# Santander Customer Transaction Prediction


**By**

**Jerin Joseph**

# Table of Contents

# 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

In [3]:
```
##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 data size :       (200000, 202)
Test data Size :        (200000, 201)
```

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

In [4]:
```
train.head(2)
```

Out[4]:

| | ID_code | target | var_0 | var_1 | var_2 | var_3 | var_4 | var_5 | var_6 | var_7 | ... | var_190 | var_191 | var_192 | var_193 | var_194 | var_195 | var_196 | var |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | train_0 | 0 | 8.9255 | -6.7863 | 11.9081 | 5.093 | 11.4607 | -9.2834 | 5.1187 | 18.6266 | ... | 4.4354 | 3.9642 | 3.1364 | 1.6910 | 18.5227 | -2.3978 | 7.8784 | 8. |
| 1 | train_1 | 0 | 11.5006 | -4.1473 | 13.8588 | 5.389 | 12.3622 | 7.0433 | 5.6208 | 16.5338 | ... | 7.6421 | 7.7214 | 2.5837 | 10.9516 | 15.4305 | 2.0339 | 8.1267 | 8. |

2 rows × 202 columns

In [5]:
```
test.head(2)
```

Out[5]:

| | ID_code | var_0 | var_1 | var_2 | var_3 | var_4 | var_5 | var_6 | var_7 | var_8 | ... | var_190 | var_191 | var_192 | var_193 | var_194 | var_195 | var_196 | va |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | test_0 | 11.0656 | 7.7798 | 12.9536 | 9.4292 | 11.4327 | -2.3805 | 5.8493 | 18.2675 | 2.1337 | ... | -2.1556 | 11.8495 | -1.4300 | 2.4508 | 13.7112 | 2.4669 | 4.3654 | 1( |
| 1 | test_1 | 8.5304 | 1.2543 | 11.3047 | 5.1858 | 9.1974 | -4.0117 | 6.0196 | 18.6316 | -4.4131 | ... | 10.6165 | 8.8349 | 0.9403 | 10.1282 | 15.5765 | 0.4773 | -1.4852 | 9 |

2 rows × 201 columns

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

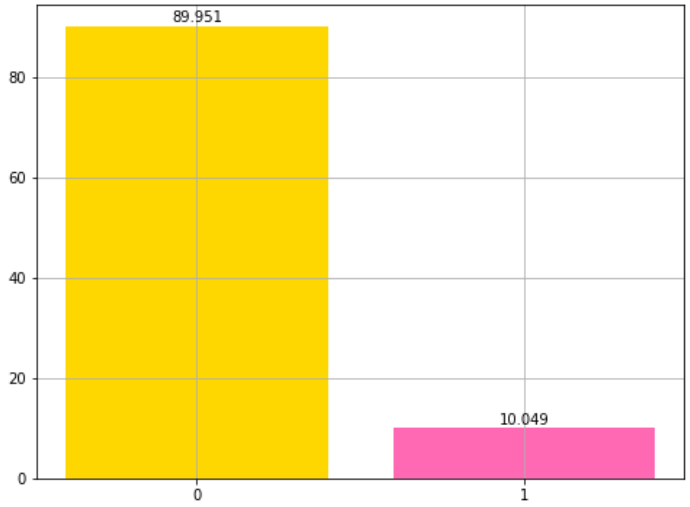
## 2.1 Exploratory Data Analysis (EDA)

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

In [6]:
```
#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")
```




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

```
In [7]: 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
```

```
In [42]: find_missing_values(train)
```

Out[42]:

| | ID_code | var_136 | var_126 | var_127 | var_128 | var_129 | var_130 | var_131 | var_132 | var_133 | ... | var_63 | var_64 | var_65 | var_66 | var_67 | var_68 | va |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Count** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | |
| **Percentage** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | |
| **Type** | object | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | ... | float64 | float64 | float64 | float64 | float64 | float64 | fl |

3 rows × 202 columns

```
In [43]: find_missing_values(test)
```

Out[43]:

| | ID_code | var_137 | var_127 | var_128 | var_129 | var_130 | var_131 | var_132 | var_133 | var_134 | ... | var_64 | var_65 | var_66 | var_67 | var_68 | var_69 | va |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Count** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | |
| **Percentage** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | |
| **Type** | object | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | float64 | ... | float64 | float64 | float64 | float64 | float64 | float64 | fl |

3 rows × 201 columns

## R Code

```
C:/Users/jerin/Desktop/R work/EDWISOR PROJECT/
> # Missing Value Analysis
> missing_train_data_values= (apply(train_data,2,function(x)sum(is.na(x))))
> sum(missing_train_data_values)
[1] 0
> missing_test_data_values = (apply(test_data,2,function(x)sum(is.na(x))))
> sum(missing_test_data_values)
[1] 0
```

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

## 2.1.3 Basic Features

In [10]:
```
train.describe()
```

Out[10]:

| | target | var_0 | var_1 | var_2 | var_3 | var_4 | var_5 | var_6 | var_7 | var_ |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.0000 |
| mean | 0.100490 | 10.679914 | -1.627622 | 10.715192 | 6.796529 | 11.078333 | -5.065317 | 5.408949 | 16.545850 | 0.2841 |
| std | 0.300653 | 3.040051 | 4.050044 | 2.640894 | 2.043319 | 1.623150 | 7.863267 | 0.866607 | 3.418076 | 3.3326 |
| min | 0.000000 | 0.408400 | -15.043400 | 2.117100 | -0.040200 | 5.074800 | -32.562600 | 2.347300 | 5.349700 | -10.5055 |
| 25% | 0.000000 | 8.453850 | -4.740025 | 8.722475 | 5.254075 | 9.883175 | -11.200350 | 4.767700 | 13.943800 | -2.3178 |
| 50% | 0.000000 | 10.524750 | -1.608050 | 10.580000 | 6.825000 | 11.108250 | -4.833150 | 5.385100 | 16.456800 | 0.3937 |
| 75% | 0.000000 | 12.758200 | 1.358625 | 12.516700 | 8.324100 | 12.261125 | 0.924800 | 6.003000 | 19.102900 | 2.9379 |
| max | 1.000000 | 20.315000 | 10.376800 | 19.353000 | 13.188300 | 16.671400 | 17.251600 | 8.447700 | 27.691800 | 10.1513 |

8 rows × 201 columns

In [11]:
```
test.describe()
```

Out[11]:

| | var_0 | var_1 | var_2 | var_3 | var_4 | var_5 | var_6 | var_7 | var_8 | var_ |
|---|---|---|---|---|---|---|---|---|---|---|
| count | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.000000 | 200000.0000 |
| mean | 10.658737 | -1.624244 | 10.707452 | 6.788214 | 11.076399 | -5.050558 | 5.415164 | 16.529143 | 0.277135 | 7.5694 |
| std | 3.036716 | 4.040509 | 2.633888 | 2.052724 | 1.616456 | 7.869293 | 0.864686 | 3.424482 | 3.333375 | 1.2318 |
| min | 0.188700 | -15.043400 | 2.355200 | -0.022400 | 5.484400 | -27.767000 | 2.216400 | 5.713700 | -9.956000 | 4.2433 |
| 25% | 8.442975 | -4.700125 | 8.735600 | 5.230500 | 9.891075 | -11.201400 | 4.772600 | 13.933900 | -2.303900 | 6.6238 |
| 50% | 10.513800 | -1.590500 | 10.560700 | 6.822350 | 11.099750 | -4.834100 | 5.391600 | 16.422700 | 0.372000 | 7.6320 |
| 75% | 12.739600 | 1.343400 | 12.495025 | 8.327600 | 12.253400 | 0.942575 | 6.005800 | 19.094550 | 2.930025 | 8.5848 |
| max | 22.323400 | 9.385100 | 18.714100 | 13.142000 | 16.037100 | 17.253700 | 8.302500 | 28.292800 | 9.665500 | 11.0036 |

8 rows × 200 columns

We can propose the following observations:

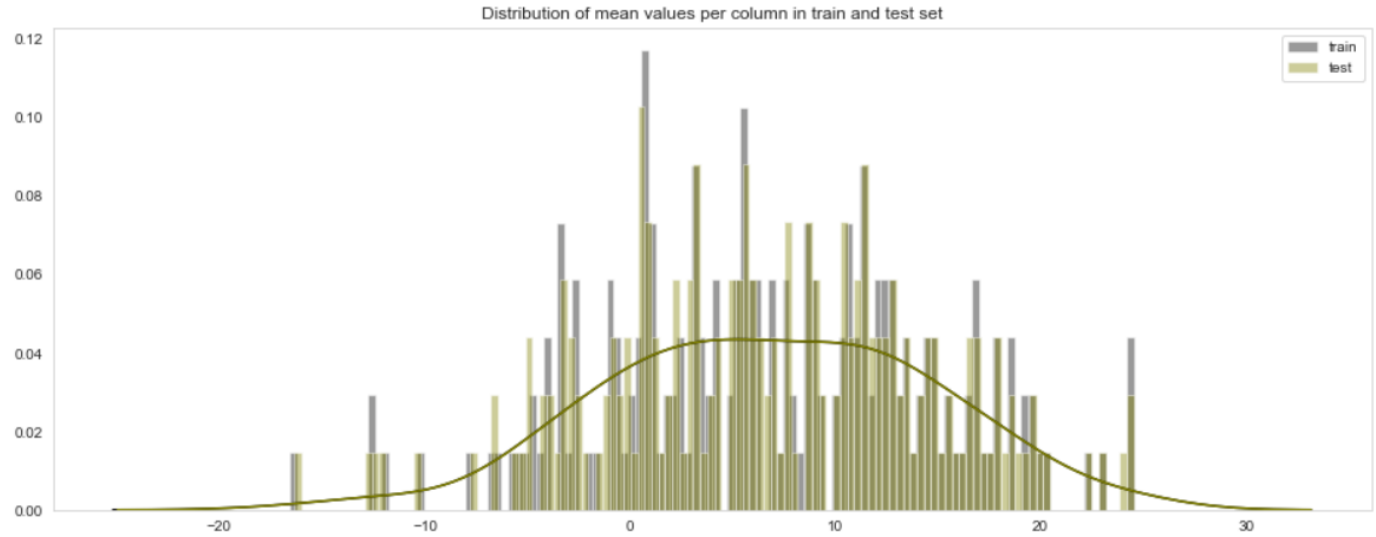
- standard deviation is relatively large for both train and test variable data
- min, max, mean, std values for train and test data looks quite close
- mean values are distributed over a large range.
- The number of values in train and test set is the same.

## 2.1.4 Data Visualizations

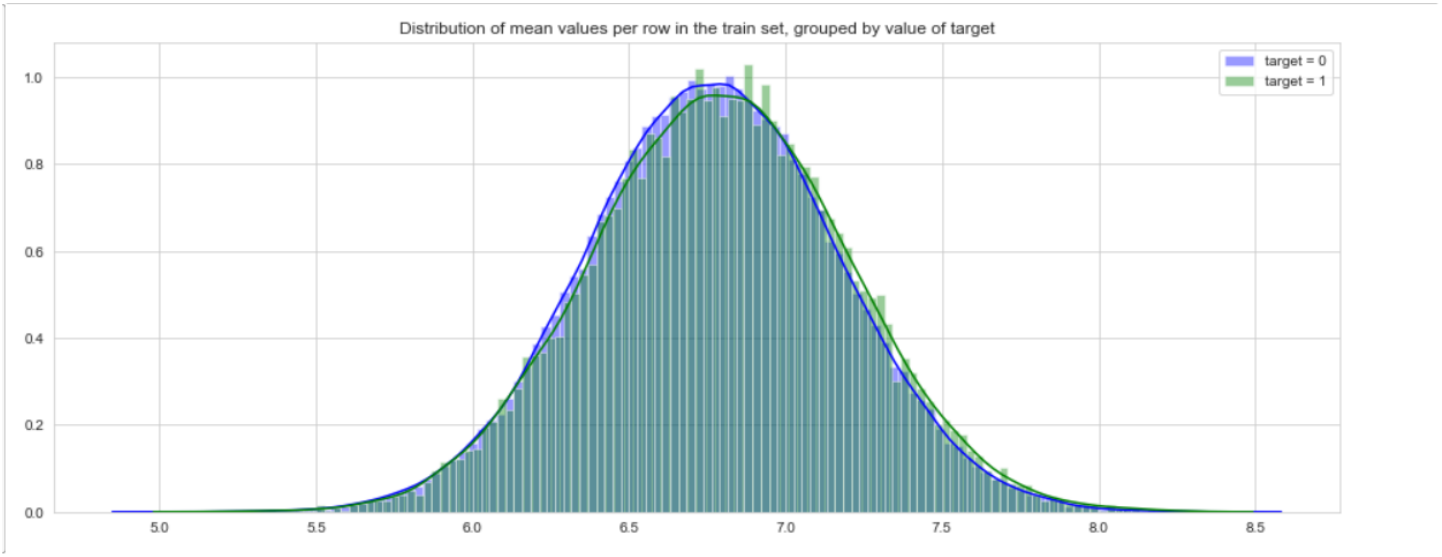
### Distribution of mean values in both train and test dataset: -

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

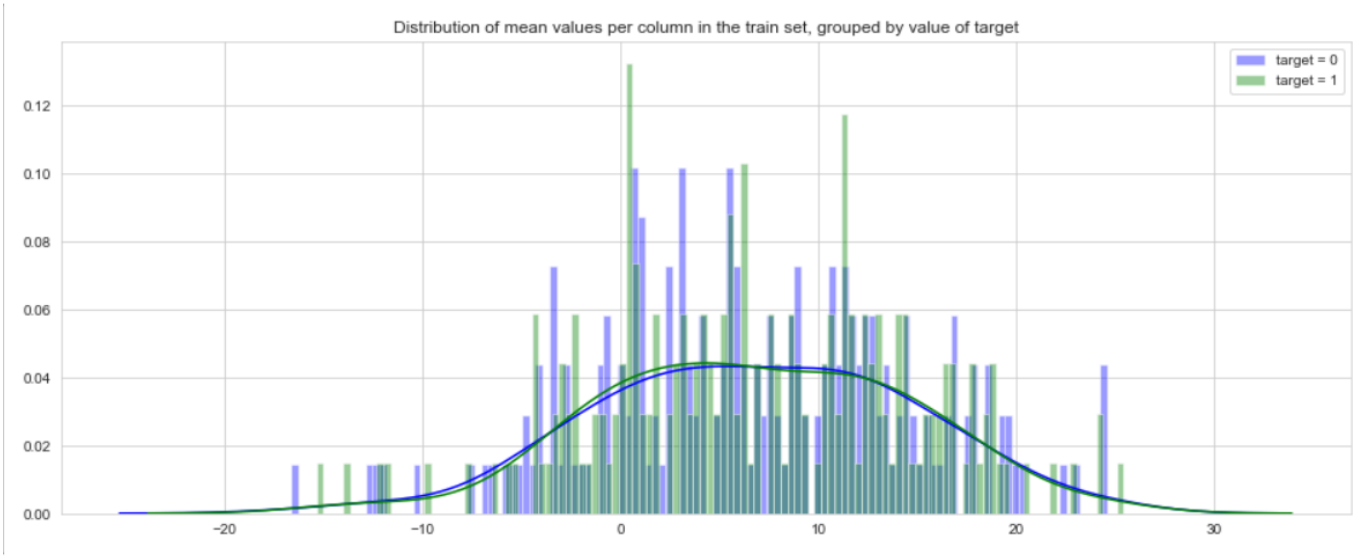

Let us look distribution of mean values per column in train and test dataset



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



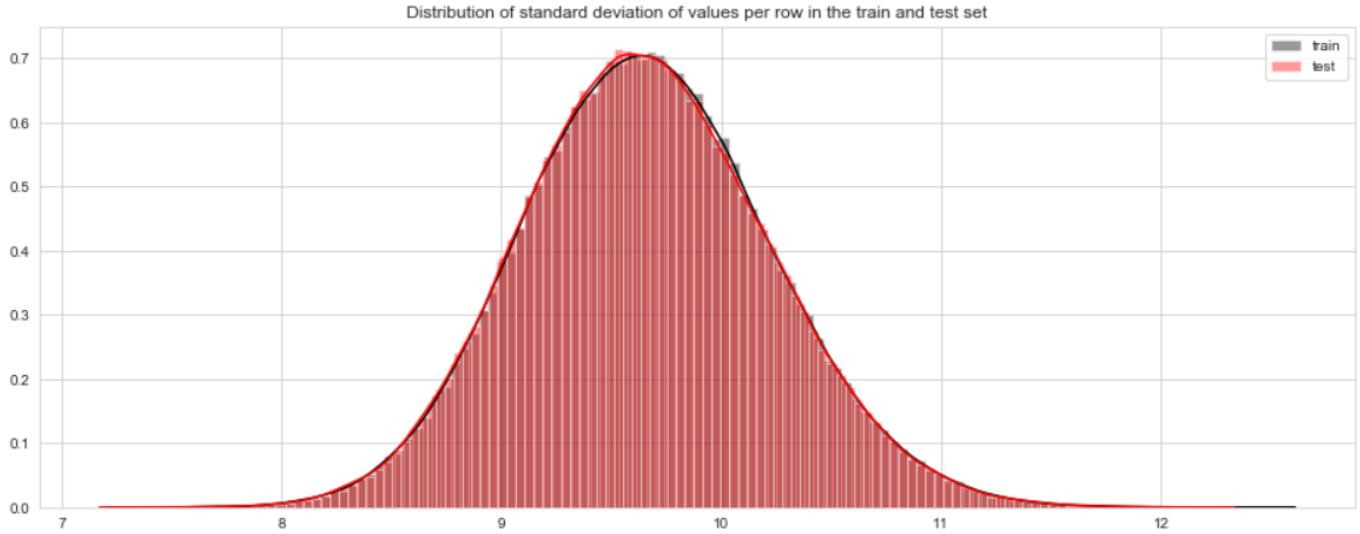

Let us look distribution of mean values per column in train dataset, grouped by value of target



