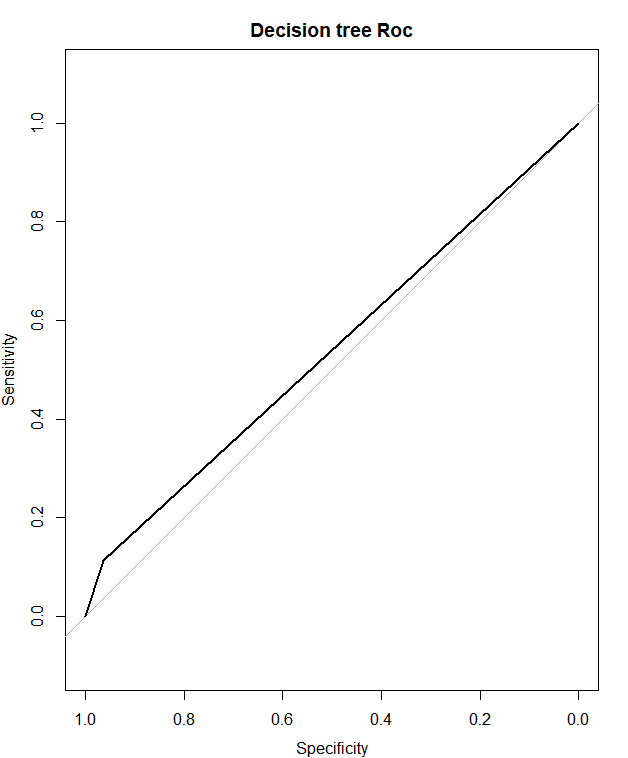
|  |
| --- |
| ############## Decision Tree ##############  #Develop Model on training data  C50\_model = C5.0(target ~., train)  #Summary of DT model  summary(C50\_model)  #Lets predict for test cases  C50\_Predictions = predict(C50\_model, test[,-1],type='class')  ##Evaluate the performance of classification model  ConfMatrix\_C50 = table(test[,1], C50\_Predictions)  #print model accuracy  getmodel\_accuracy(ConfMatrix\_C50) |

Output-





|  |
| --- |
| #get Auc score  C50\_Predictions<-as.numeric(C50\_Predictions)  roc=roc(test[,1], C50\_Predictions )  print(roc)  # plot roc\_auc curve  plot(roc ,main="Decision tree Roc") |

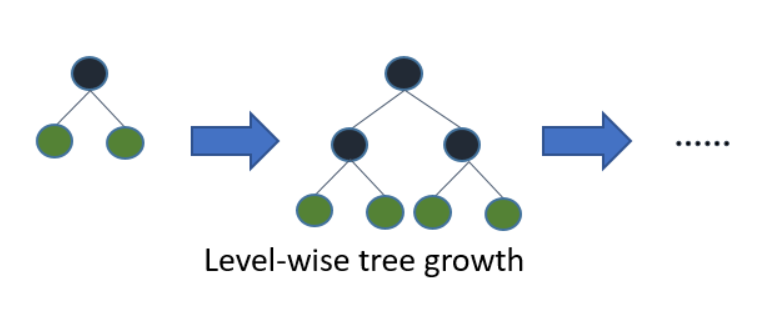


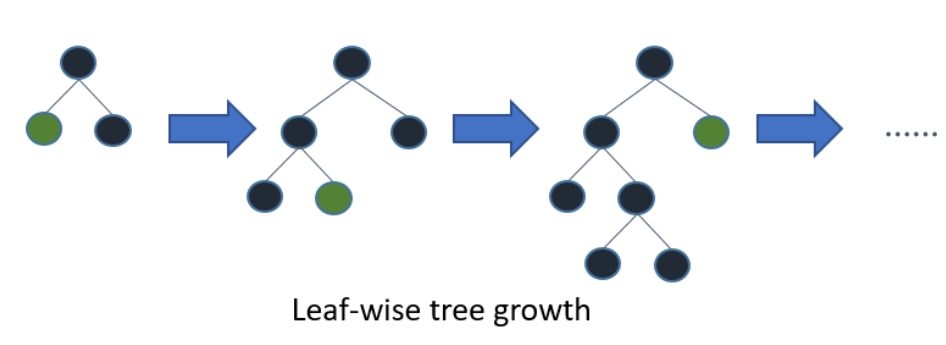
Area under the curve: 0.5384

### **2.2.5 Light GBM**

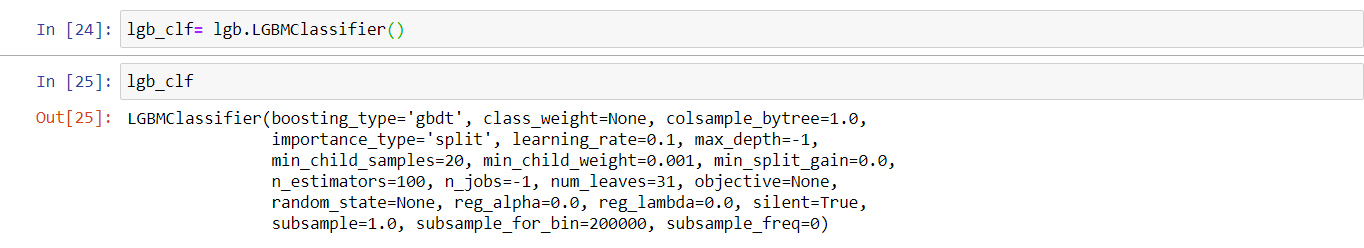
Light GBM is a gradient boosting framework that uses tree based learning algorithm. It grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

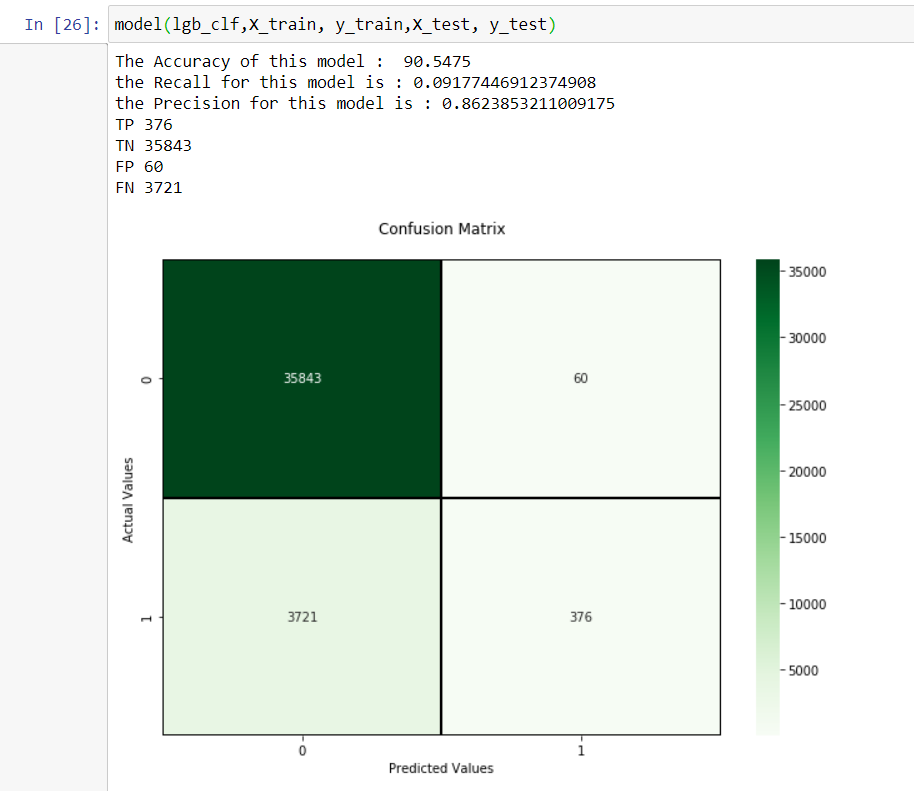
It is ‘Light’ because of its high speed. It can handle large data, requires low memory to run and focuses on accuracy of results.





