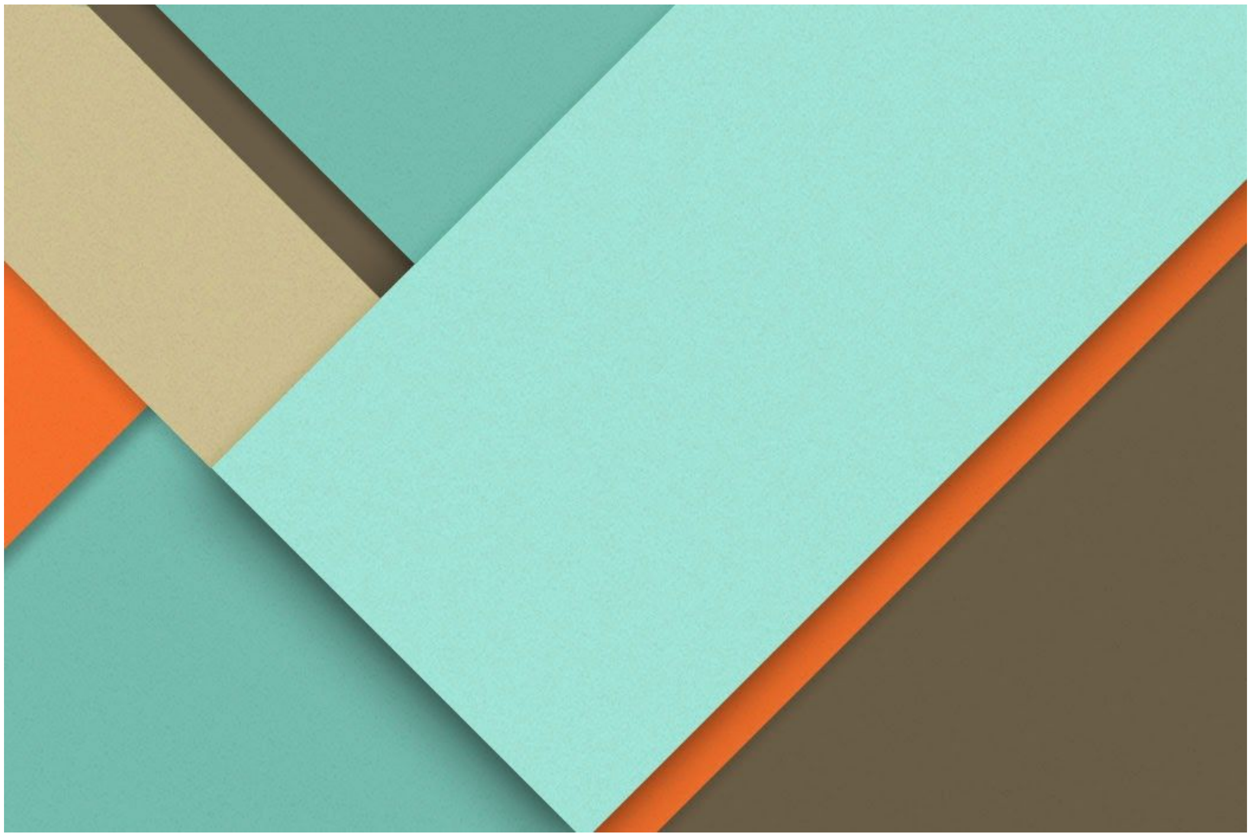


Santander Customer Transaction Prediction

15.09.2020

RAGHUNANDAN PAREEK

CHAPTER 1
INTRODUCTION

1.1 Problem Statement

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

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

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

1.2 Data

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

Table1.1 : Train dataset (Columns:1-202)

ID Code	Target	var_1	var_2	....	....	var_198	var_199
train_0	0	8.9255	-6.7863	....	....	12.7803	-1.0914
train_1	0	11.5006	-4.1473	....	....	18.3560	1.9518
train_2	0	8.6093	-2.7457	....	....	14.7222	0.3965
train_3	0	11.0604	-2.1518	....	....	17.9697	-8.9996
train_4	0	9.8369	-1.4834	....	....	17.9974	-8.8104

Table1.2 : Test dataset (Columns:1-201)

ID Code	var_1	var_2	var_3	....	....	var_198	var_199
test_0	11.0656	7.7798	12.9536	....	....	15.4722	-8.7197
test_1	8.5304	1.2543	11.3047	....	....	19.1293	-20.9760
test_2	5.4827	-10.3581	10.1407	....	....	19.8956	-23.1794
test_3	8.5374	-1.3222	12.0220	....	....	13.0168	-4.2108
test_4	11.7058	-0.1327	14.1295	....	....	13.9260	-9.1846

From the table below we have the following 200 variables,using which we will predict whether a customers will make a specific transaction in the future or not :

Table1.3 : Predictor Variables

S.NO	Predictor
1	var_0
2	var_1
3	var_2
4	var_3
5	var_4
6	var_5
. . . . . .	. . . . . .