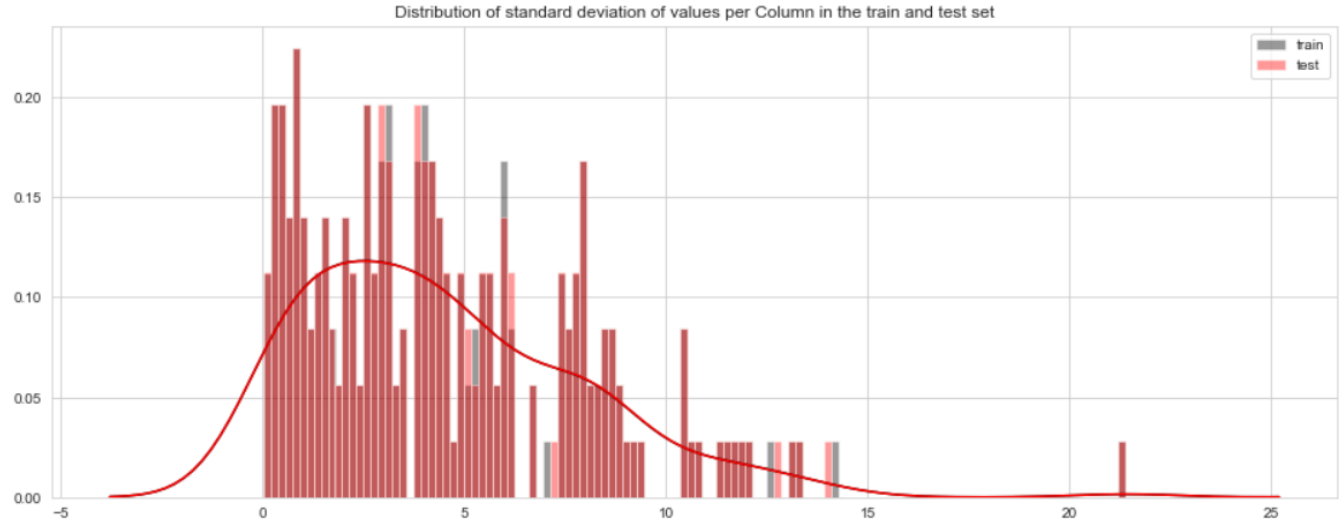

- Mean values of train and test set looks quite close
- Mean values are distributed over a large range.

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

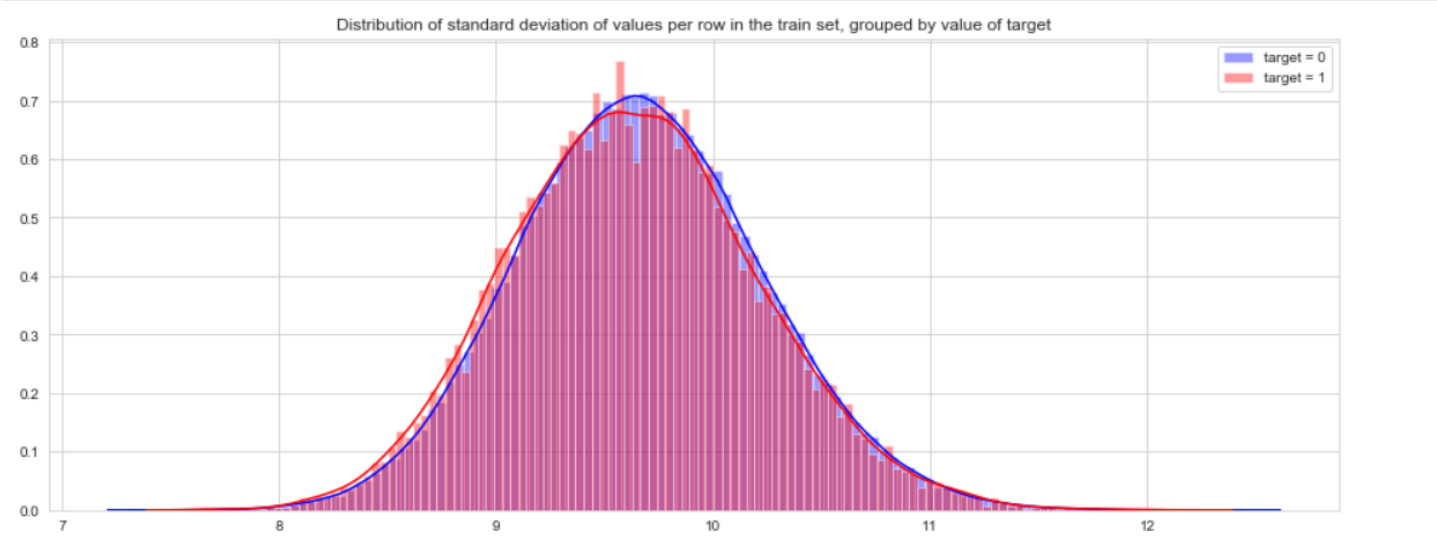
Let us look distribution of standard deviation per row in train and test dataset



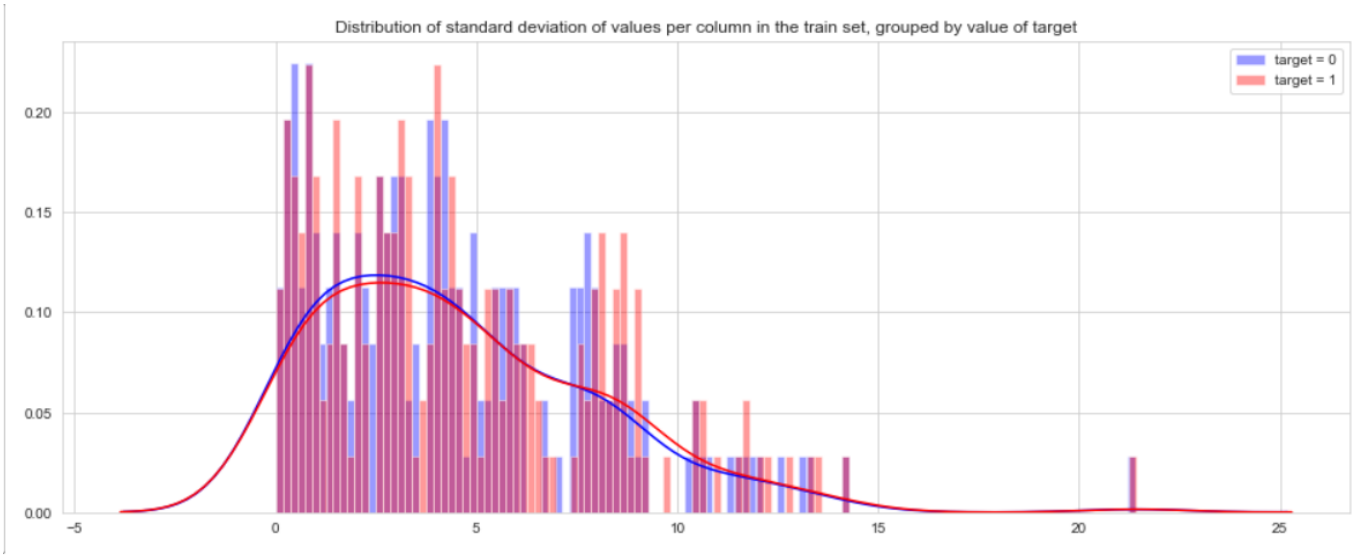

Let us look distribution of standard deviation per column in train and test dataset



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



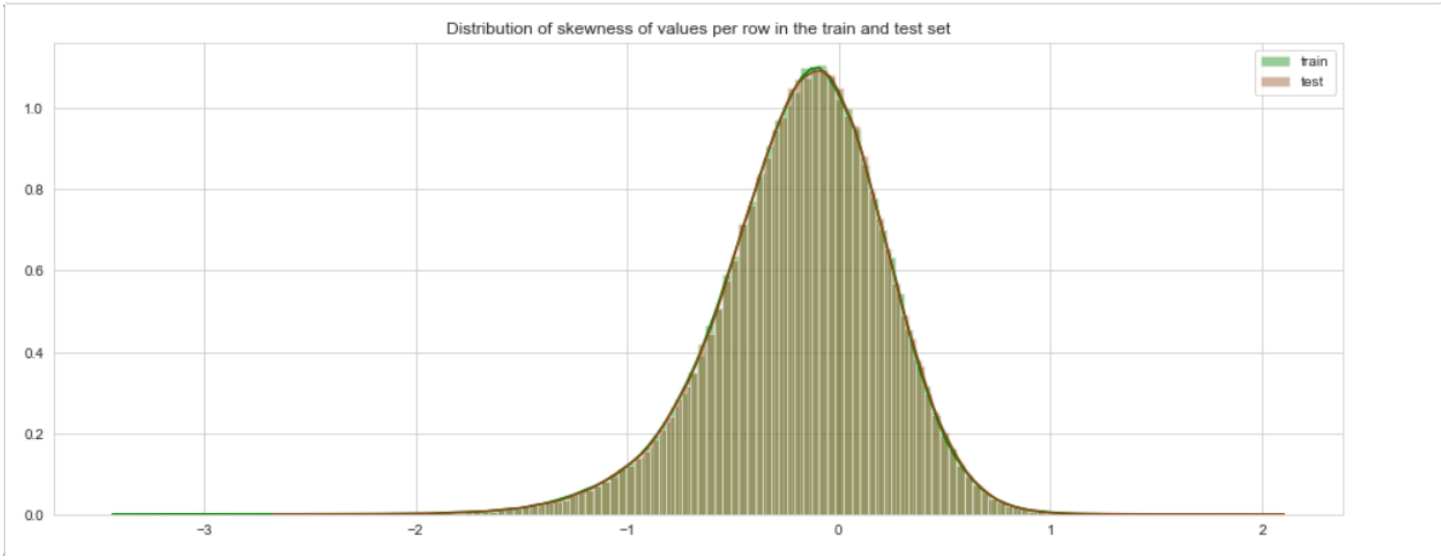

Let us look distribution of standard deviation(std) values per column in train dataset, grouped by value of target



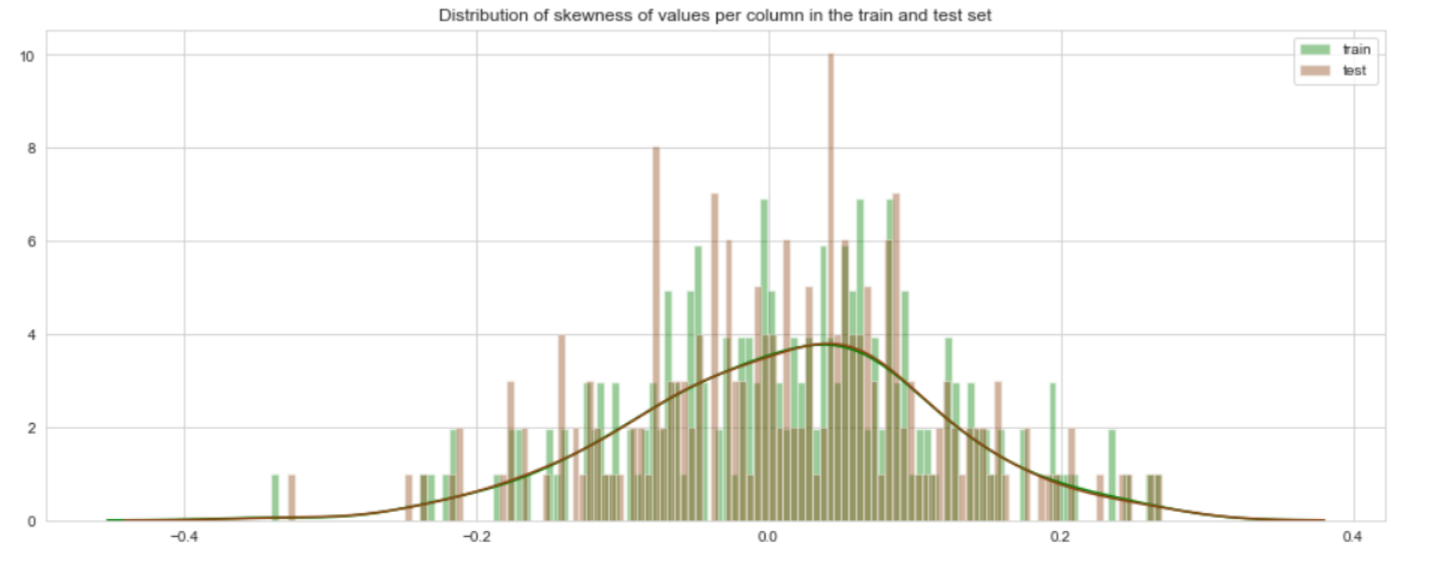

- Standard deviation is relatively large for both train and test variable data.
- Standard deviation values for train and test data looks similar

**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



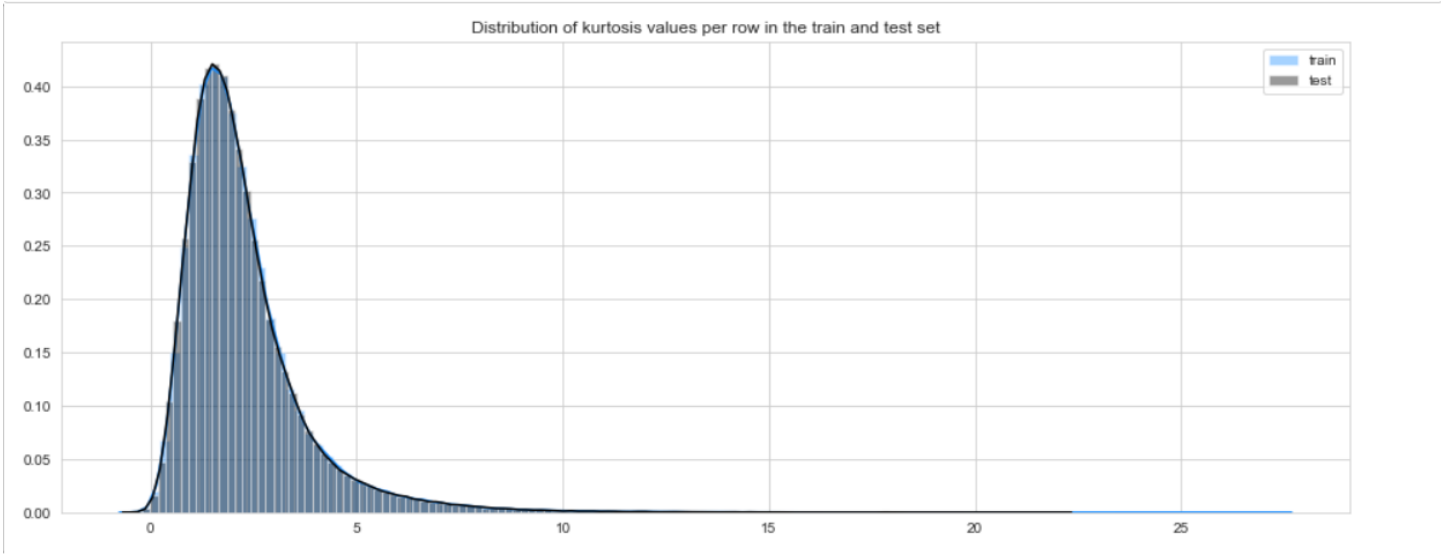

Let us look distribution of skewness of values per column in train and test dataset