**Python Code**





---------------Classification Report----------------

precision recall f1-score support

0 0.91 1.00 0.95 35903

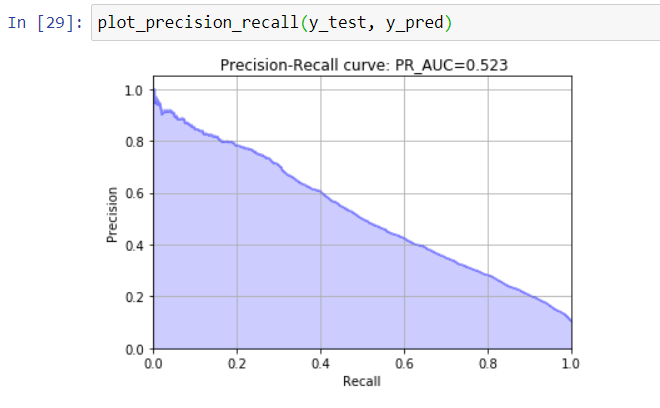
1 0.86 0.09 0.17 4097

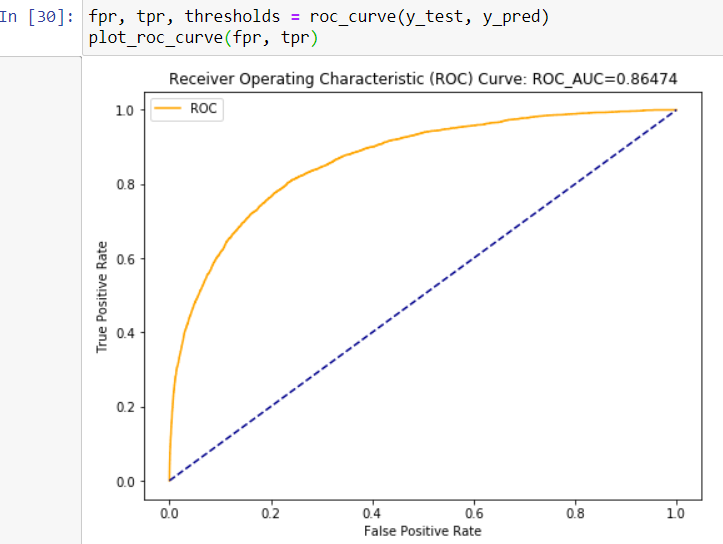
accuracy 0.91 40000

macro avg 0.88 0.55 0.56 40000

weighted avg 0.90 0.91 0.87 40000

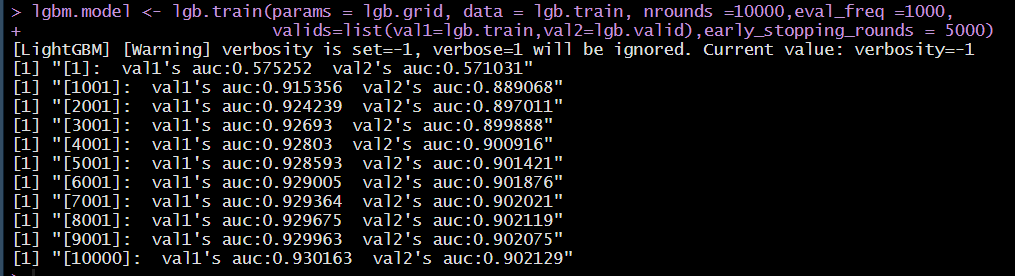
Plotting and Checking PR-AUC Score and ROC-AUC Score



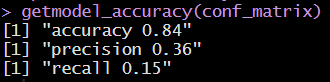


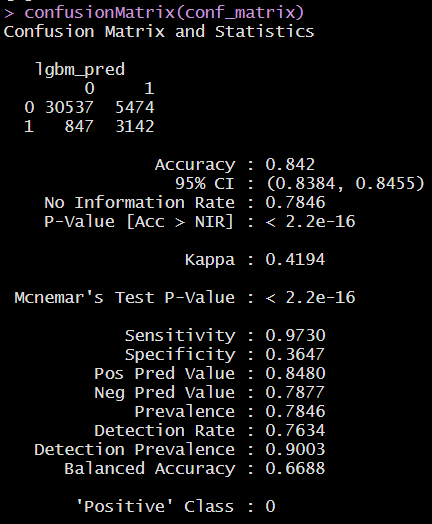
**R Code**

|  |
| --- |
| ######### light gbm Model ########  X\_train<-as.matrix(train[,-1])  y\_train<-as.matrix(train$target)  X\_valid<-as.matrix(test[,-1])  y\_valid<-as.matrix(test$target)  test\_set<-as.matrix(test\_data[,-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)  #Selecting best hyperparameters  lgb.grid = list(objective = "binary",  metric = "auc",  boost='gbdt',  max\_depth=-1,  boost\_from\_average='false',  min\_sum\_hessian\_in\_leaf = 12,  feature\_fraction = 0.05,  bagging\_fraction = 0.45,  bagging\_freq = 5,  learning\_rate=0.02,  tree\_learner='serial',  num\_leaves=20,  num\_threads=5,  min\_data\_in\_bin=150,  min\_gain\_to\_split = 30,  min\_data\_in\_leaf = 90,  verbosity=-1,  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) |

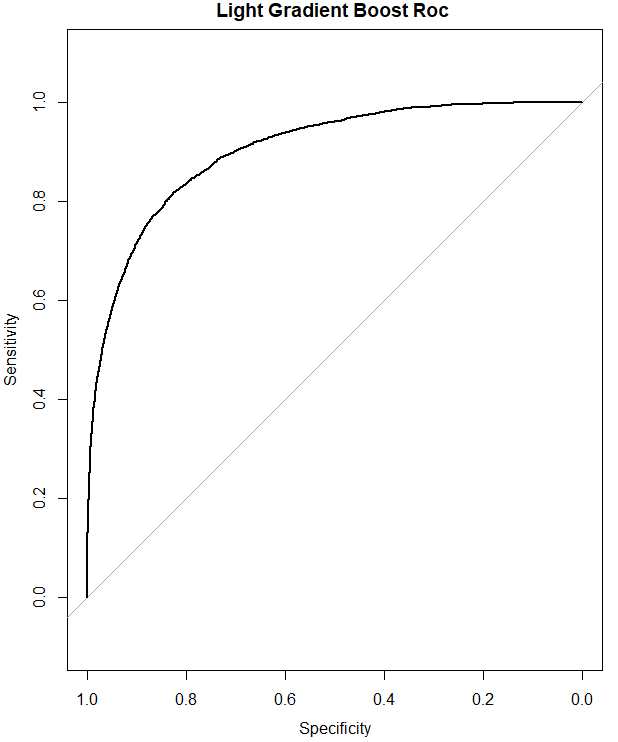


|  |
| --- |
| #lgbm model performance on test data  lgbm\_pred\_prob <- predict(lgbm.model,as.matrix(test[,-1]))  print(lgbm\_pred\_prob)  #Convert to binary output (1 and 0) with threshold 0.5  lgbm\_pred<-ifelse(lgbm\_pred\_prob>0.5,1,0)  print(lgbm\_pred)  #create confusion matrix  conf\_matrix= table(test[,1] , lgbm\_pred)  #print model accuracy  getmodel\_accuracy(conf\_matrix) |





|  |
| --- |
| # get auc  roc=roc(test[,1], lgbm\_pred\_prob)  print(roc )  # plot roc \_auc plot  plot(roc ,main ="Light Gradient Boost Roc ") |