....	....
197	var_196
198	var_197
199	var_198
200	var_199

CHAPTER 2

METHODOLOGY

2.1 Exploratory Data Analysis

Exploratory data analysis is an approach to analyzing data sets to summarize their main characteristics, often with visual methods. Exploratory data analysis is one of the most important steps in data mining in order to know features of data. It involves loading the dataset,data cleaning,normality test,typecasting of attributes, missing value analysis, outlier analysis, Attributes distributions and trends. So, we have to clean the data otherwise it will affect the performance of the model. Now we are going to explain one by one as follows.

2.1.1 Missing Value Analysis

Missing Values occur when no data value is stored for the variable in an observation. Missing data are a common occurrence and can have a significant effect on the conclusions that can be drawn from the data missing Values occur when no data value is stored for the variable in an observation. Missing data are a common occurrence and can have a significant effect on the conclusions that can be drawn from the data.

Missing Values in Dataset

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

Table2.1 : Missing Values in Train Set

Predictor	No Of Missing Values
Target	0
var_0	0
var_1	0
var_2	0
var_3	0
var_4	0
var_5	0
. . . . . .	0
. . . . . .	0
var_196	0
var_197	0
var_198	0
var_199	0
Total Missing Values in Train Set	**0**

Table2.2 : Missing Values in Test Set

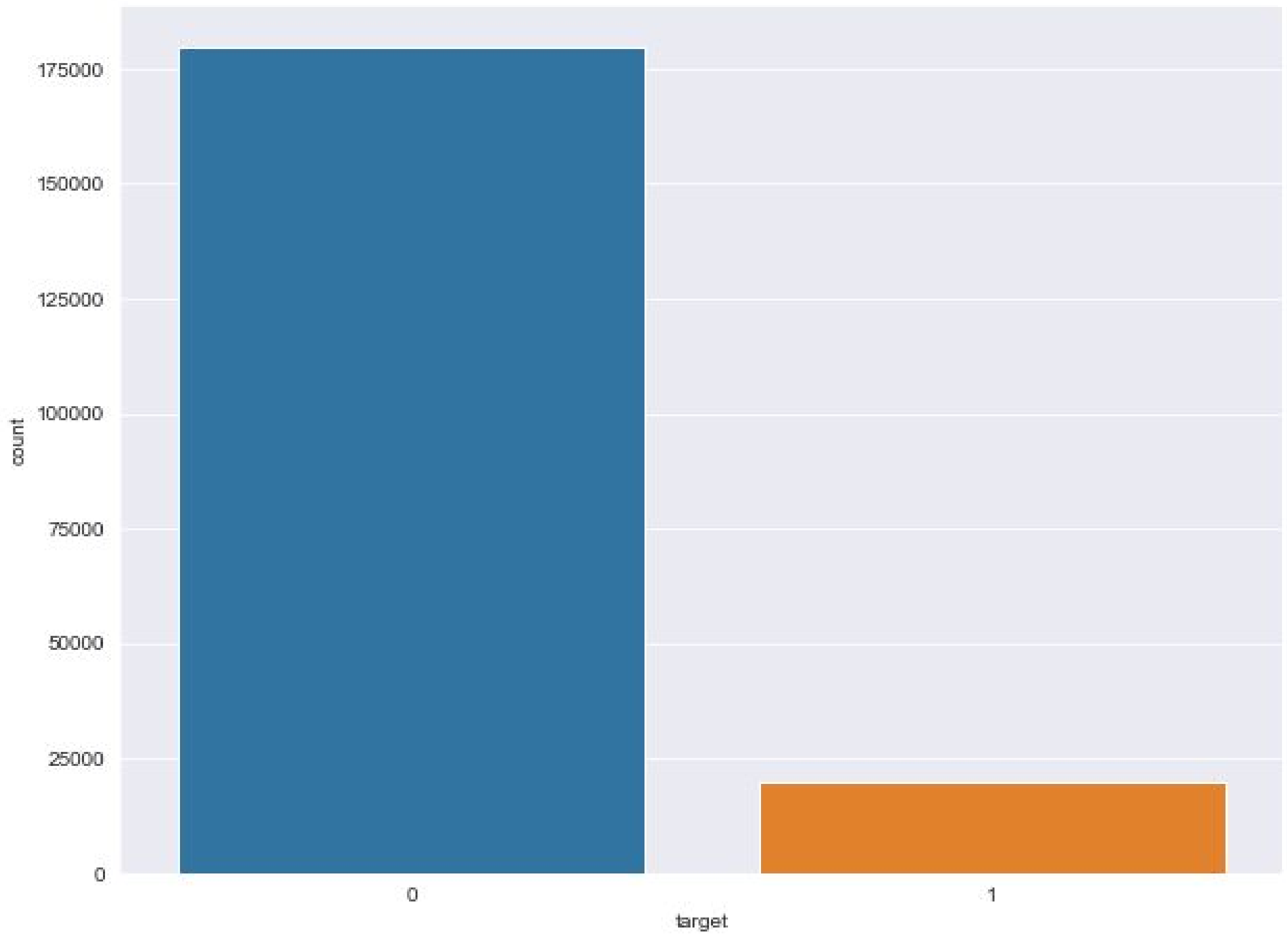
Predictor	**No Of Missing Values**
Target	0
var_0	0
var_1	0
var_2	0
var_3	0
var_4	0
var_5	0
......	0
......	0
var_196	0
var_197	0
var_198	0
var_199	0
Total Missing Values in Test Set	**0**