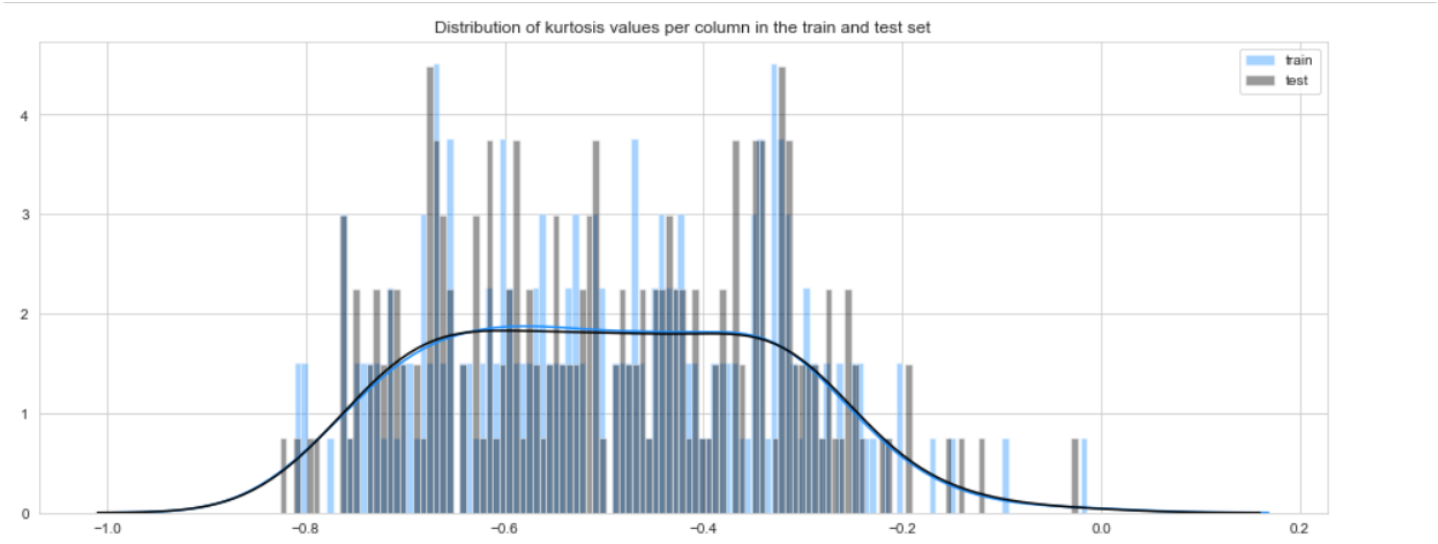

- We found the distribution is left skewed

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

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



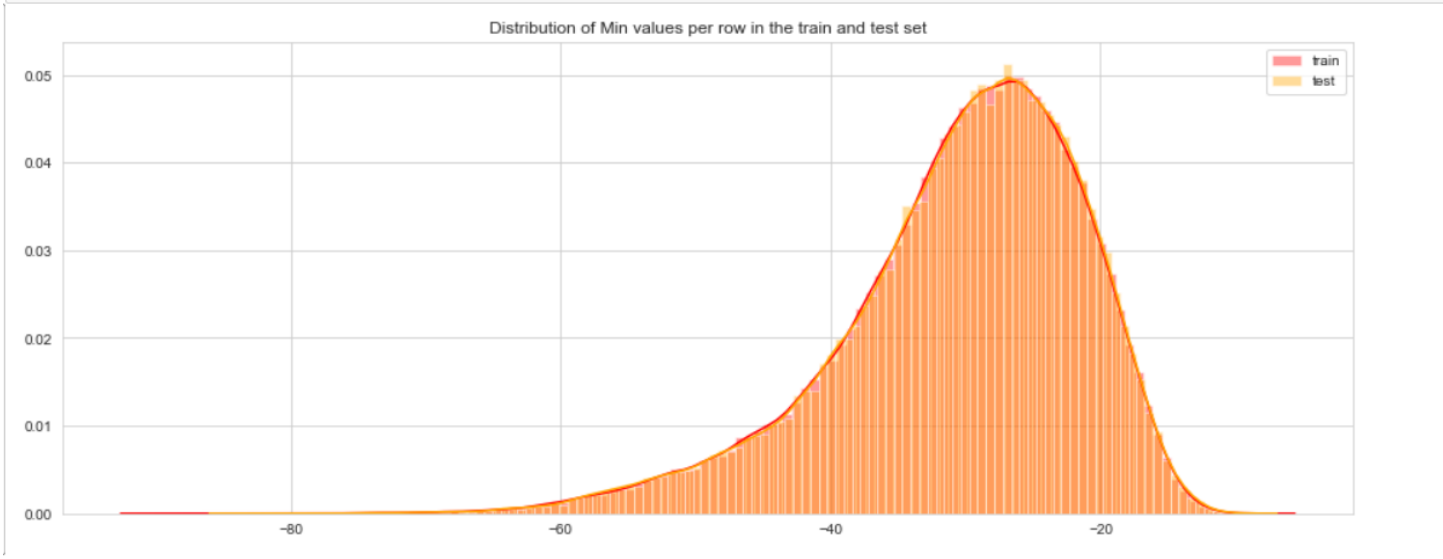

Let us look distribution of kurtosis per column in train and test dataset



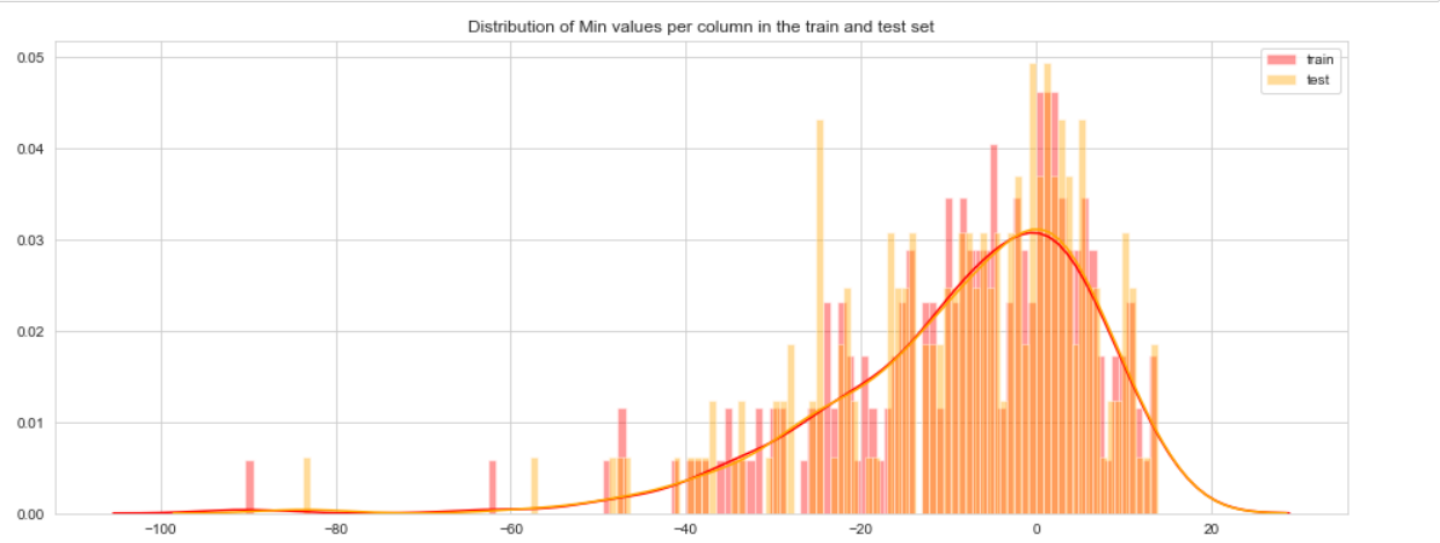

- We found the distribution to be Leptokurtic which means heavy tails on either side indicating large outliers