Area under the curve: 0.9021

# **Chapter 3: Conclusion**

## **3.1 Model Selection**

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

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

Since our dataset is an imbalance class dataset, where the proportion of positive samples is low (around 10%),

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

1. Decision Tree model performed poorly on our imbalanced data.

3. Logistic regression model performed average on our imbalanced data.

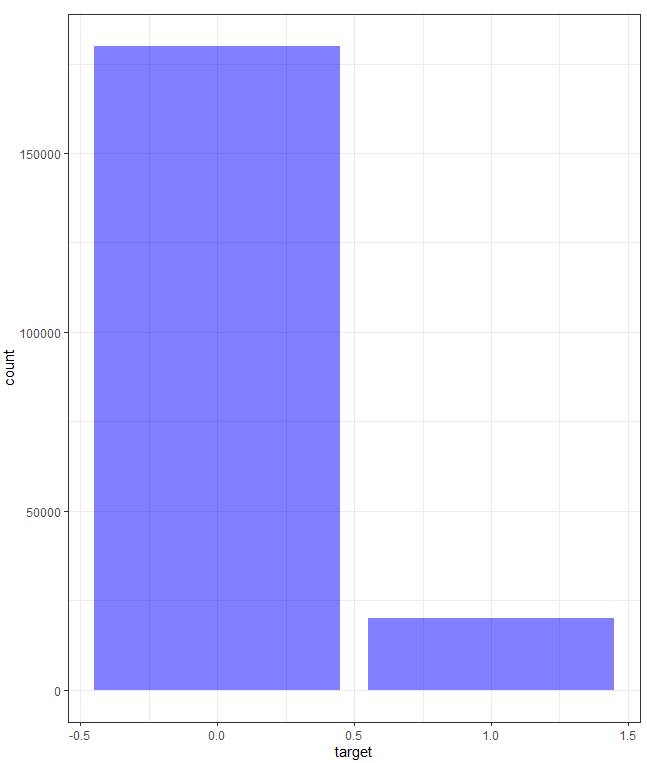
4. Light GBM model performed well on imbalanced data.

Finally, Light GBM is our best choice for identifying which customers will make a specific transaction in the future, irrespective of the amount of money transacted as we got good set of AUC scores in both python and R environment