2.1.2 Attributes Distributions and Trends

2.1.2.1 Target classes

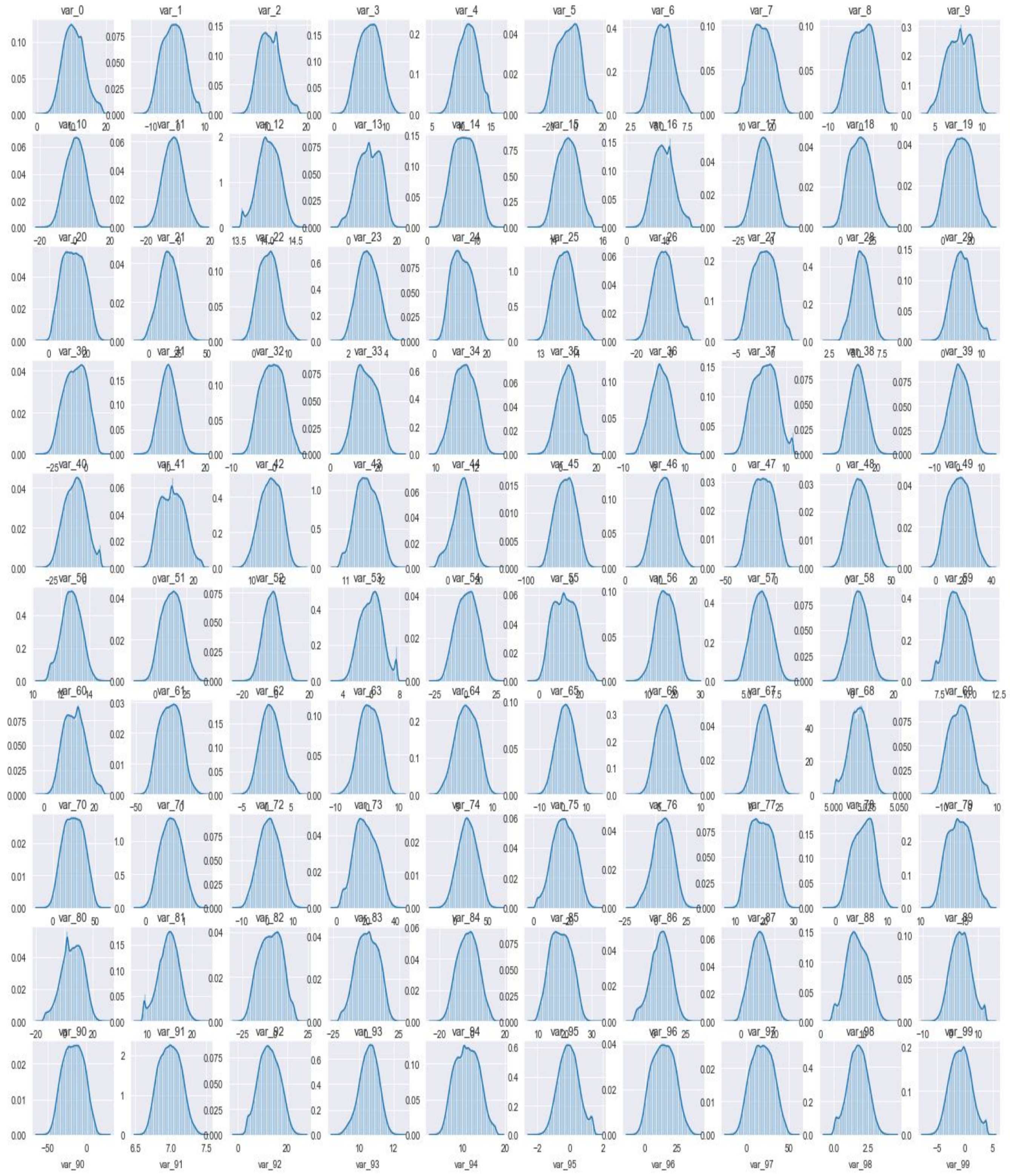
Target 0 -> No Transaction

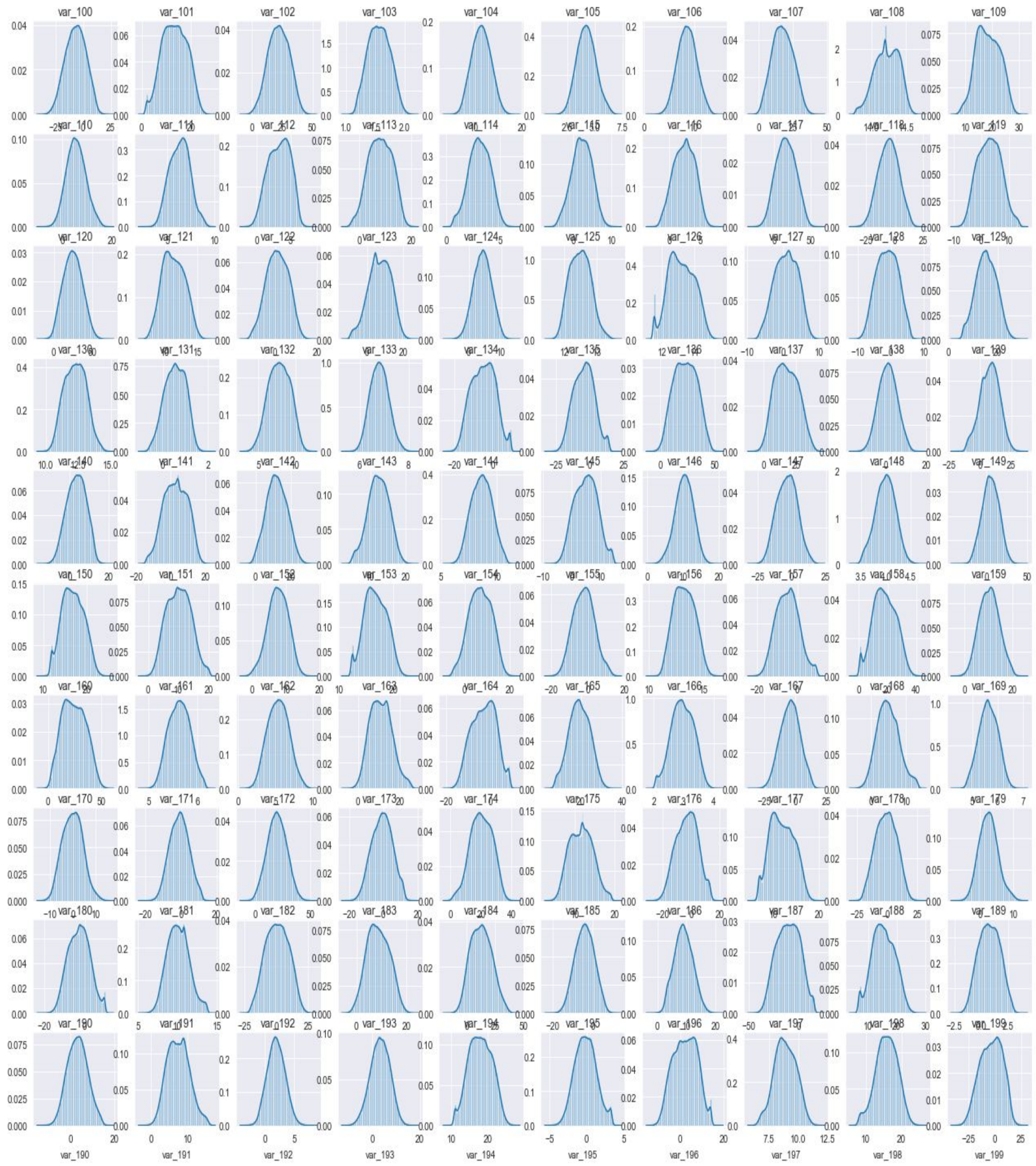
Target 1 -> Transaction Completed



From the above plot it can be observed that the dataset we have is class imbalanced data, where 90% of the data is no. of customers will not make a transaction & 10 % of the customers are those who will make a transaction.