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

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

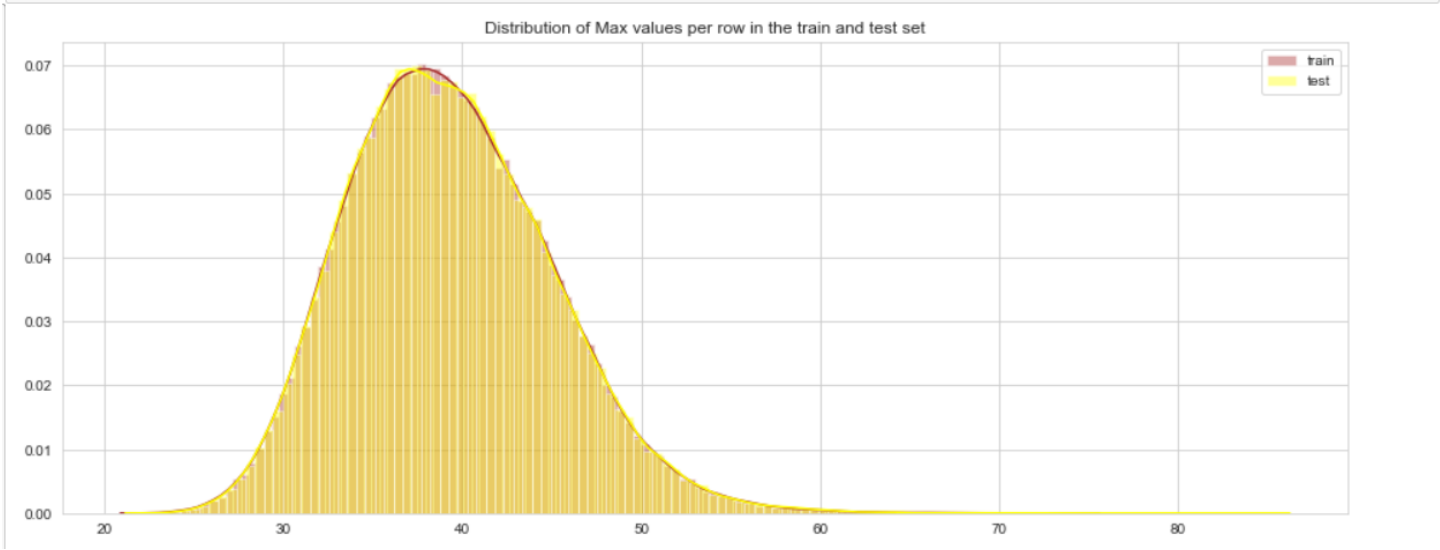



Let us look distribution of Min values per column in train and test dataset

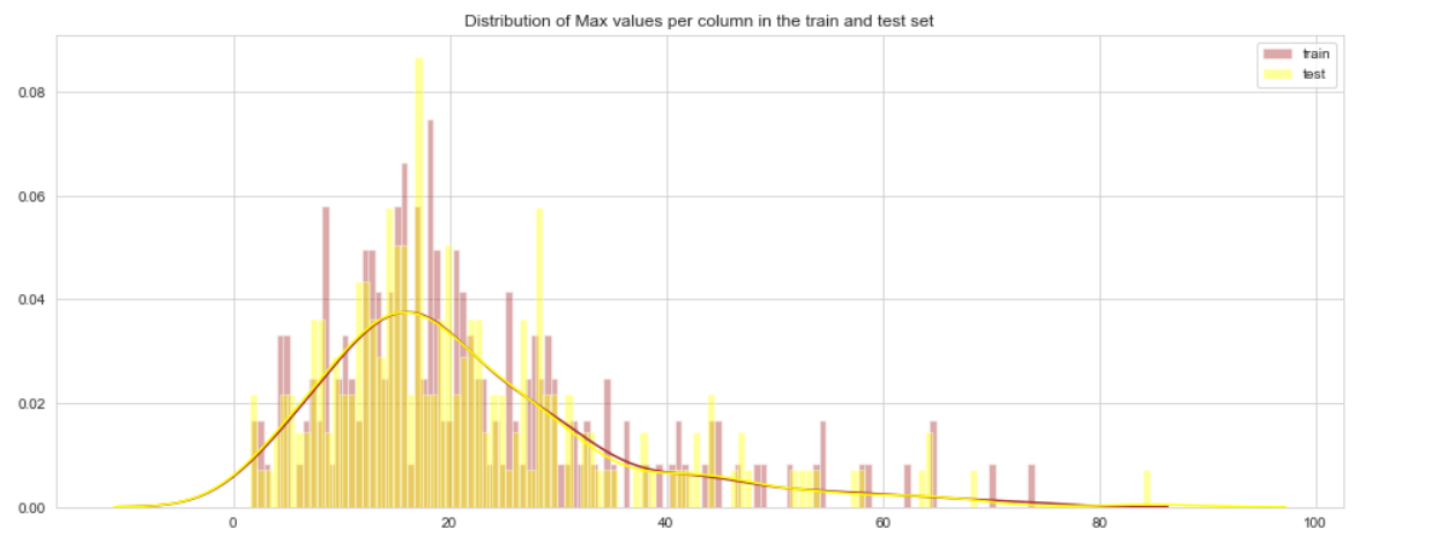


**Distribution of Max Values in both train and test dataset: -**

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

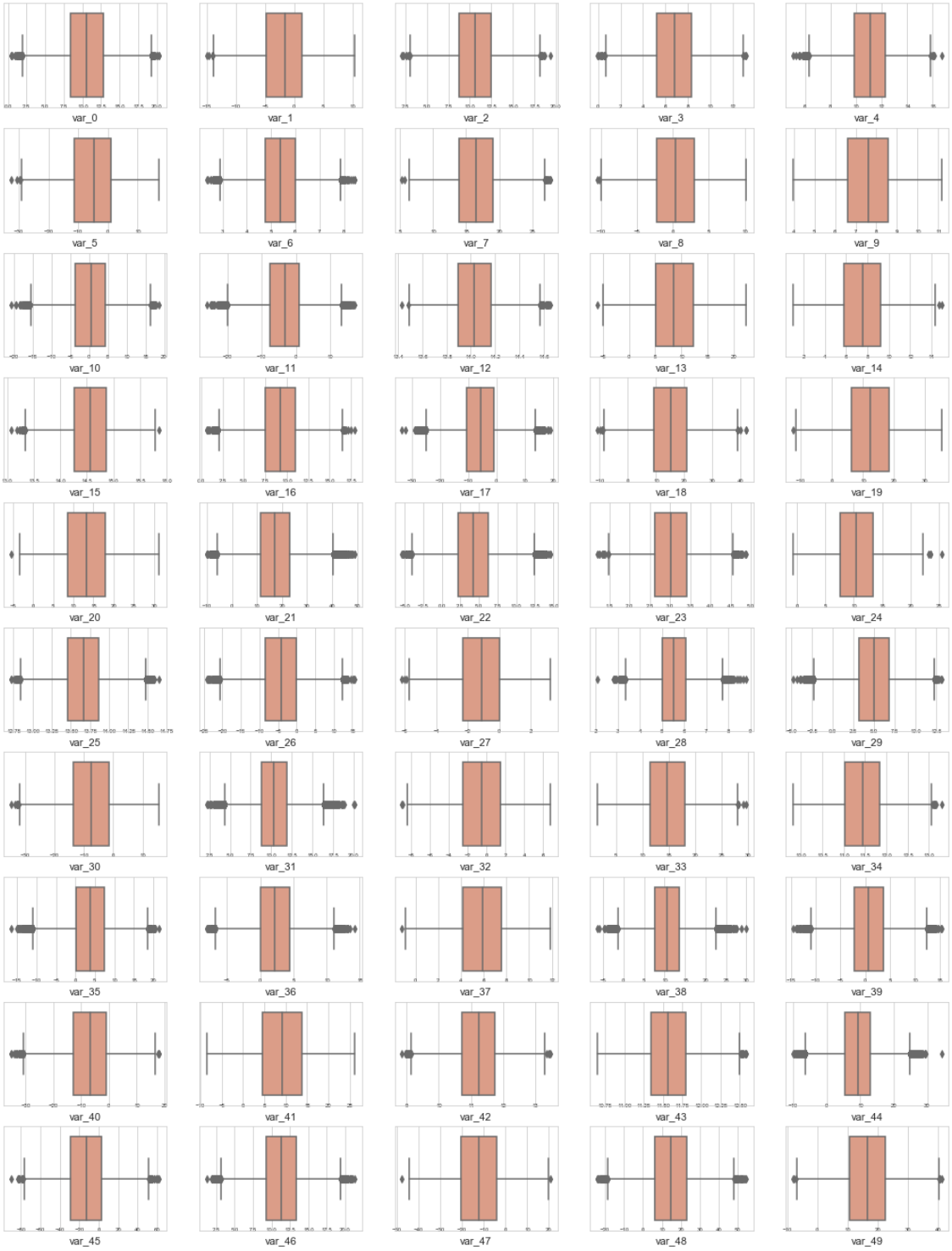



**Fig: Box plots from var_0 to var_49**

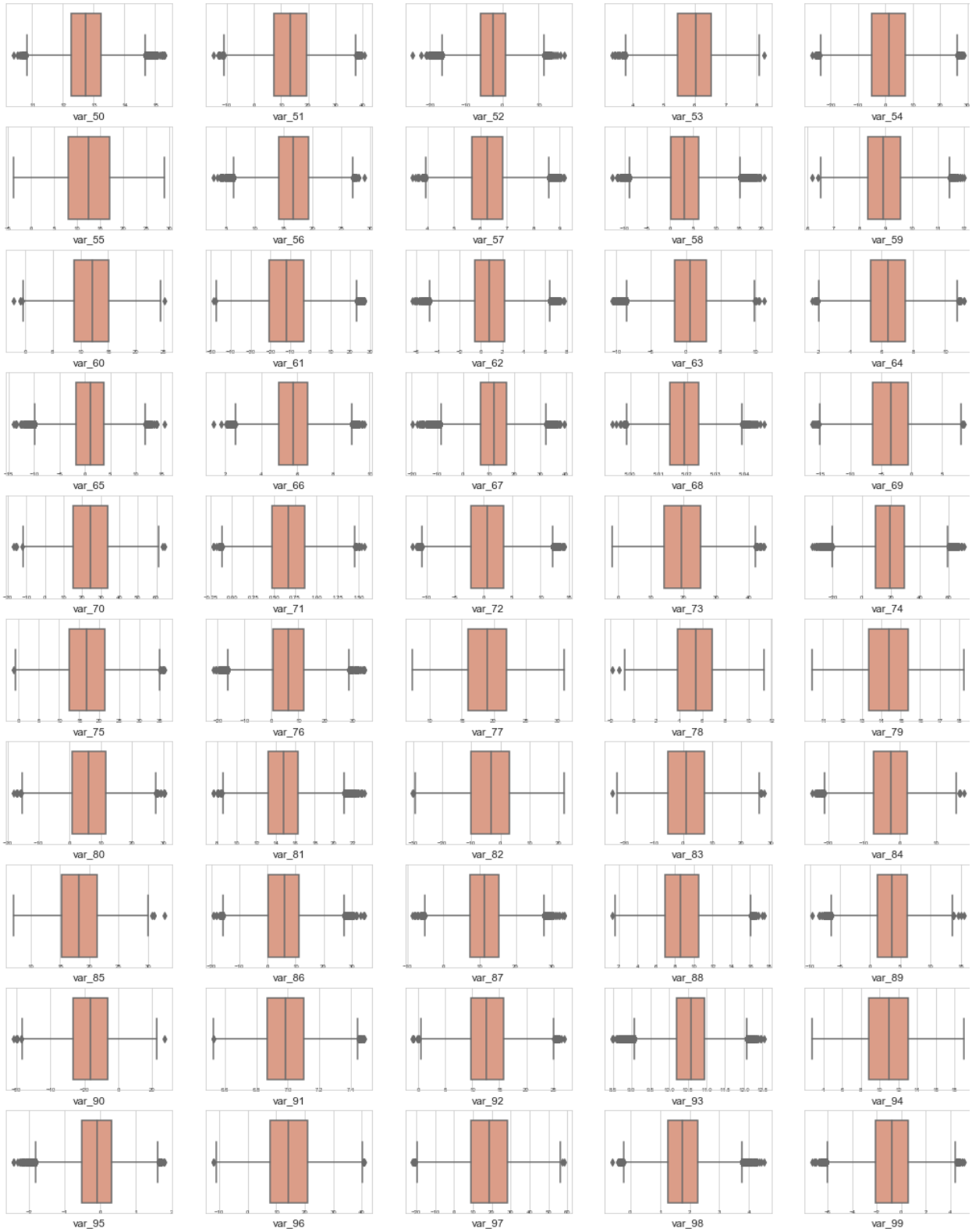



**Fig: Box plots from var_50 to var_99**

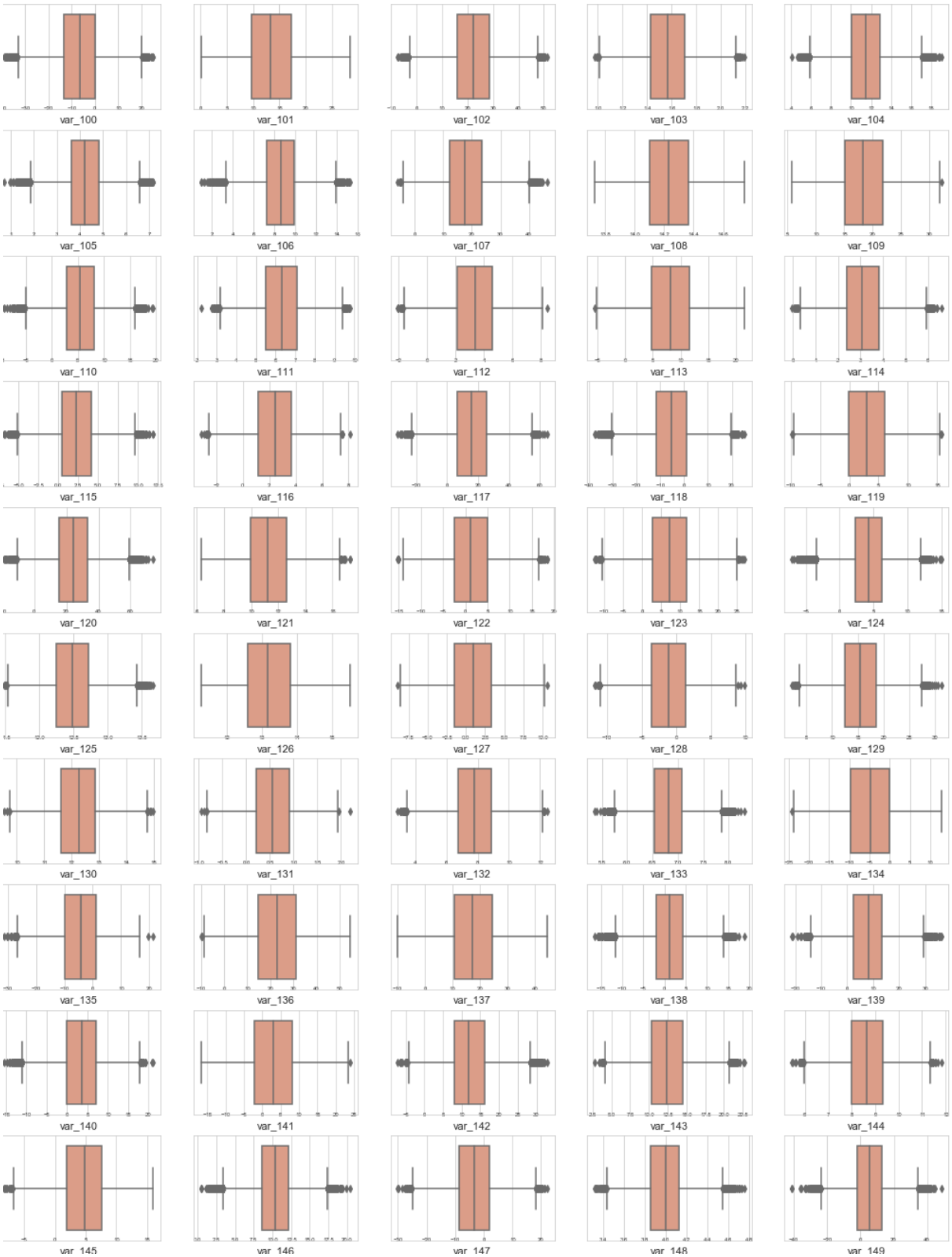



**Fig: Box plots from var_100 to var_149**

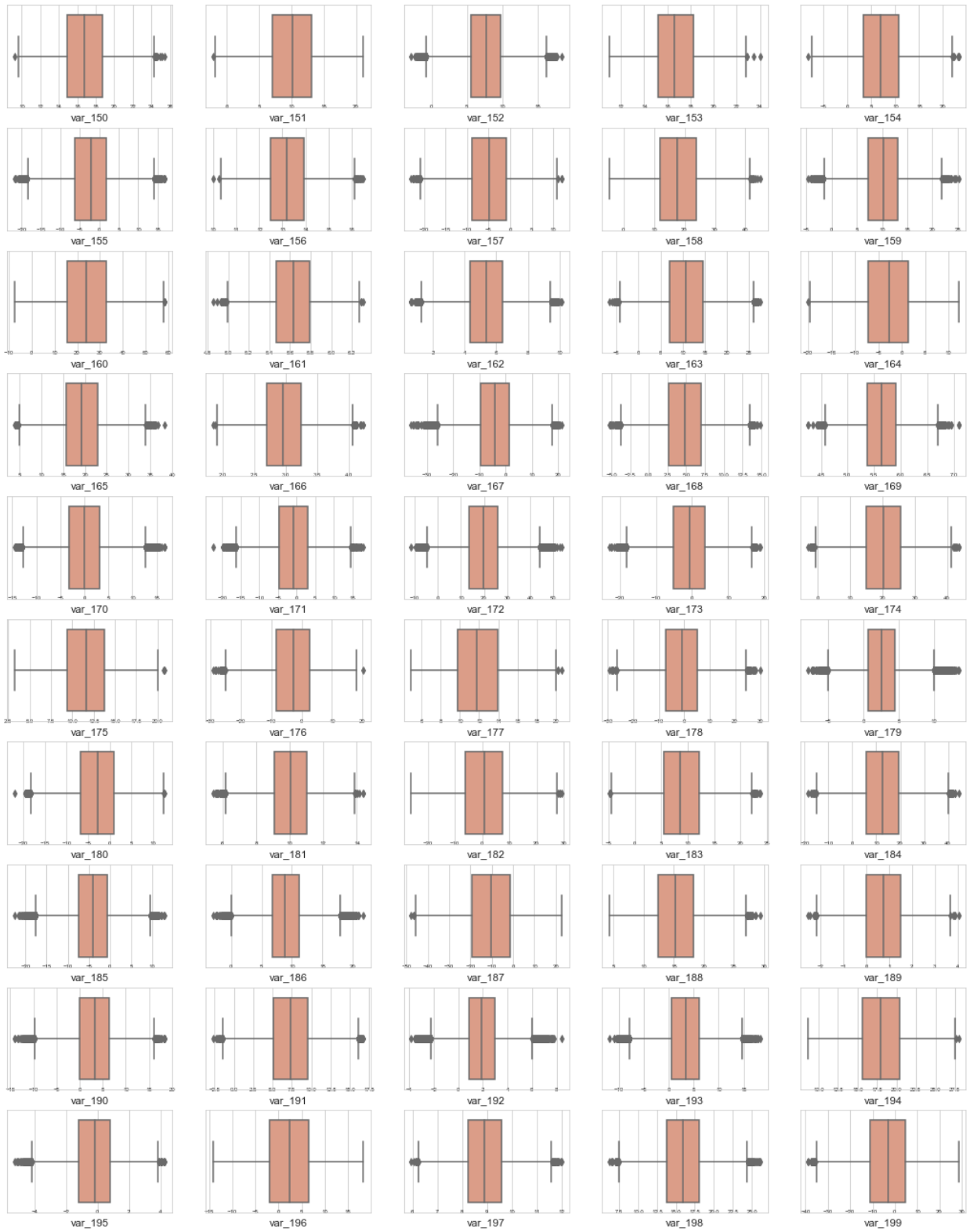



**Fig: Box plots from var_150 to var_199**

```
# #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.shape: (157999, 202)
train_out.shape: (42001, 202)
```

---

```
train_in['target'].value_counts()
```

```
0    157999
Name: target, dtype: int64
```

```
# 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()
```

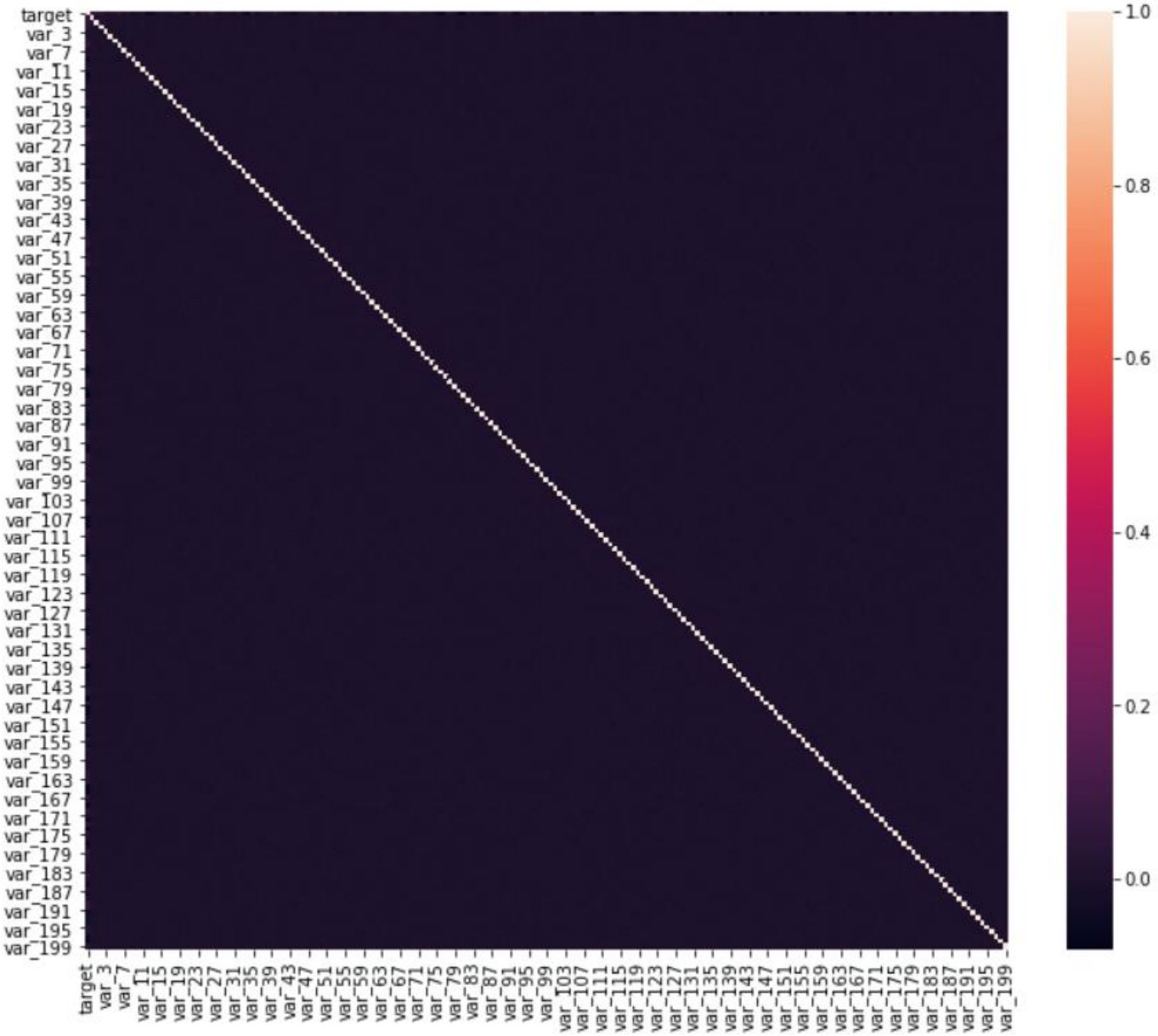
```
0    21903
1    20098
Name: target, dtype: int64
```

```
train['target'].value_counts()
```

```
0    179902
1     20098
Name: target, dtype: int64
```

- 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.