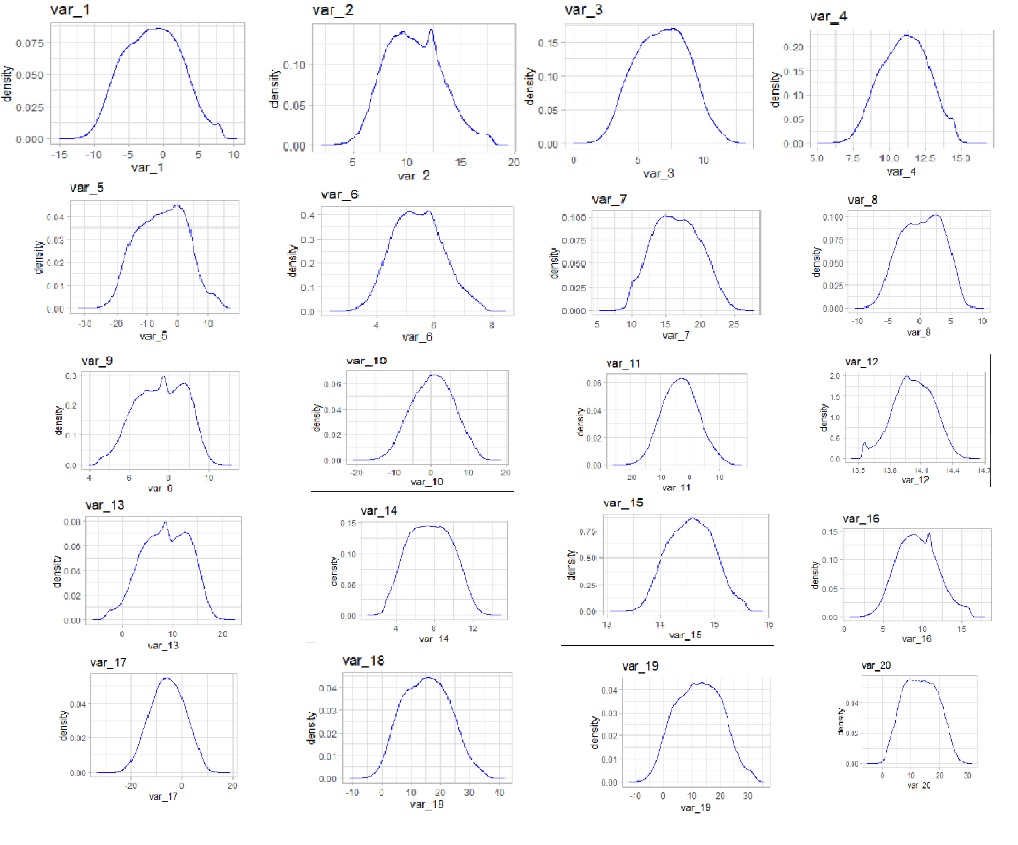
# **Appendix A – Extra Figures**

**Some ggplot2 visualizations**

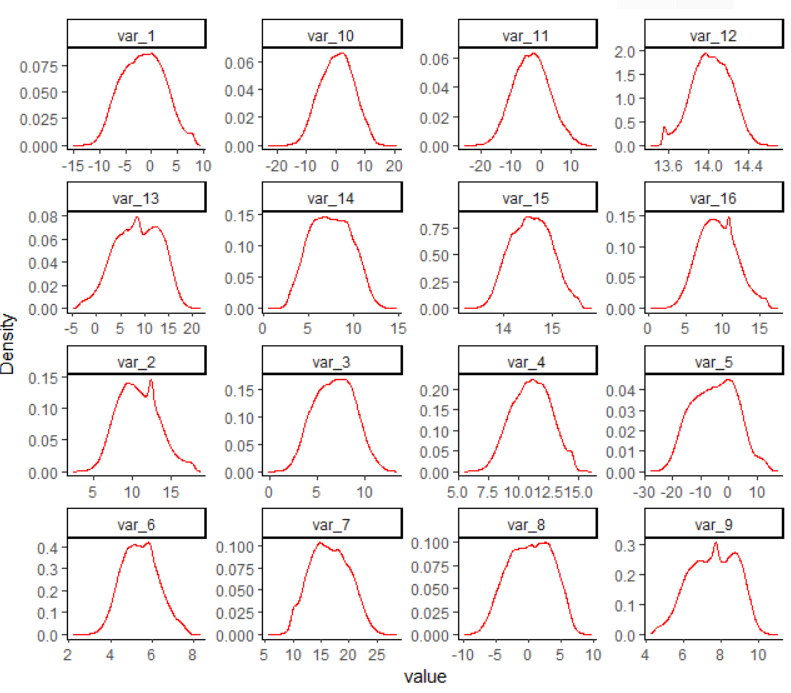
Target Value Distribution



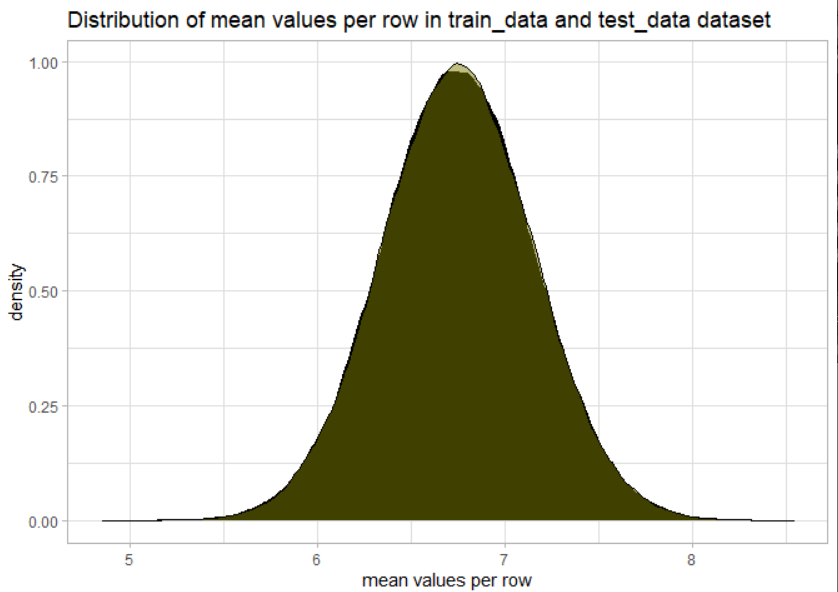
Distribution of train\_data attributes from 3 to 22

****

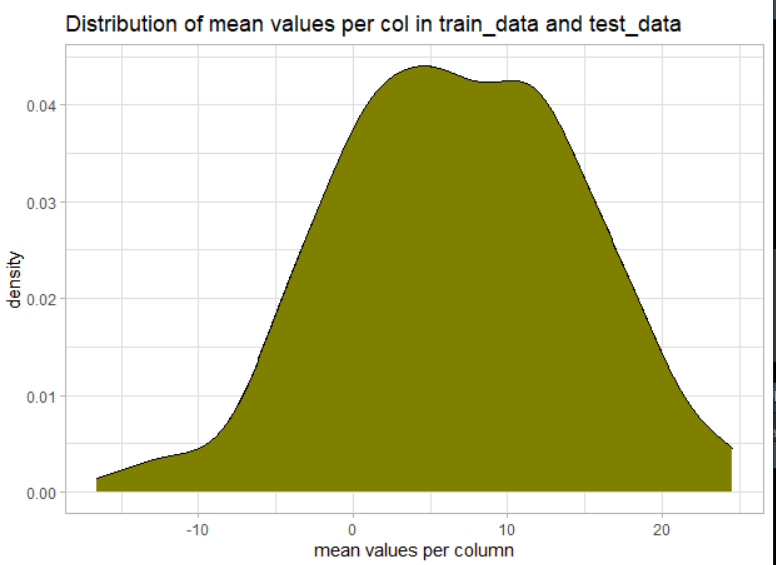
Distribution of test\_data attributes from 2 to 18



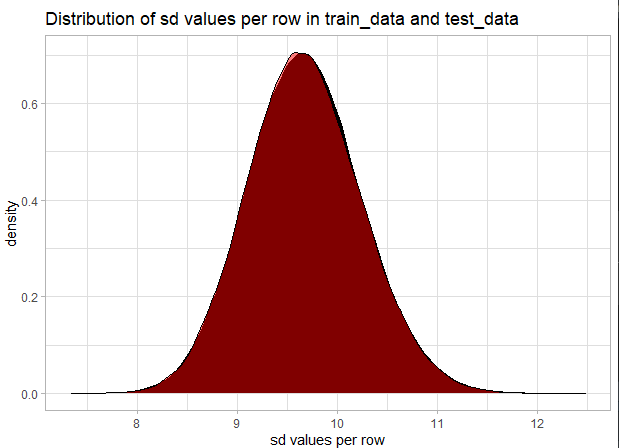
Mean values per row in train\_data and test\_data .



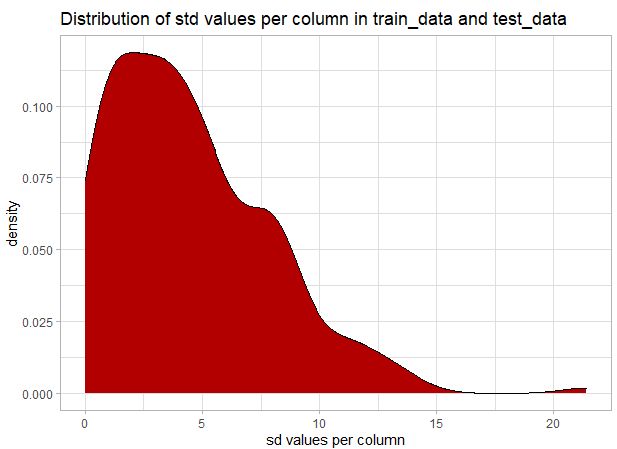
mean values per column in train\_data and test\_data



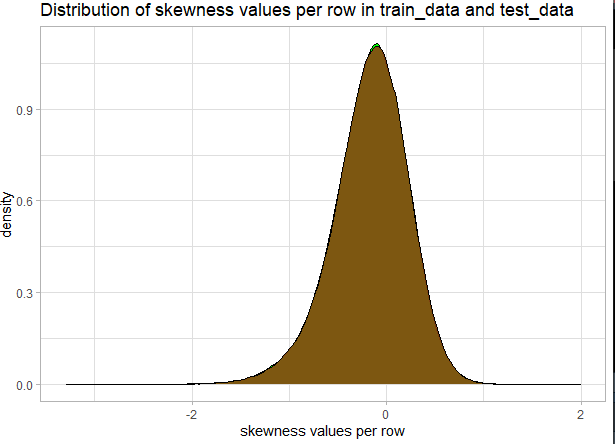
Standard deviation values per row in train\_data and test\_data



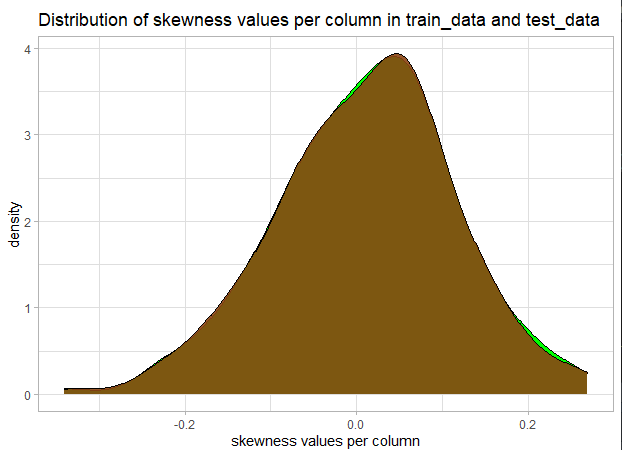
Standard deviation values per column in train\_data and test\_data



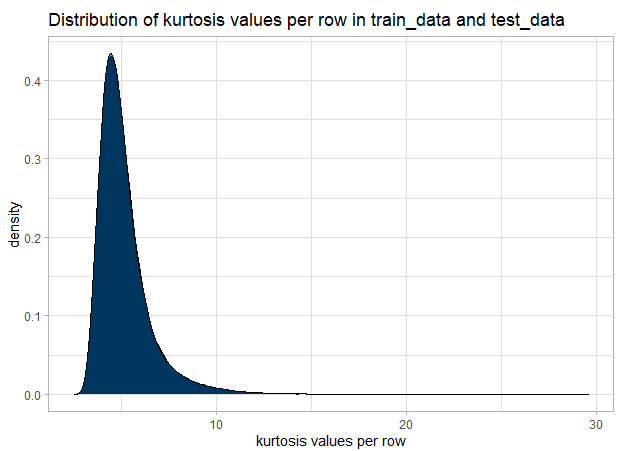
Skewness per row in train\_data and test\_data