2.1.2.2 Distribution Plot of Train Dataset Features

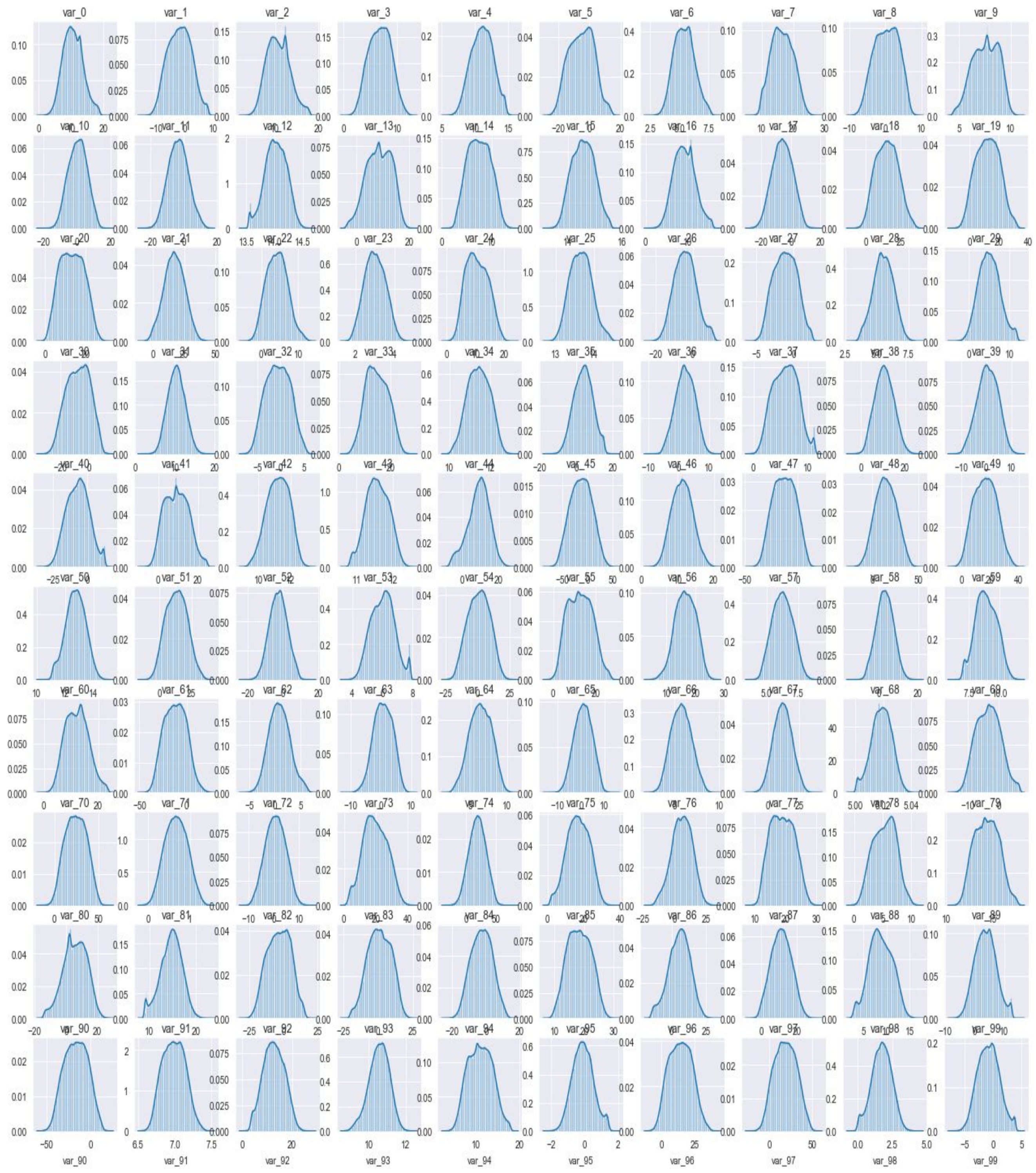


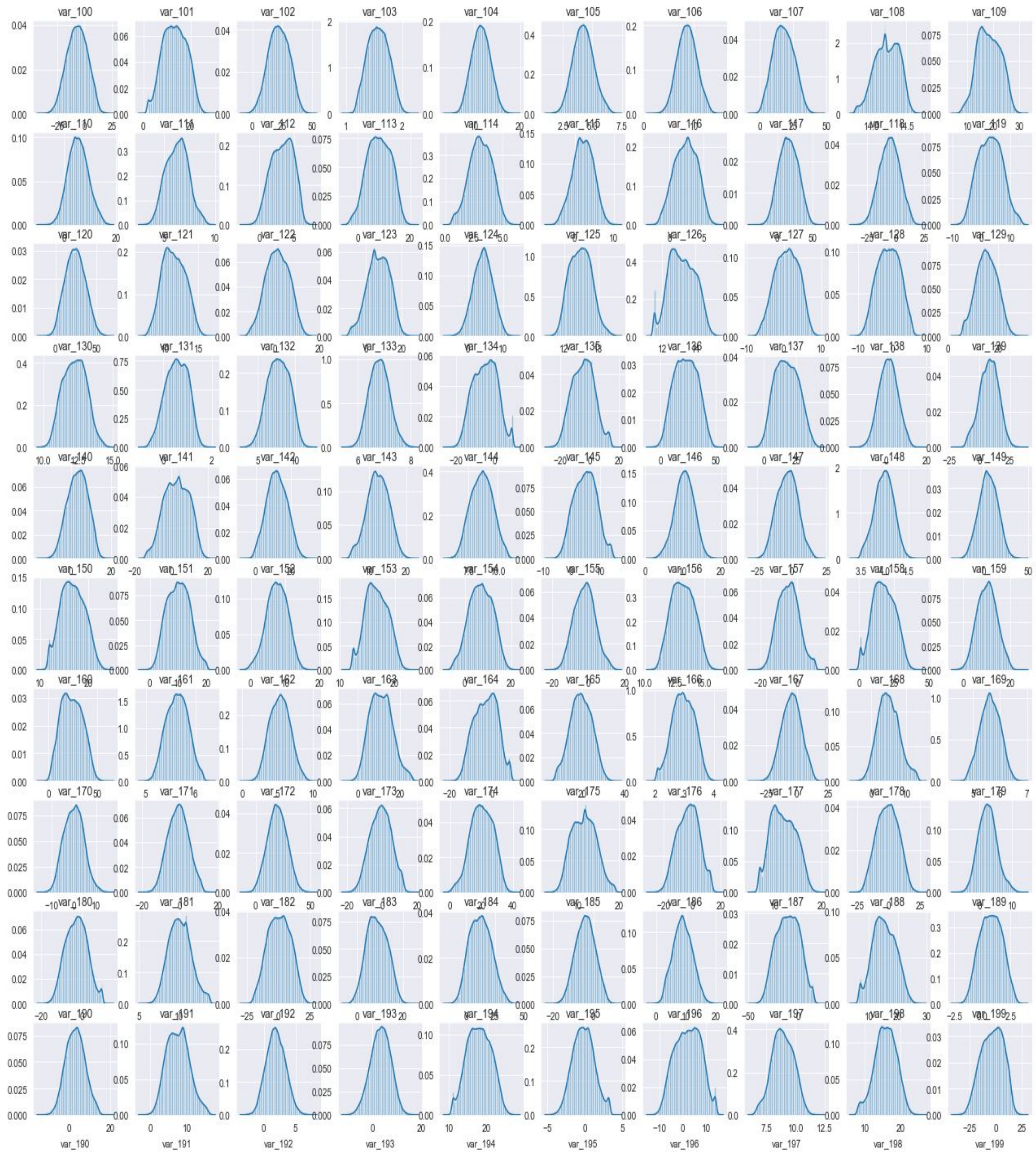


Distribution Plot of Train Dataset Features

Not all the features follow normal distribution features like var_4, var_10, var_11, var_17, var_23 and many more follows a normal distribution but features like var_1, var_2, var_9, var_12, var_13 doesn’t follow a normal distribution.