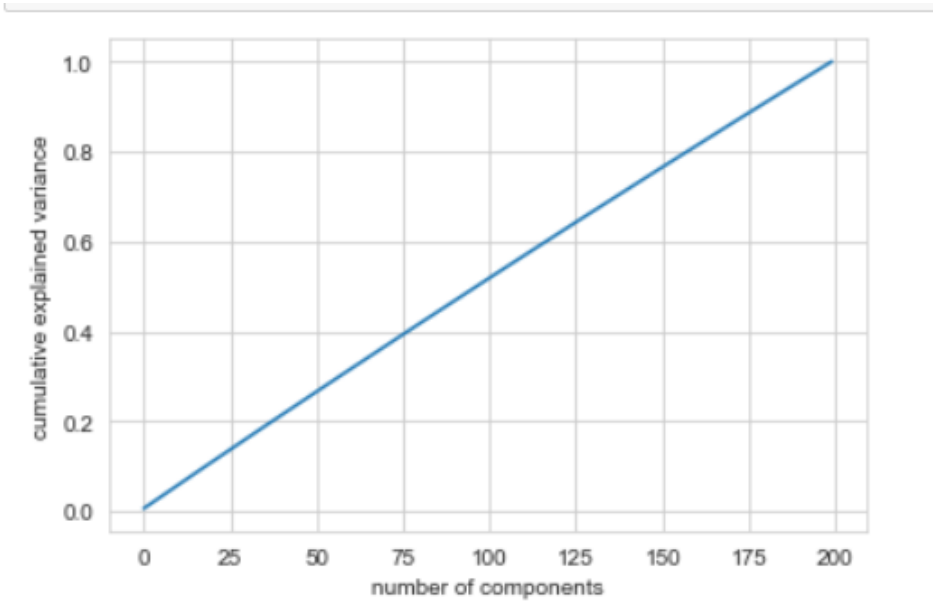
From correlation plot, we can observe that correlation between both train and test attributes are very small. It means that all both train and test attributes are independent to each other and are not correlated to each other.

## 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.

| | | Predicted class: P | Predicted class: N |
|---|---|---|---|
| Actual Class | P | True Positives (TP) | False Negatives (FN) |
| | N | False Positives (FP) | True Negatives (TN) |

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 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.

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP} + \text{FN})}$$

**ROC_AUC Scores:**

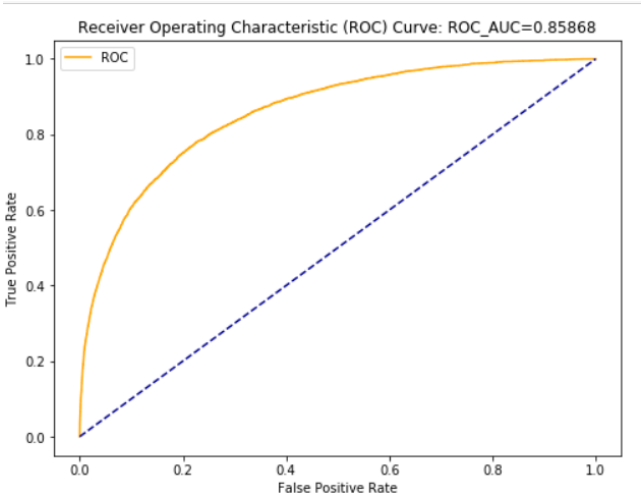
It can be more flexible to predict probabilities of an observation belonging to each class in a classification problem rather than predicting classes directly.
The reason for this is to provide our model the capability to choose and even calibrate the threshold for how to interpret the predicted probabilities.

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.

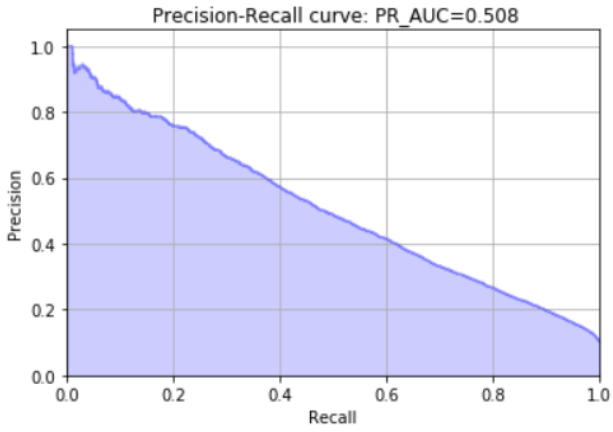
A typical ROC curve has False Positive Rate (FPR) on the X-axis and True Positive Rate (TPR) on the Y-axis.

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

```
## 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))
```

## 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]))
```

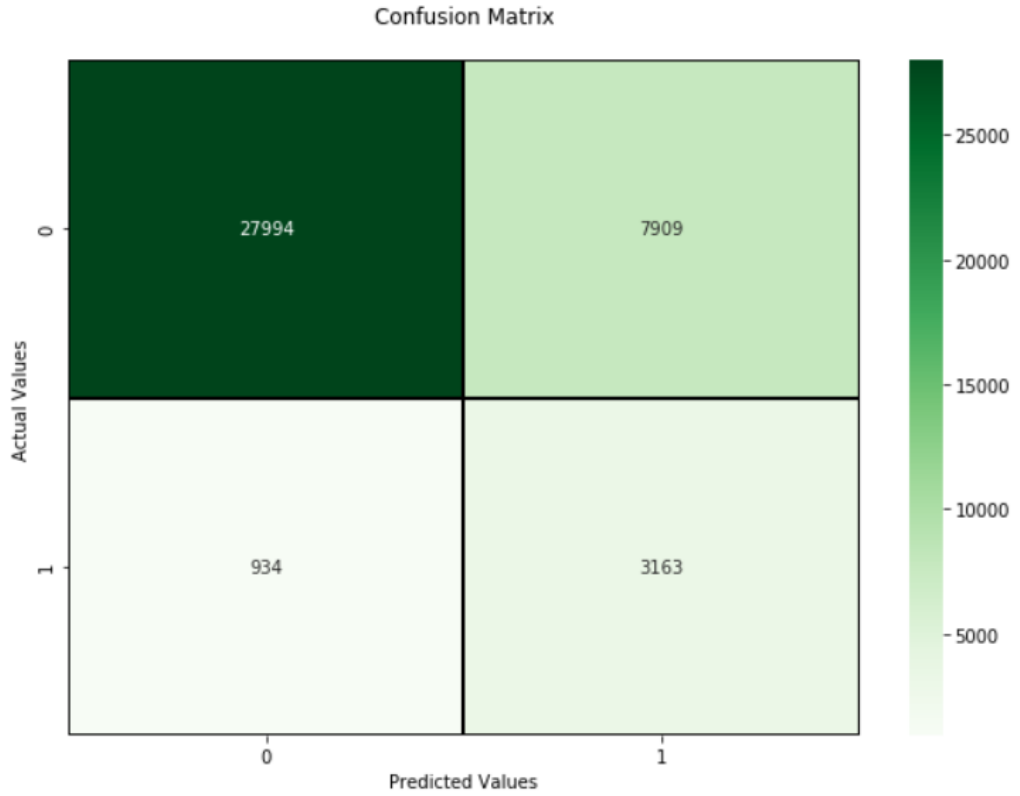
```
  print("the Precision for this model is :",cnf_matrix[1,1]/(cnf_matrix[1,1]+cnf_matrix[0,1]))
    fig= plt.figure(figsize=(10,7))
    print("TP",cnf_matrix[1,1]) # no of true transactions which are predicted as true
    print("TN",cnf_matrix[0,0]) # no of false transaction which are predicted as 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-

```
model(LogisticRegression(class_weight='balanced',max_iter=10000),X_train,y_train, X_test,y_test)
```

```
The Accuracy of this model :  77.8925
the Recall for this model is : 0.7720283134000488
the Precision for this model is : 0.2856755780346821
TP 3163
TN 27994
FP 7909
FN 934
```



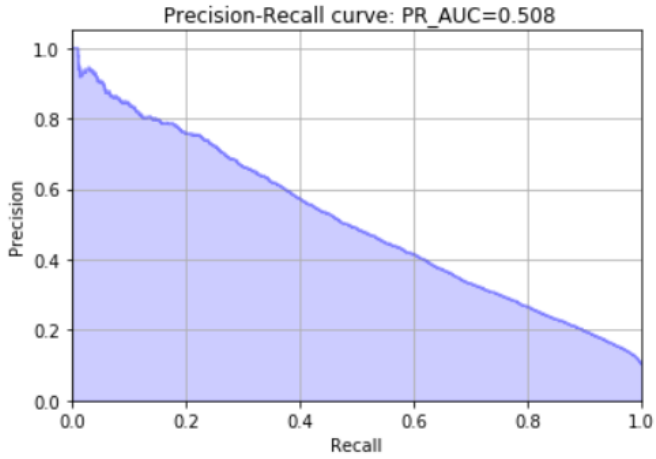

```
---------------Classification Report----------------

              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()
```

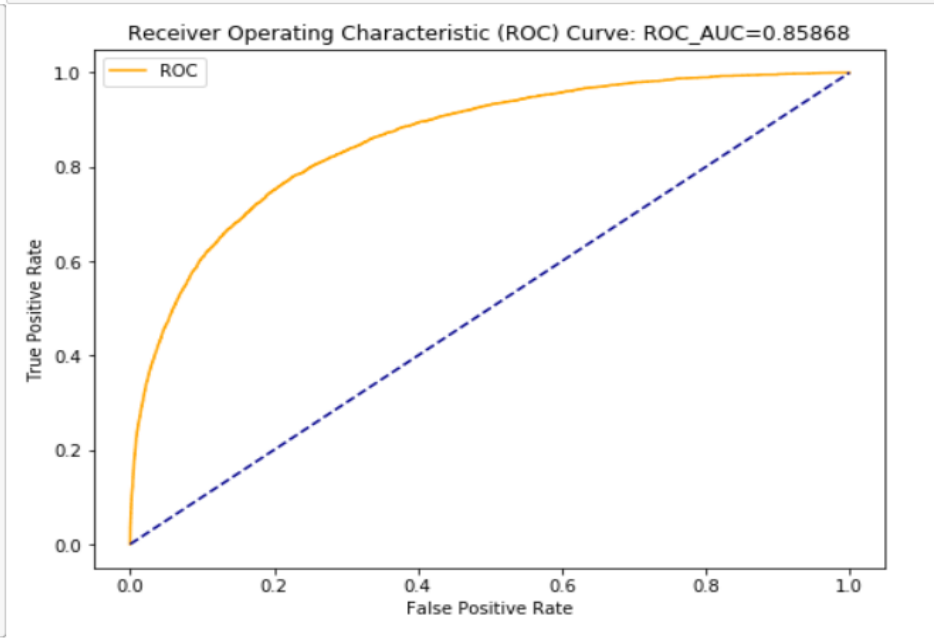
```
plot_precision_recall(y_test, y_pred)
```




```
# 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()
```

```
fpr, tpr, thresholds = roc_curve(y_test,y_pred)
plot_roc_curve(fpr, tpr)
```




**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)))
}
```

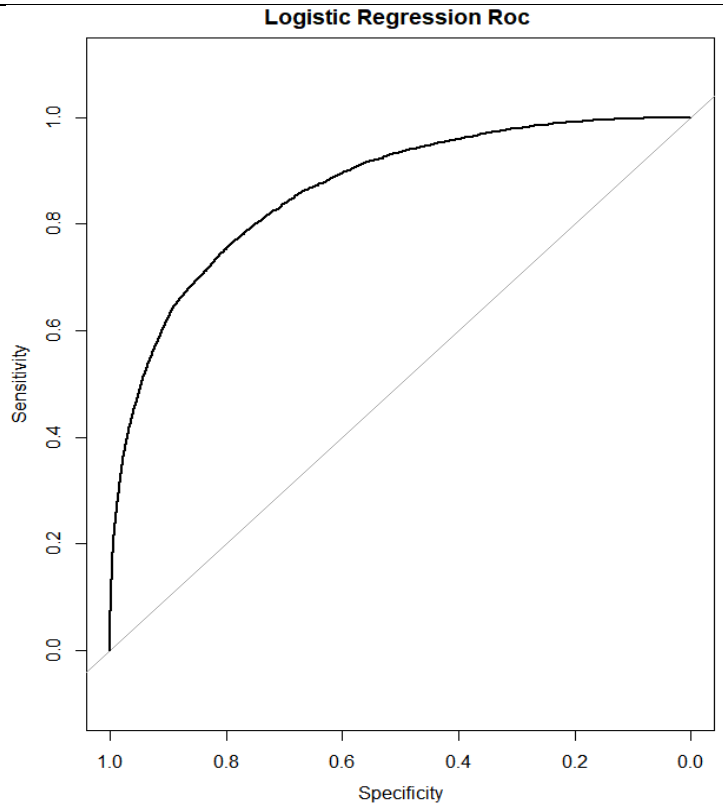
```
######################## Logistic Regression Model ############################
logit_model =glm(target~. ,data =train ,family='binomial')
# model summary
summary(logit_model)
#get model predicted  probality
y_prob =predict(logit_model , test[,-1] ,type = 'response' )
# convert   probality to class 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-

```
> getmodel_accuracy(conf_matrix)
[1] "accuracy 0.92"
[1] "precision 0.7"
[1] "recall 0.01"
```

```
> confusionMatrix(conf_matrix)
Confusion Matrix and Statistics

   y_pred
      0     1
  0 35517   494
  1  2852  1137

               Accuracy : 0.9164
                 95% CI : (0.9136, 0.919)
    No Information Rate : 0.9592
    P-Value [Acc > NIR] : 1

                  Kappa : 0.368

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9257
            Specificity : 0.6971
         Pos Pred Value : 0.9863
         Neg Pred Value : 0.2850
             Prevalence : 0.9592
         Detection Rate : 0.8879
   Detection Prevalence : 0.9003
      Balanced Accuracy : 0.8114

       'Positive' Class : 0
```

```
# 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**

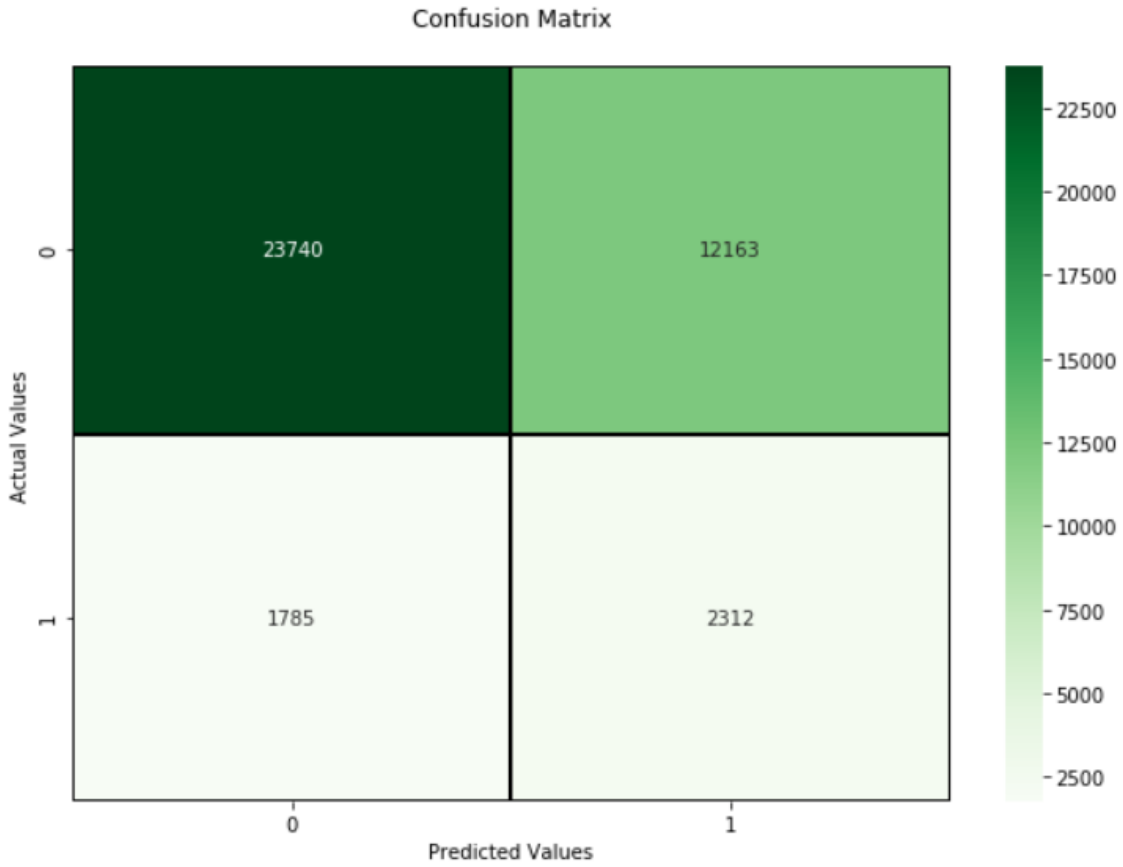
```
tree_clf = DecisionTreeClassifier(class_weight='balanced', random_state = 42,
                                  max_features = 0.7, min_samples_leaf = 80)
```