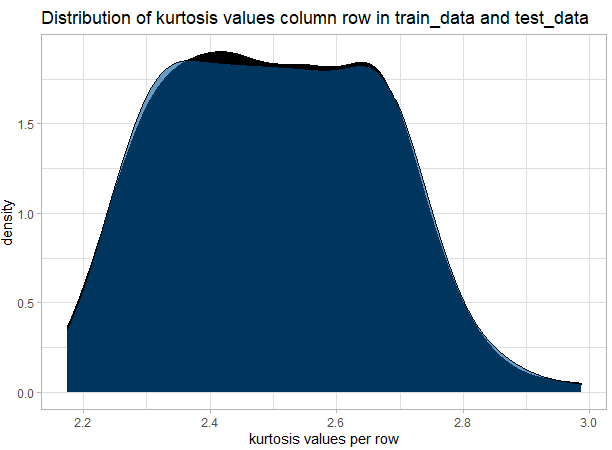
Skewness per column in train\_data and test\_data



Kurtosis per row in train\_data and test\_data



Kurtosis per column in train\_data and test\_data



# **Appendix B – Complete Python and R Code**

**Python Code**

|  |
| --- |
| #LOAD THE DATA  import os  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  import numpy as np  from sklearn.decomposition import PCA  from sklearn.model\_selection import train\_test\_split  from sklearn.linear\_model import LogisticRegression  from sklearn.preprocessing import StandardScaler  from sklearn.metrics import accuracy\_score,precision\_score,recall\_score  from sklearn.metrics import classification\_report,roc\_auc\_score,roc\_curve, auc,confusion\_matrix  from sklearn.metrics import precision\_recall\_curve  from sklearn.tree import DecisionTreeClassifier  import lightgbm as lgb  #set working directory  path = "C:/Users/jerin/Desktop/PYTHON WORK/PYTHON PROJECT/EDWISOR PROJECTS/SANTANDER CUSTOMER PREDICTION"  os.chdir(path)  os.getcwd()  ##load the data  train = pd.read\_csv(path + "/train.csv")  test = pd.read\_csv(path + "/test.csv")  print("Train data size : \t{}\nTest data Size : \t{}".format(train.shape, test.shape))  train.head(2)  test.head(2)  #target value distribution using matplotlib(bar chart)  plt.figure(figsize=(8,6))  names = ["0","1"]  values = train["target"].value\_counts(normalize=True)\*100  bar\_plot= plt.bar(names,values,color = ["gold","hotpink"])  #function to label the bars  def label\_bars(bar\_chart):  for i in bar\_plot:  height = i.get\_height()  plt.text(i.get\_x()+i.get\_width()/2., 1.005\*height,'%g'%(height),ha='center',va='bottom')  label\_bars(bar\_plot)  plt.grid()  plt.savefig("target value distribution")  # MISSING VALUE ANALYSIS  def find\_missing\_values(data\_frame):  # check for missing values and convert it into dataframe  df = pd.DataFrame(data\_frame.isnull().sum())  # rename columns of the dataframe  df = df.rename(columns = {0:"Count"})  # add a percentage variable  df["Percentage"] = (df["Count"]/len(data\_frame))\*100  # add a type variable to data types  df["Type"] = data\_frame.dtypes  # sorting values of the dataframe in descending order according to missing value count  df = df.sort\_values(by = "Count",ascending = False)  # transpose for better readability  df = df.transpose()  return df  find\_missing\_values(train)  find\_missing\_values(test)  train.describe()  test.describe()  # VISUALIZATIONS  features = [a for a in train.columns if a not in ["ID\_code","target"]]  plt.figure(figsize=(16,6))  sns.set\_style('whitegrid')  plt.title("Distribution of mean values per row in the train and test set")  sns.distplot(train[features].mean(axis=1), color="black",bins=120, label='train')  sns.distplot(test[features].mean(axis=1), color="olive",bins=120, label='test')  plt.legend();  plt.show()  plt.figure(figsize=(16,6))  plt.title("Distribution of mean values per column in train and test set")  sns.distplot(train[features].mean(axis=0), color="black",kde=True,bins=120, label='train')  sns.distplot(test[features].mean(axis=0), color="olive",kde=True,bins=120, label='test')  plt.legend()  plt.grid()  plt.figure(figsize=(16,6))  plt.title("Distribution of mean values per column in train and test set")  sns.distplot(train[features].mean(axis=0), color="black",kde=True,bins=120, label='train')  sns.distplot(test[features].mean(axis=0), color="olive",kde=True,bins=120, label='test')  plt.legend()  plt.grid()  t0 = train.loc[train["target"] == 0]  t1 = train.loc[train["target"] == 1]  plt.figure(figsize=(16,6))  plt.title("Distribution of mean values per column in the train set, grouped by value of target")  sns.distplot(t0[features].mean(axis=0),color="blue", kde=True,bins=120, label='target = 0')  sns.distplot(t1[features].mean(axis=0),color="green", kde=True,bins=120, label='target = 1')  plt.legend();  plt.show()  plt.figure(figsize=(16,6))  plt.title("Distribution of standard deviation of values per row in the train and test set")  sns.distplot(train[features].std(axis=1),color="black",kde=True,bins=120, label='train')  sns.distplot(test[features].std(axis=1),color="red", kde=True,bins=120, label='test')  plt.legend();  plt.show()  plt.figure(figsize=(16,6))  plt.title("Distribution of standard deviation of values per Column in the train and test set")  sns.distplot(train[features].std(axis=0),color="black",kde=True,bins=120, label='train')  sns.distplot(test[features].std(axis=0),color="red", kde=True,bins=120, label='test')  plt.legend();  plt.show()  t0 = train.loc[train["target"] == 0]  t1 = train.loc[train["target"] == 1]  plt.figure(figsize=(16,6))  plt.title("Distribution of standard deviation of values per row in the train set, grouped by value of target")  sns.distplot(t0[features].std(axis=1),color="blue", kde=True,bins=120, label='target = 0')  sns.distplot(t1[features].std(axis=1),color="red", kde=True,bins=120, label='target = 1')  plt.legend();  plt.show()  t0 = train.loc[train['target'] == 0]  t1 = train.loc[train["target"] == 1]  plt.figure(figsize=(16,6))  plt.title("Distribution of standard deviation of values per column in the train set, grouped by value of target")  sns.distplot(t0[features].std(axis=0),color="blue", kde=True,bins=120, label='target = 0')  sns.distplot(t1[features].std(axis=0),color="red", kde=True,bins=120, label='target = 1')  plt.legend();  plt.show()  plt.figure(figsize=(16,6))  plt.title("Distribution of skewness of values per row in the train and test set")  sns.distplot(train[features].skew(axis=1),color="green", kde=True,bins=120, label='train')  sns.distplot(test[features].skew(axis=1),color="saddlebrown", kde=True,bins=120, label='test')  plt.legend();  plt.show()  plt.figure(figsize=(16,6))  plt.title("Distribution of skewness of values per column in the train and test set")  sns.distplot(train[features].skew(axis=0),color="green", kde=True,bins=120, label='train')  sns.distplot(test[features].skew(axis=0),color="saddlebrown", kde=True,bins=120, label='test')  plt.legend();  plt.show()  plt.figure(figsize=(16,6))  plt.title("Distribution of kurtosis values per row in the train and test set")  sns.distplot(train[features].kurtosis(axis=1),color="dodgerblue", kde=True,bins=120, label='train')  sns.distplot(test[features].kurtosis(axis=1),color="black", kde=True,bins=120, label='test')  plt.legend();  plt.show()  plt.figure(figsize=(16,6))  plt.title("Distribution of kurtosis values per column in the train and test set")  sns.distplot(train[features].kurtosis(axis=0),color="dodgerblue", kde=True,bins=120, label='train')  sns.distplot(test[features].kurtosis(axis=0),color="black", kde=True,bins=120, label='test')  plt.legend();  plt.show()  plt.figure(figsize=(16,6))  plt.title("Distribution of Min values per row in the train and test set")  sns.distplot(train[features].min(axis=1),color="red", kde=True,bins=120, label='train')  sns.distplot(test[features].min(axis=1),color="orange", kde=True,bins=120, label='test')  plt.legend();  plt.show()  plt.figure(figsize=(16,6))  plt.title("Distribution of Min values per column in the train and test set")  sns.distplot(train[features].min(axis=0),color="red", kde=True,bins=120, label='train')  sns.distplot(test[features].min(axis=0),color="orange", kde=True,bins=120, label='test')  