2.1.2.2 Distribution Plot of Test Dataset Feature

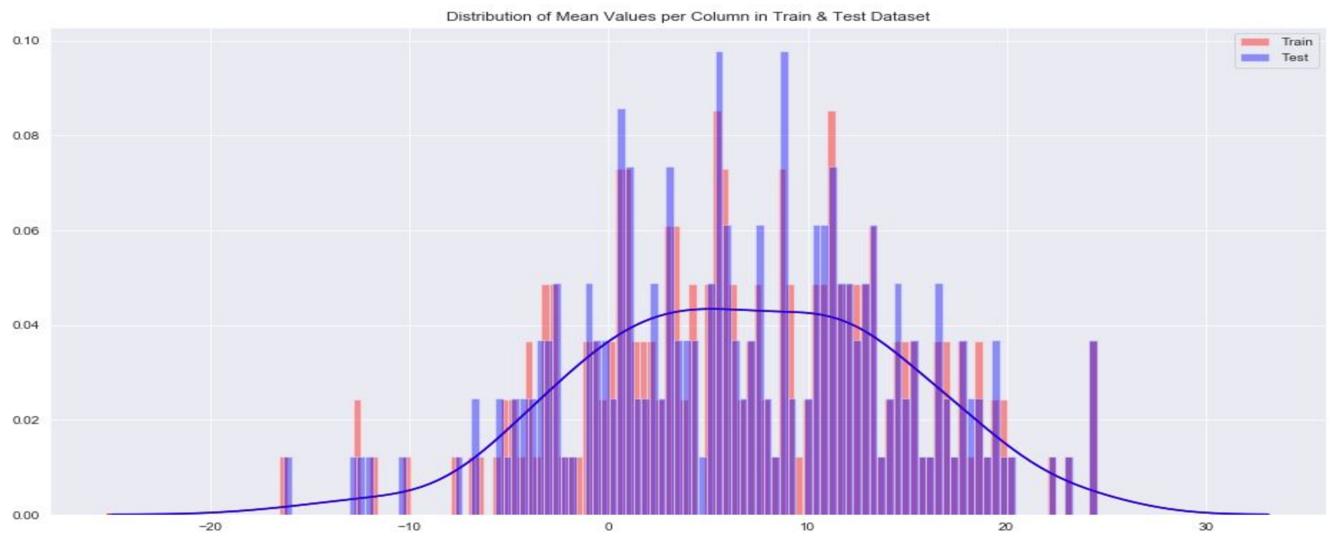




Distribution Plot of Test Dataset Features

Not all the features follow normal distribution features like var_1, var_3, var_10, var_11, var_17 and many more follows a normal distribution but features like var_2, var_6, var_9, var_13, var_16 doesn't follow a normal distribution.