```
tree_clf
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight='balanced', criterion='gini',
                       max_depth=None, max_features=0.7, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=80, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

```
model(tree_clf,X_train, y_train,X_test, y_test)
```

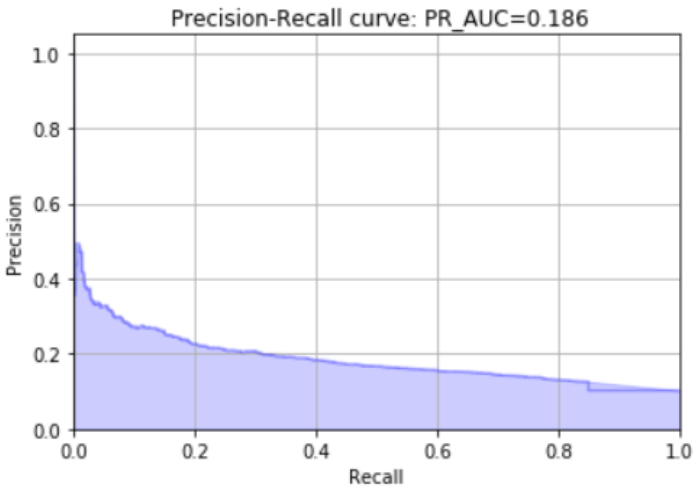
```
The Accuracy of this model :  65.13
the Recall for this model is : 0.5643153526970954
the Precision for this model is : 0.15972366148531952
TP 2312
TN 23740
FP 12163
FN 1785
```



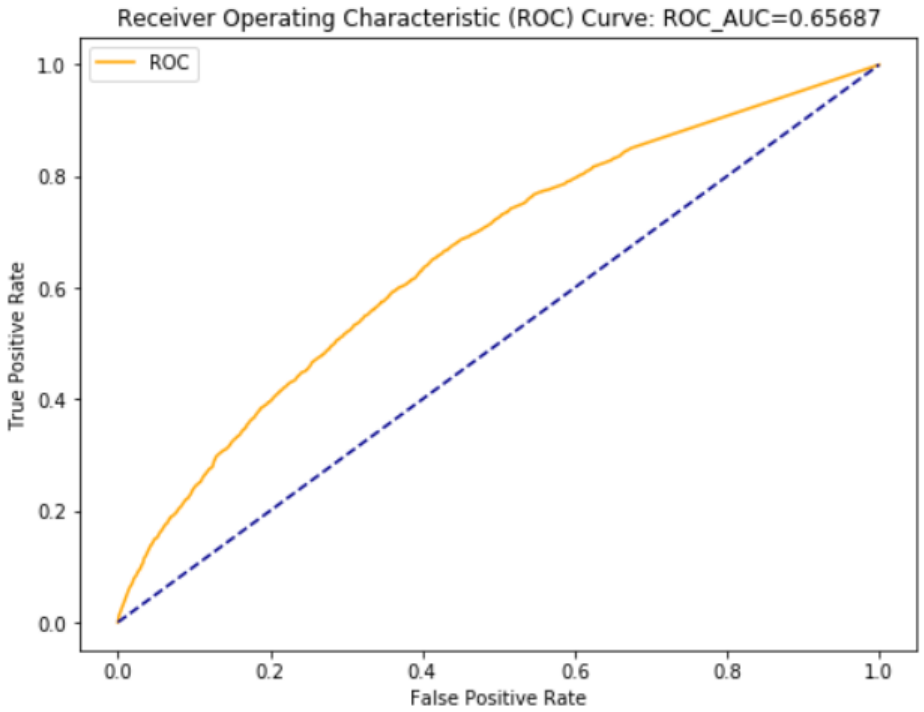

```
---------------Classification Report----------------

              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

```
In [21]: plot_precision_recall(y_test, y_pred)
```




```
In [23]: fpr, tpr, thresholds = roc_curve(y_test, y_pred)
         plot_roc_curve(fpr, tpr)
```



**R Code**

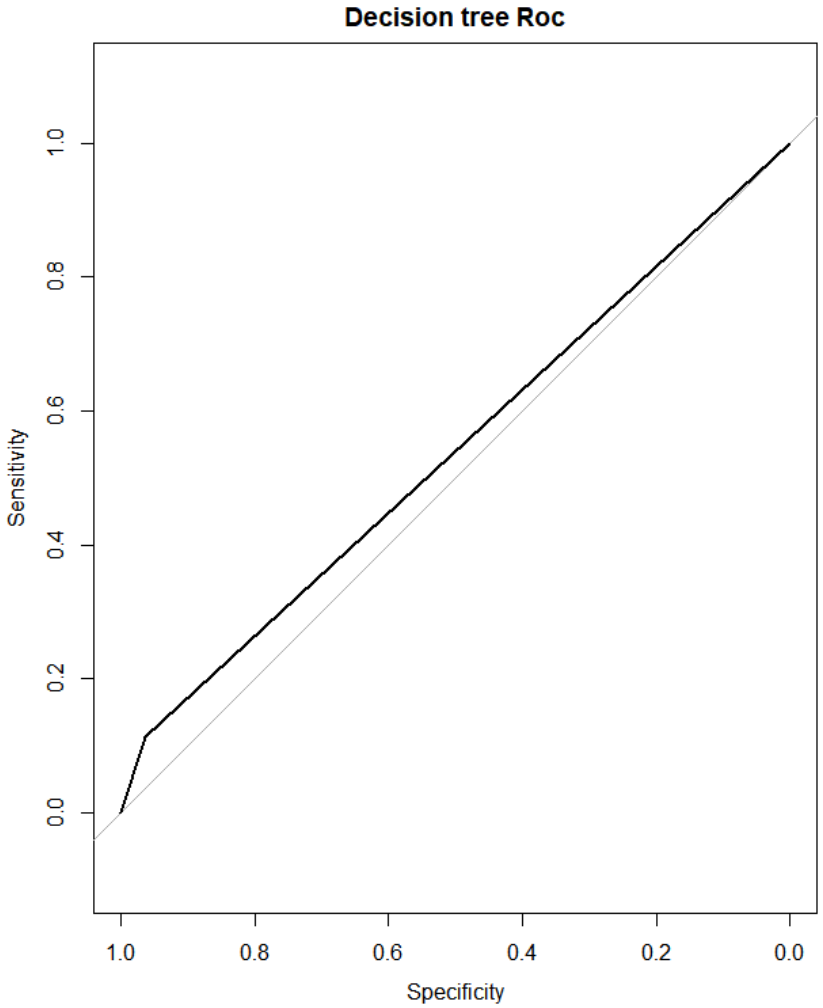
```
############### Decision Tree ###############

#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-

```
> getmodel_accuracy(ConfMatrix_C50)
[1] "accuracy 0.88"
[1] "precision 0.26"
[1] "recall 0.04"
```

```
> confusionMatrix(ConfMatrix_C50)
Confusion Matrix and Statistics

   C50_Predictions
        0     1
  0 34645  1335
  1  3561   458

               Accuracy : 0.8776
                 95% CI : (0.8743, 0.8808)
    No Information Rate : 0.9552
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1019

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9068
            Specificity : 0.2554
         Pos Pred Value : 0.9629
         Neg Pred Value : 0.1140
             Prevalence : 0.9552
         Detection Rate : 0.8661
   Detection Prevalence : 0.8995
      Balanced Accuracy : 0.5811

       'Positive' Class : 0
```

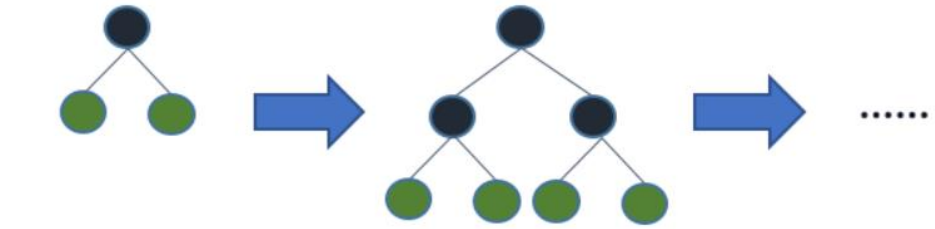
```
#get Auc score
C50_Predictions<-as.numeric(C50_Predictions)
roc=roc(test[,1], C50_Predictions )
print(roc)
# plot roc_auc curve
plot(roc ,main="Decision tree Roc")
```



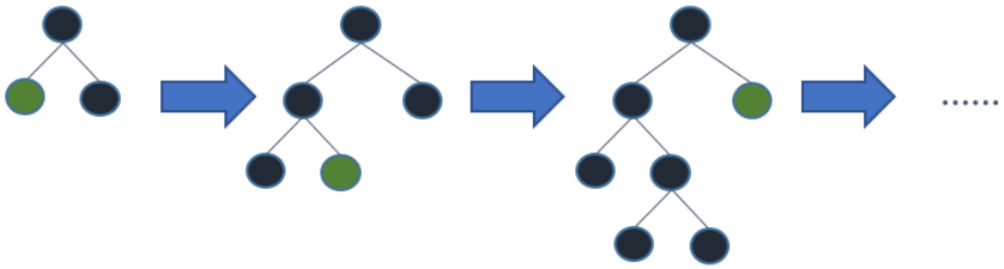

Area under the curve: 0.5384

### 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.



Level-wise tree growth



Leaf-wise tree growth

**Python Code**

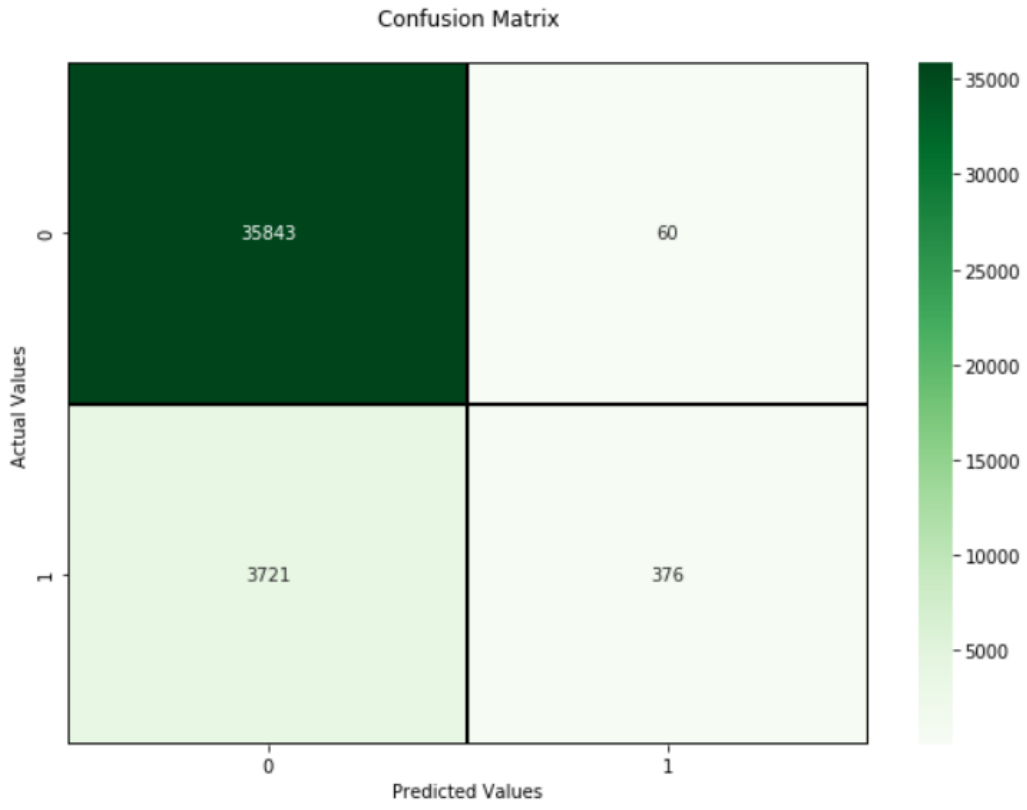
In [24]:
```
lgb_clf= lgb.LGBMClassifier()
```

In [25]:
```
lgb_clf
```

Out[25]:
```
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
               importance_type='split', learning_rate=0.1, max_depth=-1,
               min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
               n_estimators=100, n_jobs=-1, num_leaves=31, objective=None,
               random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=True,
               subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

In [26]:
```
model(lgb_clf,X_train, y_train,X_test, y_test)
```