plt.legend();  plt.show()  plt.figure(figsize=(16,6))  plt.title("Distribution of Max values per row in the train and test set")  sns.distplot(train[features].max(axis=1),color="brown", kde=True,bins=120, label='train')  sns.distplot(test[features].max(axis=1),color="yellow", kde=True,bins=120, label='test')  plt.legend();  plt.show()  plt.figure(figsize=(16,6))  plt.title("Distribution of Max values per column in the train and test set")  sns.distplot(train[features].max(axis=0),color="brown", kde=True,bins=120, label='train')  sns.distplot(test[features].max(axis=0),color="yellow", kde=True,bins=120, label='test')  plt.legend();  plt.show()  #OUTLIER ANALYSIS  # function to plot multiple boxplots  def show\_boxplot(df,feature):  plt.figure(figsize = (18, 24))  sns.set\_style('whitegrid')  for i in enumerate(feature):  plt.subplot(10, 5,i[0]+1)  sns.boxplot(i[1], data = df,color= "darksalmon")  plt.xlabel(i[1],fontsize=11)  plt.tick\_params(axis='x', labelsize=6, pad=-6)  plt.tick\_params(axis='y', labelsize=6)  # From var\_0 to var\_49  features\_2to52 = train.columns.values[2:52]  show\_boxplot(train,features\_2to52)  # From var\_50 to var\_99  features\_52to102 = train.columns.values[52:102]  show\_boxplot(train,features\_52to102)  # From var\_100 to var\_149  features\_102to152 = train.columns.values[102:152]  show\_boxplot(train,features\_102to152)  # From var\_150 to var\_199  features\_152to202 = train.columns.values[152:202]  show\_boxplot(train,features\_152to202)  # #Detect from IQR and delete outliers from data  #iqr stands for inter quartile range  Q1 = train.quantile(0.25)  Q3 = train.quantile(0.75)  IQR = Q3 - Q1  train\_in = train[~((train < (Q1 - 1.5 \* IQR)) |(train > (Q3 + 1.5 \* IQR))).any(axis=1)]  train\_out = train[((train < (Q1 - 1.5 \* IQR)) |(train > (Q3 + 1.5 \* IQR))).any(axis=1)]  print("train\_in.shape:",train\_in.shape)  print("train\_out.shape:",train\_out.shape)    train\_in['target'].value\_counts()  # comparing the 'train' and 'df\_out' dataset,  # we can say that all the data points with target equals to 1 are present as outliers  train\_out['target'].value\_counts()  train['target'].value\_counts()  #FEATURE SELECTION  #CORRELATION ANALYIS  #Set the width and height of the plot  plt.subplots(figsize=(12, 10))  #Generate correlation matrix  corr = train.corr()  #Plot using seaborn library  sns.heatmap(corr);  corr  (corr['target']).sort\_values(ascending=False).head(50)  correlation = train[features].corr().abs().unstack().sort\_values().reset\_index()  correlation = correlation[correlation['level\_0'] != correlation['level\_1']]  correlation.tail(10)  # Another Method to compute the correlation matrix and find out +vely and -vely correlated features  corr1 = train[features].corr()  np.fill\_diagonal(corr1.values,np.nan)  corr1.max().max(),corr1.min().min()  #DIMENSIONALITY REDUCTION  #PCA  # At this point of time we should check for among 200 features which variables are useful for us.  # As the features are anonymous. This can be done by using PCA  # However, since we found that the correlation between different features in the training dataset is not that significant, so using PCA might not be meaningful  # scale the data before using PCA  scaler = StandardScaler()  X\_train = scaler.fit\_transform(train[features])  X\_test = scaler.transform(test[features])  pca = PCA()  a = pca.fit\_transform(X\_train)  b = pca.transform(X\_test)  explained\_variance = pca.explained\_variance\_ratio\_  plt.plot(np.arange(200),np.cumsum(explained\_variance))  plt.xlabel('number of components')  plt.ylabel('cumulative explained variance');  #MODELLING  # Splitting the train and test data  Target = train['target']  # Input dataset for Train and Test  train\_inp = train.drop(columns = ['target', 'ID\_code'])  test\_inp = test.drop(columns = ['ID\_code'])  X\_train, X\_test, y\_train, y\_test = train\_test\_split(train\_inp, Target, test\_size=0.2, random\_state = 42)  print ("X\_train: ", X\_train.shape)  print ("y\_train: ", y\_train.shape)  print("X\_test: ", X\_test.shape)  print ("y\_test: ", y\_test.shape)  ## Created a Model function for modeling with confusion matrix and classification report  def model(model,features\_train,labels\_train,features\_test,labels\_test):  clf= model  clf.fit(features\_train,labels\_train)  pred=clf.predict(features\_test)  cnf\_matrix=confusion\_matrix(labels\_test,pred)  print("The Accuracy of this model : ",accuracy\_score(labels\_test,pred)\*100 )  print("the Recall for this model is :",cnf\_matrix[1,1]/(cnf\_matrix[1,1]+cnf\_matrix[1,0]))  print("the Precision for this model is :",cnf\_matrix[1,1]/(cnf\_matrix[1,1]+cnf\_matrix[0,1]))  fig= plt.figure(figsize=(10,7))  print("TP",cnf\_matrix[1,1]) # no of true transactions which are predicted as true  print("TN",cnf\_matrix[0,0]) # no of false transaction which are predicted as false  print("FP",cnf\_matrix[0,1]) # no of false transactions which are predicted as true  print("FN",cnf\_matrix[1,0]) # no of true transactions which are predicted as false  sns.heatmap(cnf\_matrix,cmap="Greens",annot=True,fmt="d",linewidths=1,linecolor='black')  plt.title("Confusion Matrix\n")  plt.xlabel("Predicted Values")  plt.ylabel("Actual Values")  plt.show()  print("\n---------------Classification Report----------------\n")  print(classification\_report(labels\_test,pred))  #LOGISTIC REGRESSION  model(LogisticRegression(class\_weight='balanced',max\_iter=10000),X\_train,y\_train, X\_test,y\_test)  logreg\_scaled = LogisticRegression(class\_weight='balanced',max\_iter=10000).fit(X\_train,y\_train)  y\_pred = logreg\_scaled.predict\_proba(X\_test)[:,1]  #Precision-Recall is a useful measure of success of prediction when the classes are very imbalanced  def plot\_precision\_recall(y\_test, y\_pred):  precision, recall, threshold = precision\_recall\_curve(y\_test, y\_pred)  plt.step(recall, precision, color='b', alpha=0.3,where='post')  plt.fill\_between(recall, precision, alpha=0.2, color='b')  plt.xlabel('Recall')  plt.ylabel('Precision')  # set the y range  plt.ylim([0.0, 1.05])  # set the x raneg  plt.xlim([0.0, 1.0])  plt.title(' Precision-Recall curve: PR\_AUC={0:0.3f}'.format( auc(recall, precision)))  plt.grid()  plot\_precision\_recall(y\_test, y\_pred)  def plot\_roc\_curve(fpr, tpr):  fig= plt.figure(figsize=(8,6))  plt.plot(fpr, tpr, color='orange', label='ROC')  plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.title('Receiver Operating Characteristic (ROC) Curve: ROC\_AUC={0:0.5f}'.format(roc\_auc\_score(y\_test, y\_pred)))  plt.legend()  plt.show()  fpr, tpr, thresholds = roc\_curve(y\_test,y\_pred)  plot\_roc\_curve(fpr, tpr)  # DECISION TREE  tree\_clf = DecisionTreeClassifier(class\_weight='balanced', random\_state = 42,  max\_features = 0.7, min\_samples\_leaf = 80)  tree\_clf  model(tree\_clf,X\_train, y\_train,X\_test, y\_test)  y\_pred= tree\_clf.predict\_proba(X\_test)[:, 1]  plot\_precision\_recall(y\_test, y\_pred)  fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)  plot\_roc\_curve(fpr, tpr)  #LGBM  lgb\_clf= lgb.LGBMClassifier()  lgb\_clf  model(lgb\_clf,X\_train, y\_train,X\_test, y\_test)  y\_pred= lgb\_clf.predict\_proba(X\_test)[:, 1]  plot\_precision\_recall(y\_test, y\_pred)  fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)  plot\_roc\_curve(fpr, tpr) |