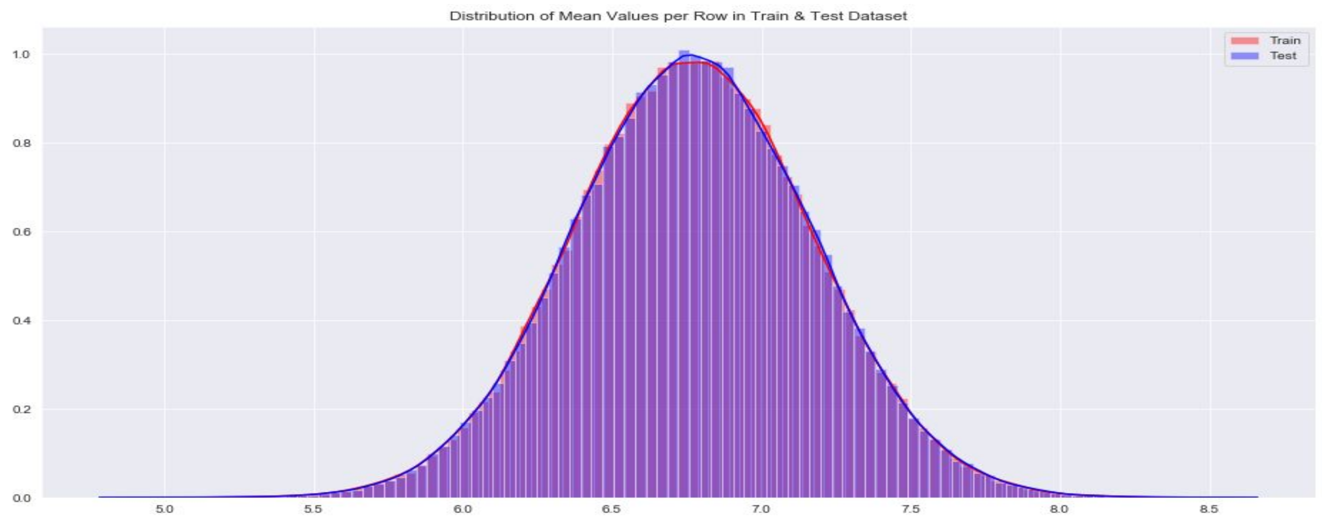
2.1.2.3 Distribution Plot of Mean Values



Distribution of Plot Mean Values per Column in Train & Test Dataset

Following observation is made after looking at the mean graph for column values of data

- The column wise mean distribution is not Gaussian.
- The majority of columns have the mean value between -10 and 20.

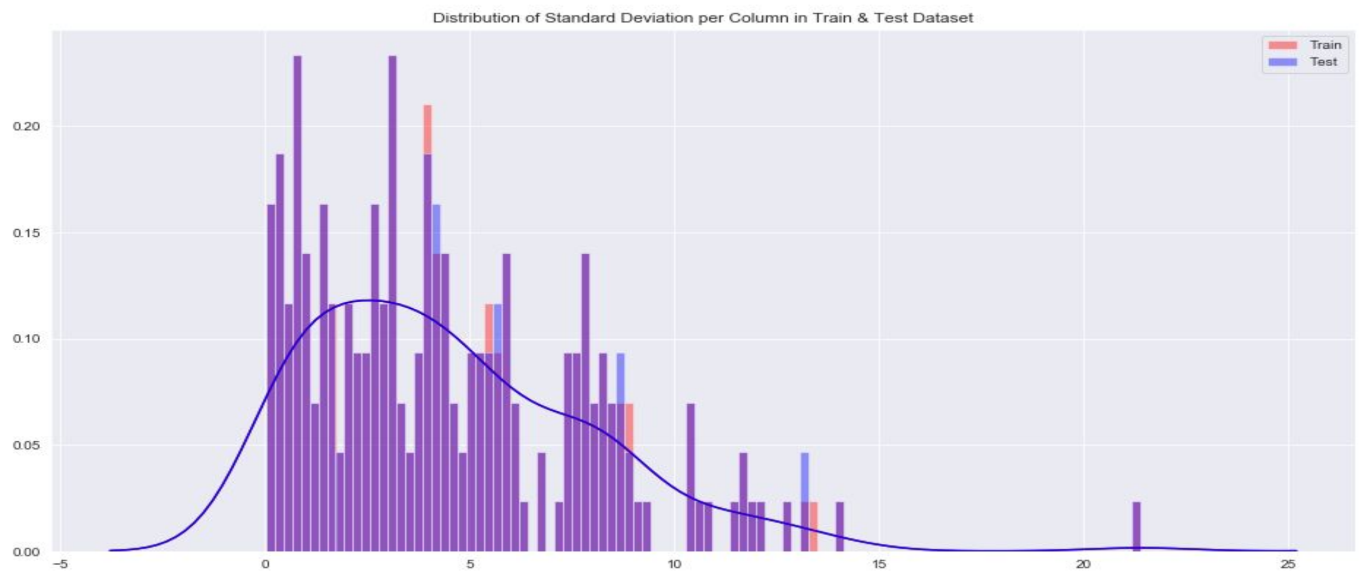


Distribution Plot of Mean Values per Row in Train & Test Dataset

Following observation is made on the mean graph for row values of data:

- The graph looks kind of Gaussian with a mean value of 6.7.
- From the above graph, we can say that there are around 90% of a feature whose mean lies between 6 and 7.5.