```
The Accuracy of this model :  90.5475
the Recall for this model is : 0.09177446912374908
the Precision for this model is : 0.8623853211009175
TP 376
TN 35843
FP 60
FN 3721
```

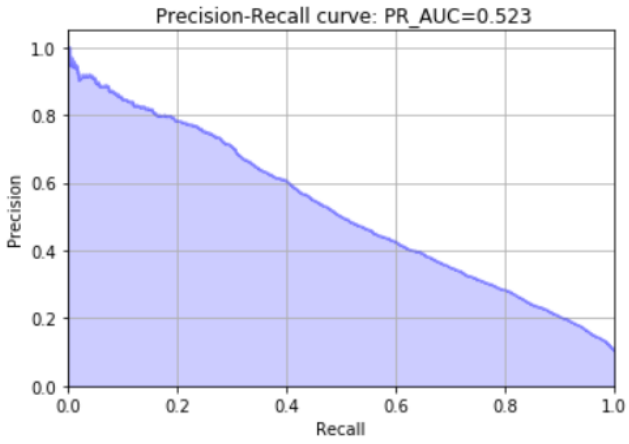



----------Classification Report----------

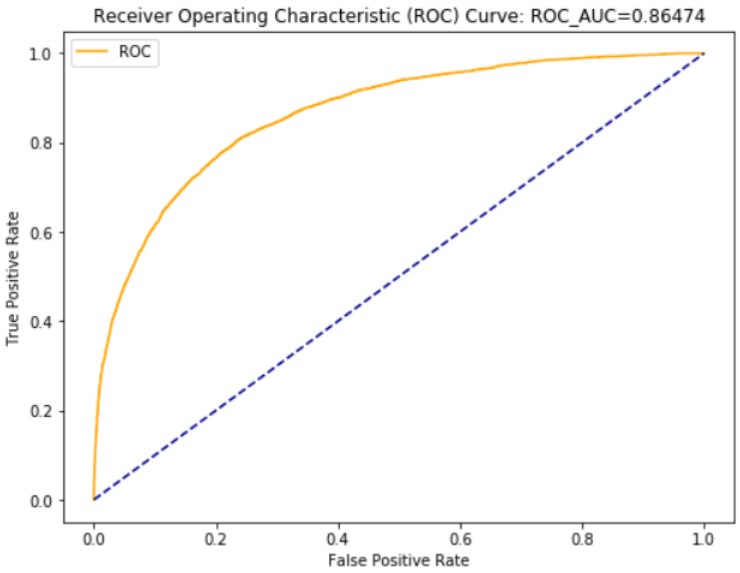
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.91 | 1.00 | 0.95 | 35903 |
| 1 | 0.86 | 0.09 | 0.17 | 4097 |
| accuracy | | | 0.91 | 40000 |
| macro avg | 0.88 | 0.55 | 0.56 | 40000 |
| weighted avg | 0.90 | 0.91 | 0.87 | 40000 |

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

```
In [29]: plot_precision_recall(y_test, y_pred)
```




```
In [30]: fpr, tpr, thresholds = roc_curve(y_test, y_pred)
         plot_roc_curve(fpr, tpr)
```



**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)
```