**R Code**

|  |
| --- |
| rm(list=ls(all=T))  library(ggplot2)  library(tidyverse)  library(DataExplorer)  library(moments)  library(C50)  library(glmnet)  library(pROC)  library(lightgbm)  #load the data  setwd("C:/Users/jerin/Desktop/R work/EDWISOR PROJECT")  train\_data= read.csv("train.csv", header = T)  test\_data=read.csv("test.csv", header = T)  head(train\_data)  #Dimension of train\_data and test\_data data  dim(train\_data)  dim(test\_data)  #storing ID\_code of test\_data train\_data data  train\_data\_ID\_code\_orignal = train\_data$ID\_code  test\_data\_ID\_code\_orignal = test\_data$ID\_code  #removing Idcode from orginal dataset  train\_data$ID\_code=NULL  test\_data$ID\_code=NULL  #check dimension of dataset after removing column  (dim(train\_data))  (dim(test\_data))  #convert to factor  train\_data$target<-as.factor(train\_data$target)  #Target Value Distribution  table(train\_data$target)/length(train\_data$target)\*100  plot1<-ggplot(train\_data,aes(target))+theme\_bw()+geom\_bar(stat='count',fill='blue',alpha=0.5)  plot1  # Missing Value Analysis  missing\_train\_data\_values= (apply(train\_data,2,function(x)sum(is.na(x))))  sum(missing\_train\_data\_values)  missing\_test\_data\_values = (apply(test\_data,2,function(x)sum(is.na(x))))  sum(missing\_test\_data\_values)  #Summary of the dataset  str(train\_data)  str(test\_data)  ########## Visualisations ###################  #Distribution of train\_data attributes from 3 to 102  for (var in names(train\_data)[c(3:102)]){  target<-train\_data$target  plot<-ggplot(train\_data, aes(x=.data[[var]],fill=target)) +  geom\_density(kernel='gaussian',color = "blue") + ggtitle(var)+theme\_light()  print(plot)  }  #Distribution of train\_data attributes from 103 to 202  for (var in names(train\_data)[c(103:202)]){  target<-train\_data$target  plot<-ggplot(train\_data, aes(x=.data[[var]], fill=target)) +  geom\_density(kernel='gaussian',color="red") + ggtitle(var)+theme\_light()  print(plot)  }  #Distribution of test\_data attributes from 2 to 101  plot\_density(test\_data[,c(2:101)], ggtheme = theme\_classic(),geom\_density\_args = list(color='red'))  #Distribution of test\_data attributes from 102 to 201  plot\_density(test\_data[,c(102:201)], ggtheme = theme\_classic(),geom\_density\_args = list(color='red'))  #Applying the function to find mean values per row in train\_data and test\_data data.  train\_data\_mean<-(apply(train\_data[,-c(1,2)],MARGIN=1,FUN=mean))  test\_data\_mean<-(apply(test\_data[,-c(1)],MARGIN=1,FUN=mean))  ggplot()+  #Distribution of mean values per row in train\_data data  geom\_density(data=train\_data[,-c(1,2)],aes(x=train\_data\_mean),kernel='gaussian',show.legend=TRUE,fill="black")+theme\_light()+  #Distribution of mean values per row in test\_data data  geom\_density(data=test\_data[,-c(1)],aes(x=test\_data\_mean),kernel='gaussian',show.legend=TRUE,fill="#808000",alpha=0.5)+  labs(x='mean values per row',title="Distribution of mean values per row in train\_data and test\_data")  #Applying the function to find mean values per column in train\_data and test\_data data.  train\_data\_mean<-(apply(train\_data[,-c(1,2)],MARGIN=2,FUN=mean))  test\_data\_mean<-apply(test\_data[,-c(1)],MARGIN=2,FUN=mean)  ggplot()+  #Distribution of mean values per column in train\_data data  geom\_density(aes(x=train\_data\_mean),kernel='gaussian',show.legend=TRUE,fill="black")+theme\_light()+  #Distribution of mean values per column in test\_data data  geom\_density(aes(x=test\_data\_mean),kernel='gaussian',show.legend=TRUE,fill="#808000")+  labs(x='mean values per column',title="Distribution of mean values per col in train\_data and test\_data")  #Applying the function to find standard deviation values per row in train\_data and test\_data.  train\_data\_sd<-apply(train\_data[,-c(1,2)],MARGIN=1,FUN=sd)  test\_data\_sd<-apply(test\_data[,-c(1)],MARGIN=1,FUN=sd)  ggplot()+  #Distribution of sd values per row in train\_data data  geom\_density(data=train\_data[,-c(1,2)],aes(x=train\_data\_sd),kernel='gaussian',show.legend=TRUE,fill='black')+theme\_light()+  #Distribution of mean values per row in test\_data data  geom\_density(data=test\_data[,-c(1)],aes(x=test\_data\_sd),kernel='gaussian',show.legend=TRUE,fill='red',alpha=0.5)+  labs(x='sd values per row',title="Distribution of sd values per row in train\_data and test\_data")  #Applying the function to find sd values per column in train\_data and test\_data data.  train\_data\_sd<-apply(train\_data[,-c(1,2)],MARGIN=2,FUN=sd)  test\_data\_sd<-apply(test\_data[,-c(1)],MARGIN=2,FUN=sd)  ggplot()+  #Distribution of sd values per row in train\_data data  geom\_density(aes(x=train\_data\_sd),kernel='gaussian',show.legend=TRUE,fill='black')+theme\_light()+  #Distribution of mean values per row in test\_data data  geom\_density(aes(x=test\_data\_sd),kernel='gaussian',show.legend=TRUE,fill='red',alpha=0.7)+  labs(x='sd values per column',title="Distribution of std values per column in train\_data and test\_data")  #Applying the function to find skewness values per row in train\_data and test\_data.  train\_data\_skew<-apply(train\_data[,-c(1,2)],MARGIN=1,FUN=skewness)  test\_data\_skew<-apply(test\_data[,-c(1)],MARGIN=1,FUN=skewness)  ggplot()+  #Distribution of skewness values per row in train\_data data  geom\_density(aes(x=train\_data\_skew),kernel='gaussian',show.legend=TRUE,fill='green')+theme\_light()+  #Distribution of skewness values per column in test\_data data  geom\_density(aes(x=test\_data\_skew),kernel='gaussian',show.legend=TRUE,fill="#8b4513",alpha= 0.9)+  labs(x='skewness values per row',title="Distribution of skewness values per row in train\_data and test\_data")  #Applying the function to find skewness values per column in train\_data and test\_data  train\_data\_skew<-apply(train\_data[,-c(1,2)],MARGIN=2,FUN=skewness)  test\_data\_skew<-apply(test\_data[,-c(1)],MARGIN=2,FUN=skewness)  ggplot()+  #Distribution of skewness values per column in train\_data data  geom\_density(aes(x=train\_data\_skew),kernel='gaussian',show.legend=TRUE,fill='green')+theme\_light()+  #Distribution of skewness values per column in test\_data data  geom\_density(aes(x=test\_data\_skew),kernel='gaussian',show.legend=TRUE,fill="#8b4513",alpha=0.9)+  labs(x='skewness values per column',title="Distribution of skewness values per column in train\_data and test\_data")  #Applying the function to find kurtosis values per row in train\_data and test\_data.  train\_data\_kurtosis<-apply(train\_data[,-c(1,2)],MARGIN=1,FUN=kurtosis)  test\_data\_kurtosis<-apply(test\_data[,-c(1)],MARGIN=1,FUN=kurtosis)  ggplot()+  #Distribution of kurtosis values per row in train\_data data  geom\_density(aes(x=train\_data\_kurtosis),kernel='gaussian',show.legend=TRUE,fill='black')+theme\_light()+  #Distribution of kurtosis values per row in test\_data data  geom\_density(aes(x=test\_data\_kurtosis),kernel='gaussian',show.legend=TRUE,fill='#005A9C',alpha=0.6)+  labs(x='kurtosis values per row',title="Distribution of kurtosis values per row in train\_data and test\_data")  #Applying the function to find kurtosis values per column in train\_data and test\_data.  train\_data\_kurtosis<-apply(train\_data[,-c(1,2)],MARGIN=2,FUN=kurtosis)  test\_data\_kurtosis<-apply(test\_data[,-c(1)],MARGIN=2,FUN=kurtosis)  ggplot()+  #Distribution of kurtosis values per column in train\_data data  geom\_density(aes(x=train\_data\_kurtosis),kernel='gaussian',show.legend=TRUE,fill='black')+theme\_light()+  #Distribution of kurtosis values per column in test\_data data  geom\_density(aes(x=test\_data\_kurtosis),kernel='gaussian',show.legend=TRUE,fill='#005A9C',alpha=0.6)+  labs(x='kurtosis values per row',title="Distribution of kurtosis values column row in train\_data and test\_data")  #Correlations (method 1)  cormat <- cor(train\_data[,-c(1,2)])  summary(cormat[upper.tri(cormat)]) #Correlations between features nearly zero.  #Correlations in train\_data data(method 2)  #convert factor to int  train\_data$target<-as.numeric(train\_data$target)  train\_data\_correlations<-cor(train\_data[,c(2:202)])  train\_data\_correlations  #Correlations in test\_data data  test\_data\_correlations<-cor(test\_data\_df[,c(2:201)])  test\_data\_correlations  ####### Modelling ###############  getmodel\_accuracy=function(conf\_matrix)  {  model\_parm =list()  tn =conf\_matrix[1,1]  tp =conf\_matrix[2,2]  fp =conf\_matrix[1,2]  fn =conf\_matrix[2,1]  p =(tp)/(tp+fp)  r =(fp)/(fp+tn)  f1=2\*((p\*r)/(p+r))  print(paste("accuracy",round((tp+tn)/(tp+tn+fp+fn),2)))  print(paste("precision",round(p ,2)))  print(paste("recall",round(r,2)))  }  ##split the data into train and test  set.seed(1234)  require("caret")  train\_data.index = createDataPartition(train\_data$target, p = .80, list = FALSE)  train = train\_data[ train\_data.index,]  test = train\_data[-train\_data.index,]  #dimension of train\_data and validation data  dim(train)  dim(test)  ##################### Logistic Regression Model ########################  logit\_model =glm(target~. ,data =train ,family='binomial')  # model summary  summary(logit\_model)  #get model predicted probality  y\_prob =predict(logit\_model , test[,-1] ,type = 'response' )  # convert probality to class according to thresshold  y\_pred = ifelse(y\_prob >0.5, 1, 0)  #create confusion matrix  conf\_matrix= table(test[,1] , y\_pred)  #print model accuracy  getmodel\_accuracy(conf\_matrix)  confusionMatrix(conf\_matrix)  # get auc  roc=roc(test[,1], y\_prob)  print(roc )  # plot roc \_auc plot  plot(roc ,main ="Logistic Regression Roc ")  ############## Decision Tree ##############  #Develop Model on training data  C50\_model = C5.0(target ~., train)  #Summary of DT model  summary(C50\_model)  #Lets predict for test cases  C50\_Predictions = predict(C50\_model, test[,-1],type='class')  ##Evaluate the performance of classification model  ConfMatrix\_C50 = table(test[,1], C50\_Predictions)  #print model accuracy  getmodel\_accuracy(ConfMatrix\_C50)  #fucntion to calculate the different error metrics  confusionMatrix(ConfMatrix\_C50)  #get Auc score  C50\_Predictions<-as.numeric(C50\_Predictions)  roc=roc(test[,1], C50\_Predictions )  print(roc)  # plot roc\_auc curve  plot(roc ,main="Decision tree Roc")  ######### light gbm Model ########  X\_train<-as.matrix(train[,-1])  y\_train<-as.matrix(train$target)  X\_valid<-as.matrix(test[,-1])  y\_valid<-as.matrix(test$target)  test\_set<-as.matrix(test\_data[,-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)  #Selecting best hyperparameters  lgb.grid = list(objective = "binary",  metric = "auc",  boost='gbdt',  max\_depth=-1,  boost\_from\_average='false',  min\_sum\_hessian\_in\_leaf = 12,  feature\_fraction = 0.05,  bagging\_fraction = 0.45,  bagging\_freq = 5,  learning\_rate=0.02,  tree\_learner='serial',  num\_leaves=20,  num\_threads=5,  min\_data\_in\_bin=150,  min\_gain\_to\_split = 30,  min\_data\_in\_leaf = 90,  verbosity=-1,  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)  #lgbm model performance on test data  lgbm\_pred\_prob <- predict(lgbm.model,as.matrix(test[,-1]))  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)  #create confusion matrix  conf\_matrix= table(test[,1] , lgbm\_pred)  #print model accuracy  getmodel\_accuracy(conf\_matrix)  confusionMatrix(conf\_matrix)  # get auc  roc=roc(test[,1], lgbm\_pred\_prob)  print(roc )  # plot roc \_auc plot  plot(roc ,main ="Light Gradient Boost Roc ") |