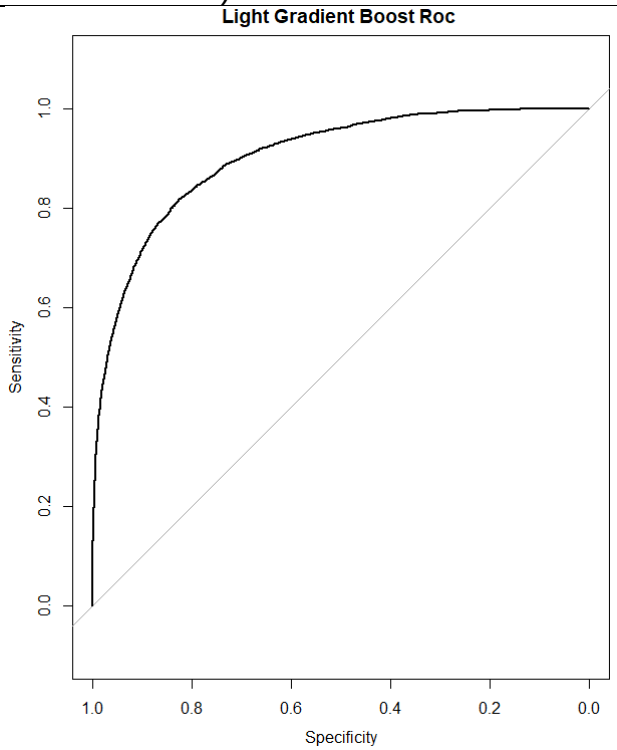
```
> lgbm.model <- lgb.train(params = lgb.grid, data = lgb.train, nrounds =10000,eval_freq =1000,
+                         valids=list(val1=lgb.train,val2=lgb.valid),early_stopping_rounds = 5000)
[LightGBM] [Warning] verbosity is set=-1, verbose=1 will be ignored. Current value: verbosity=-1
[1] "[1]:  val1's auc:0.575252  val2's auc:0.571031"
[1] "[1001]:  val1's auc:0.915356  val2's auc:0.889068"
[1] "[2001]:  val1's auc:0.924239  val2's auc:0.897011"
[1] "[3001]:  val1's auc:0.92693  val2's auc:0.899888"
[1] "[4001]:  val1's auc:0.92803  val2's auc:0.900916"
[1] "[5001]:  val1's auc:0.928593  val2's auc:0.901421"
[1] "[6001]:  val1's auc:0.929005  val2's auc:0.901876"
[1] "[7001]:  val1's auc:0.929364  val2's auc:0.902021"
[1] "[8001]:  val1's auc:0.929675  val2's auc:0.902119"
[1] "[9001]:  val1's auc:0.929963  val2's auc:0.902075"
[1] "[10000]:  val1's auc:0.930163  val2's auc:0.902129"
```

```
#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)
```

```
> getmodel_accuracy(conf_matrix)
[1] "accuracy 0.84"
[1] "precision 0.36"
[1] "recall 0.15"
```

```
> confusionMatrix(conf_matrix)
Confusion Matrix and Statistics

   lgbm_pred
       0     1
  0 30537  5474
  1   847  3142

               Accuracy : 0.842
                 95% CI : (0.8384, 0.8455)
    No Information Rate : 0.7846
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4194

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

            Sensitivity : 0.9730
            Specificity : 0.3647
         Pos Pred Value : 0.8480
         Neg Pred Value : 0.7877
             Prevalence : 0.7846
         Detection Rate : 0.7634
   Detection Prevalence : 0.9003
      Balanced Accuracy : 0.6688

       'Positive' Class : 0
```

```
# 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,

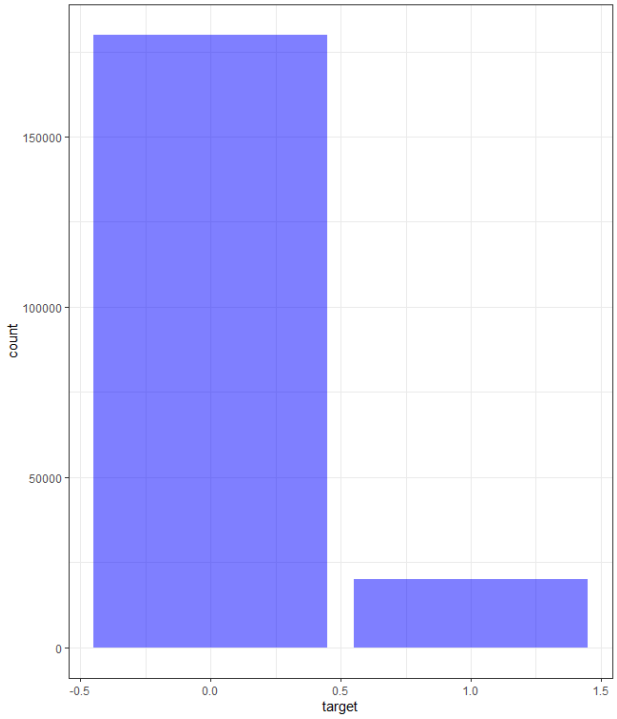
1. Decision Tree model performed poorly on our imbalanced data.
3. Logistic regression model performed average on our imbalanced data.
4. Light GBM model performed well on imbalanced data.

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

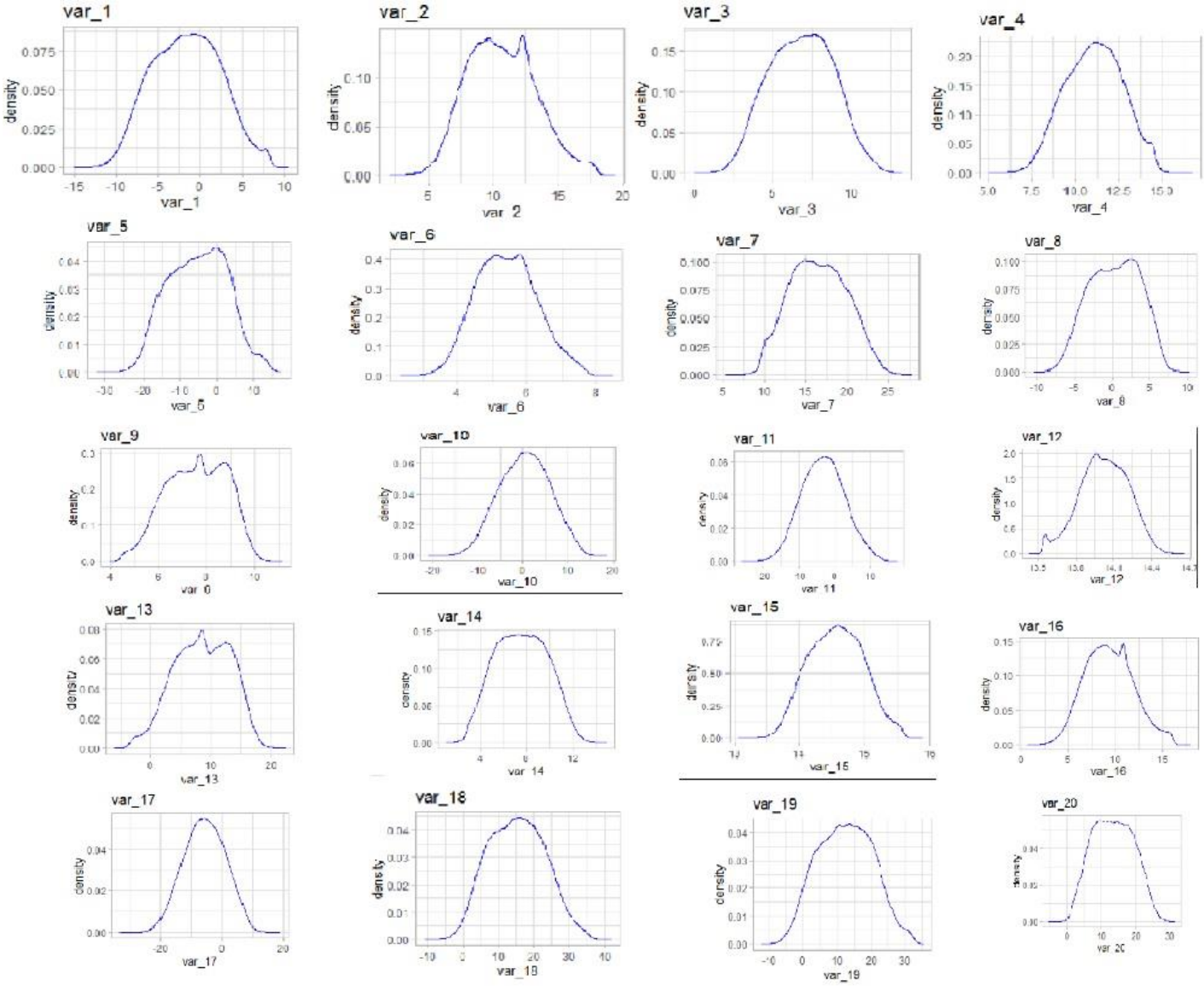
# Appendix A – Extra Figures

### Some ggplot2 visualizations

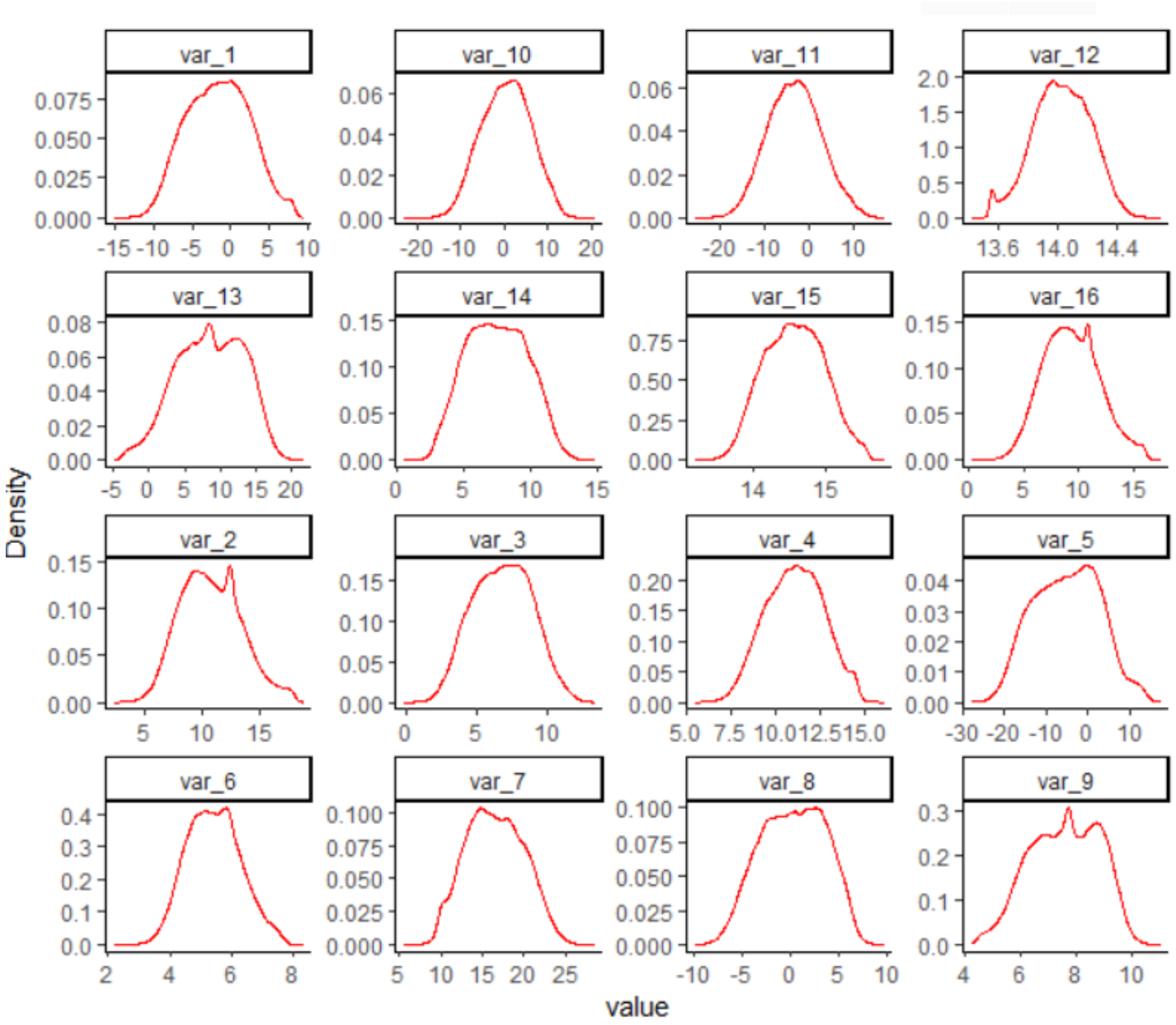
Target Value Distribution



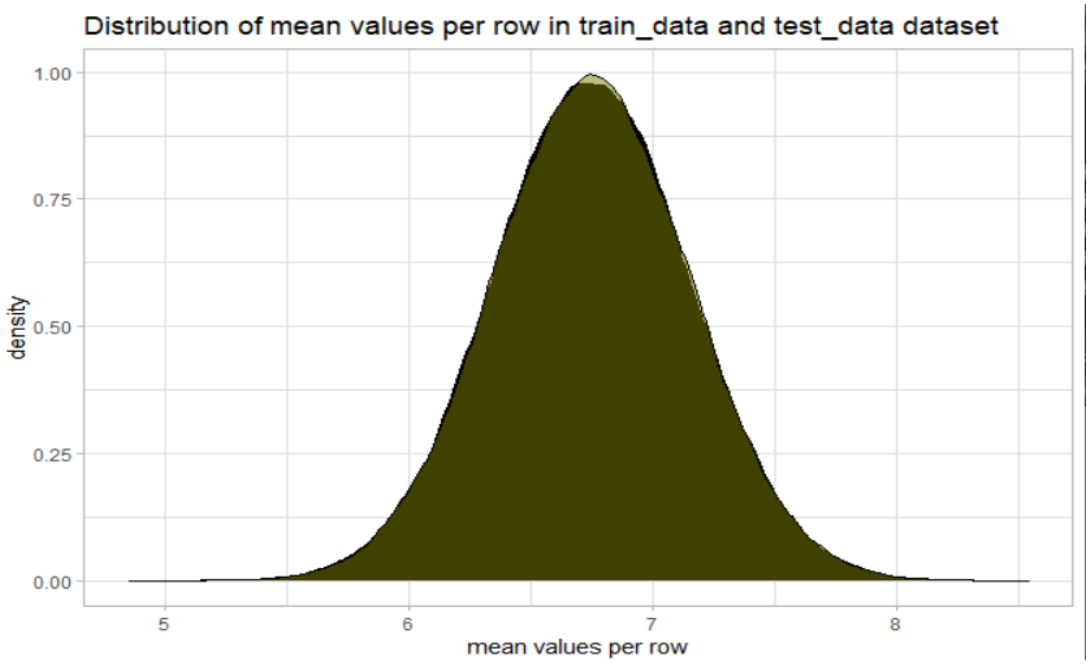
Distribution of train_data attributes from 3 to 22



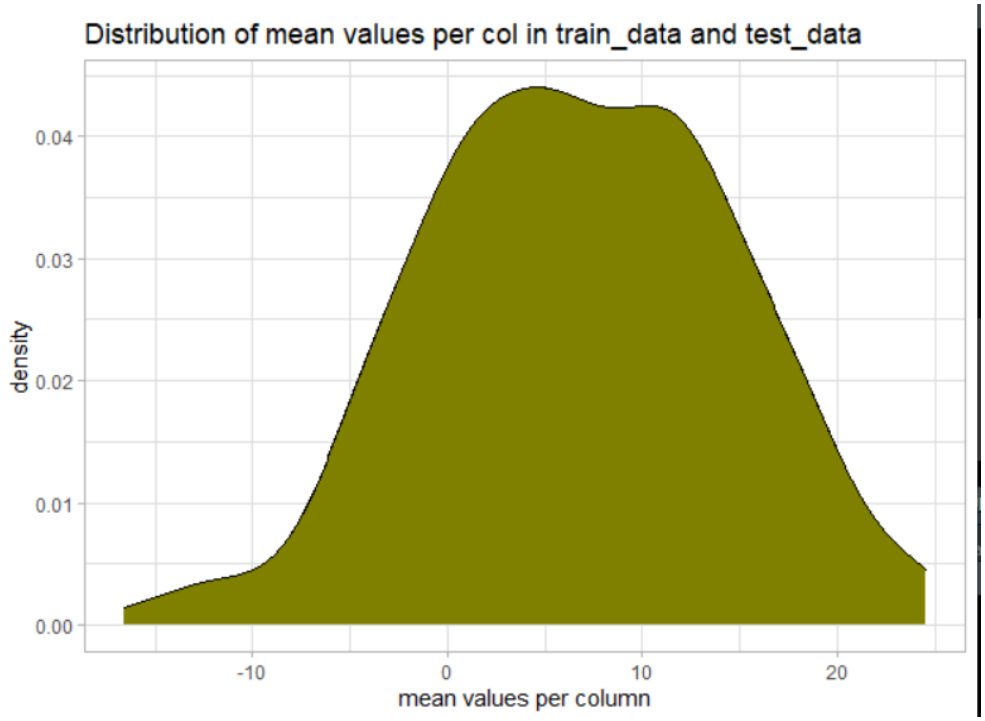
Distribution of test_data attributes from 2 to 18



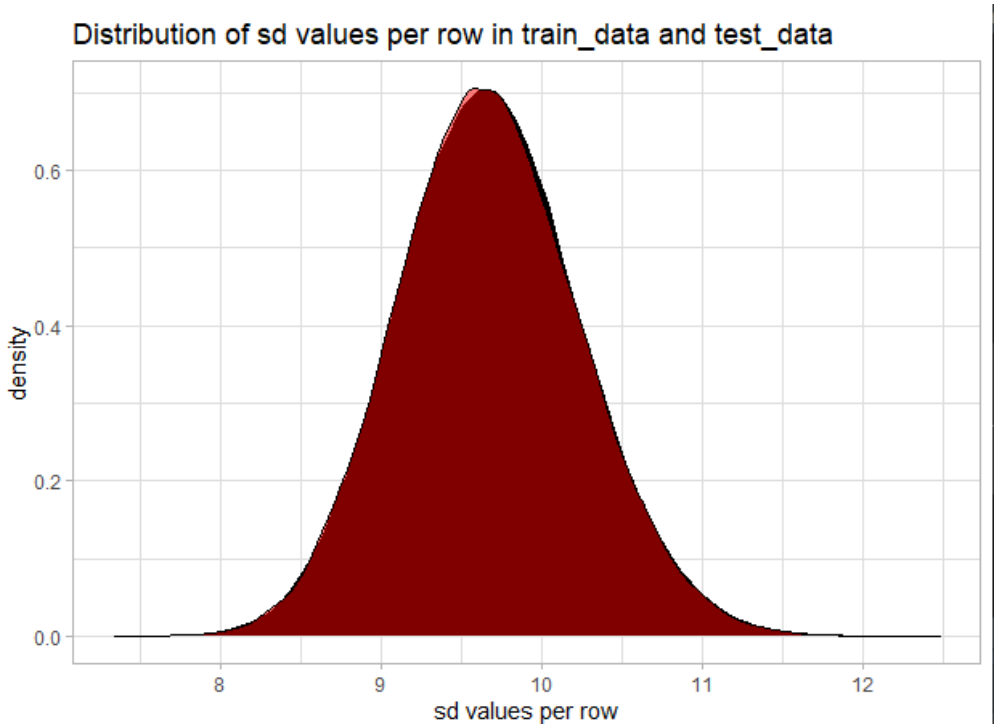
Mean values per row in train_data and test_data .



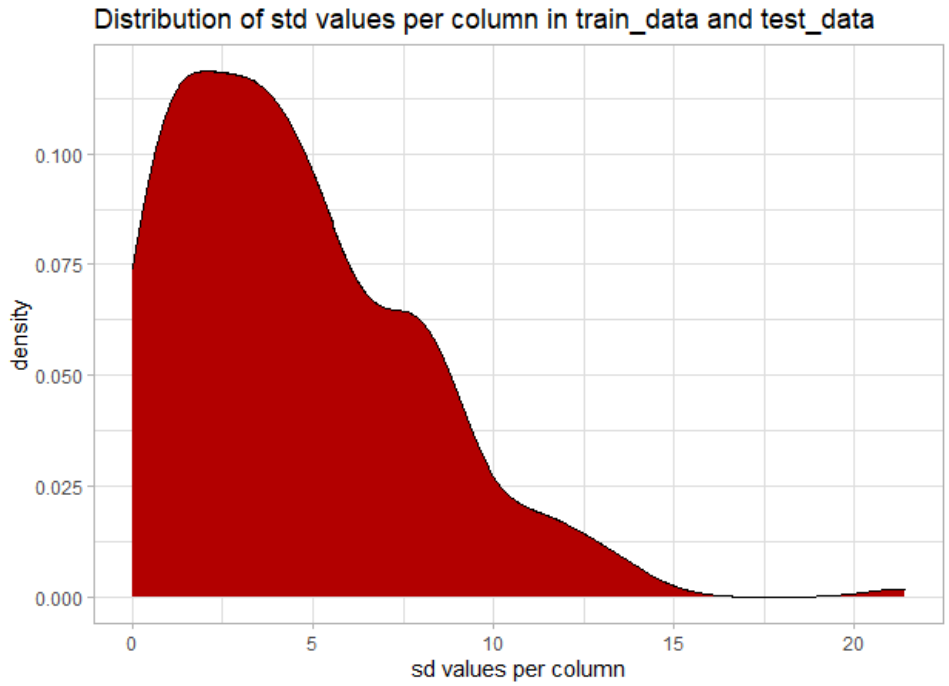

mean values per column in train_data and test_data



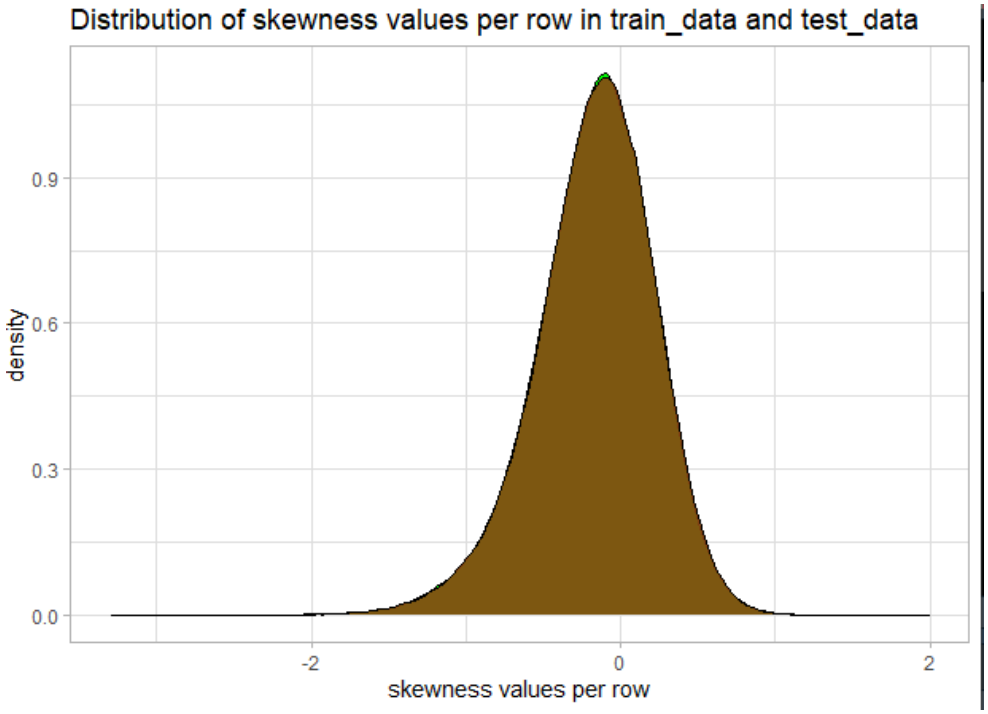
Standard deviation values per row in train_data and test_data



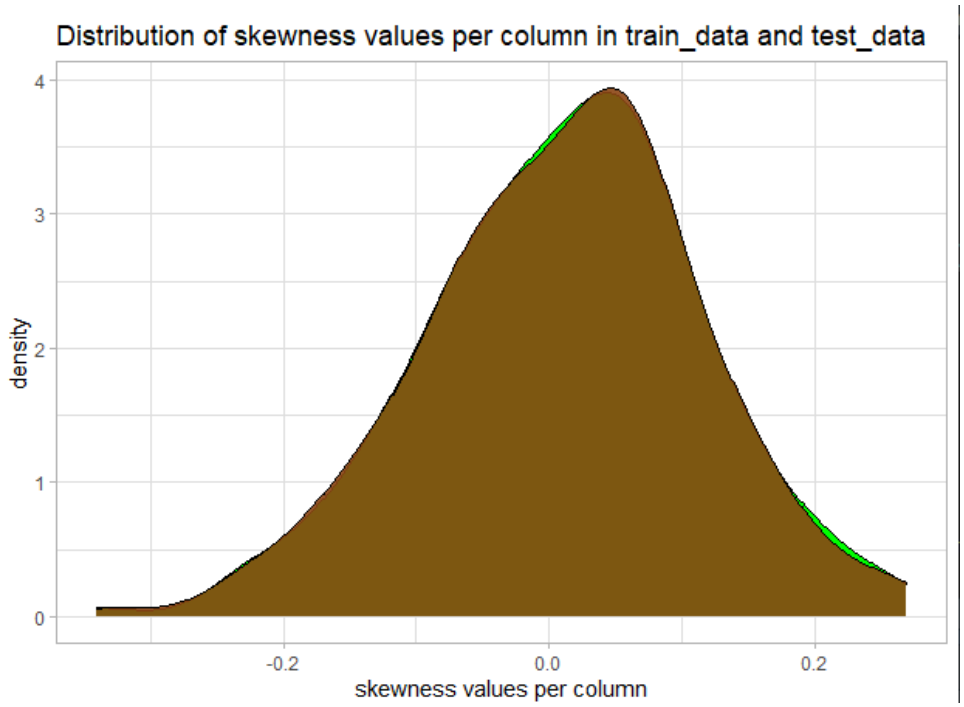

Standard deviation values per column in train_data and test_data



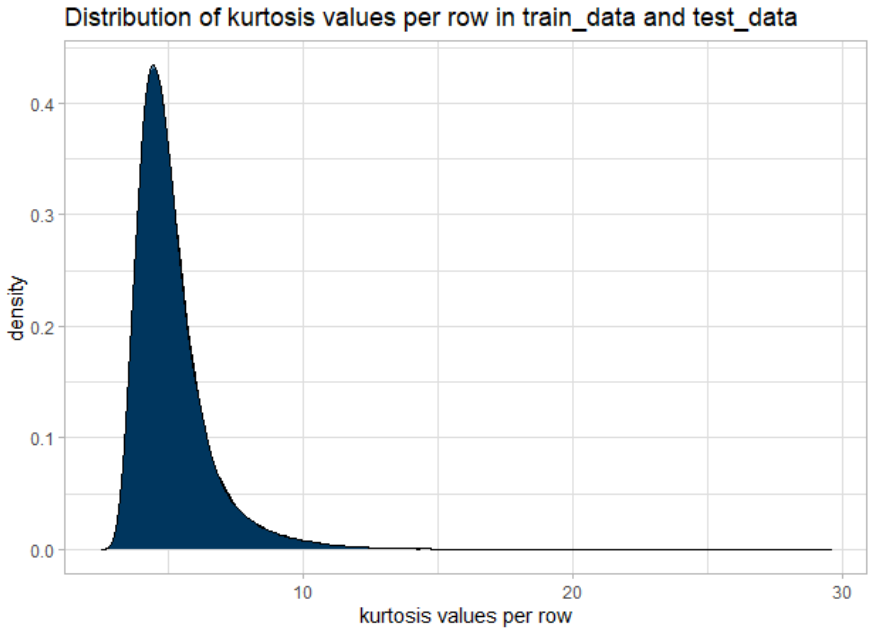
Skewness per row in train_data and test_data



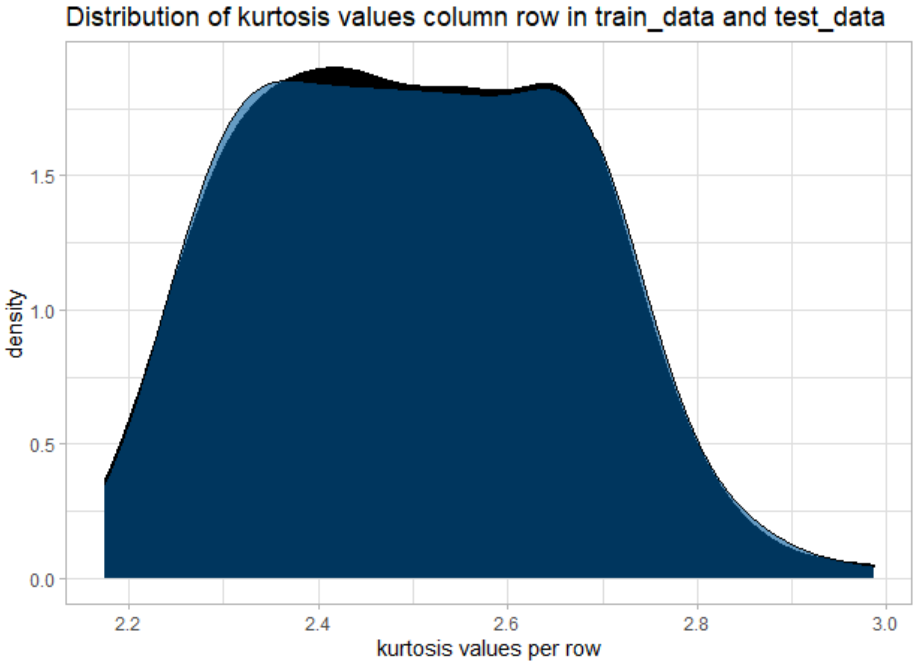

Skewness per column in train_data and test_data



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_l
ight()+
  #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_li
ght()+
  #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_li
ght()+
  #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',alph
a=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',alph
a=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 ")
```