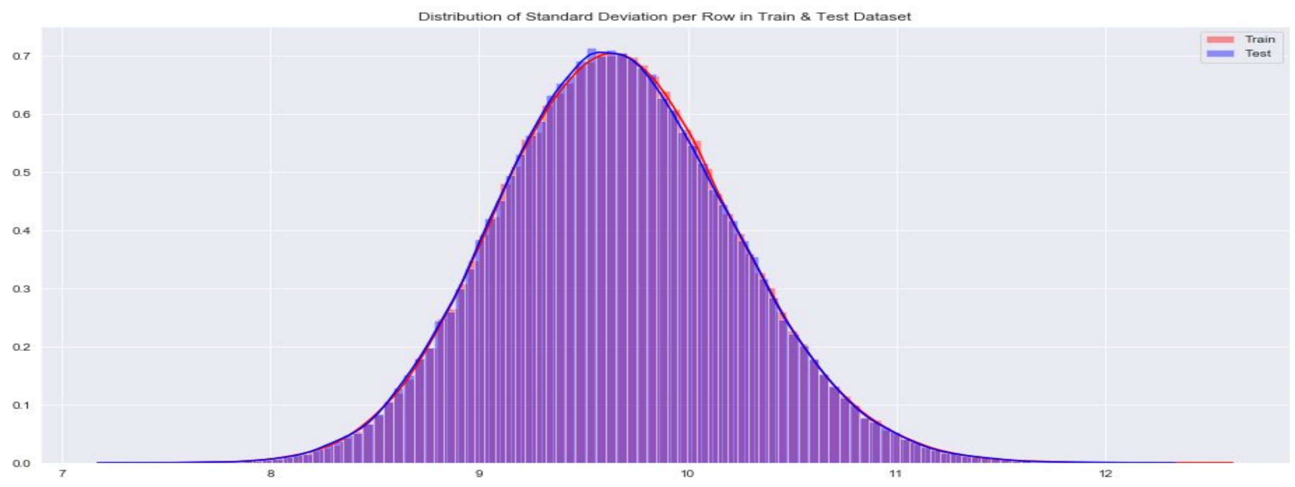
2.1.2.4 Distribution Plot of Standard Deviation



Distribution Plot of Standard Deviation per Column in Train & Test Dataset

Following observation is made on Standard Deviation for column values of data

- The column wise Standard Deviation distribution is not Gaussian.
- The majority of columns have the mean value between 0 and 15.

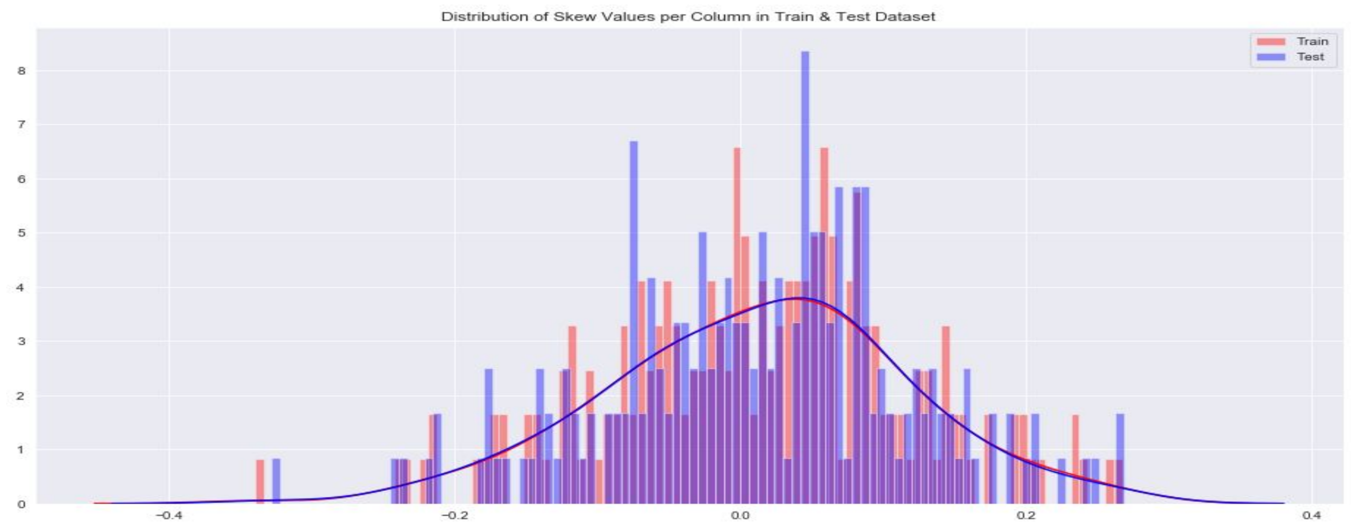


Distribution Plot of Standard Deviation per Column in Train & Test Dataset

Following observation is made on Standard Deviation for row values of data

- The row wise distribution kind of follows the Gaussian distribution
- Around 90% of features having the standard deviation around in the range of 8.5–11.

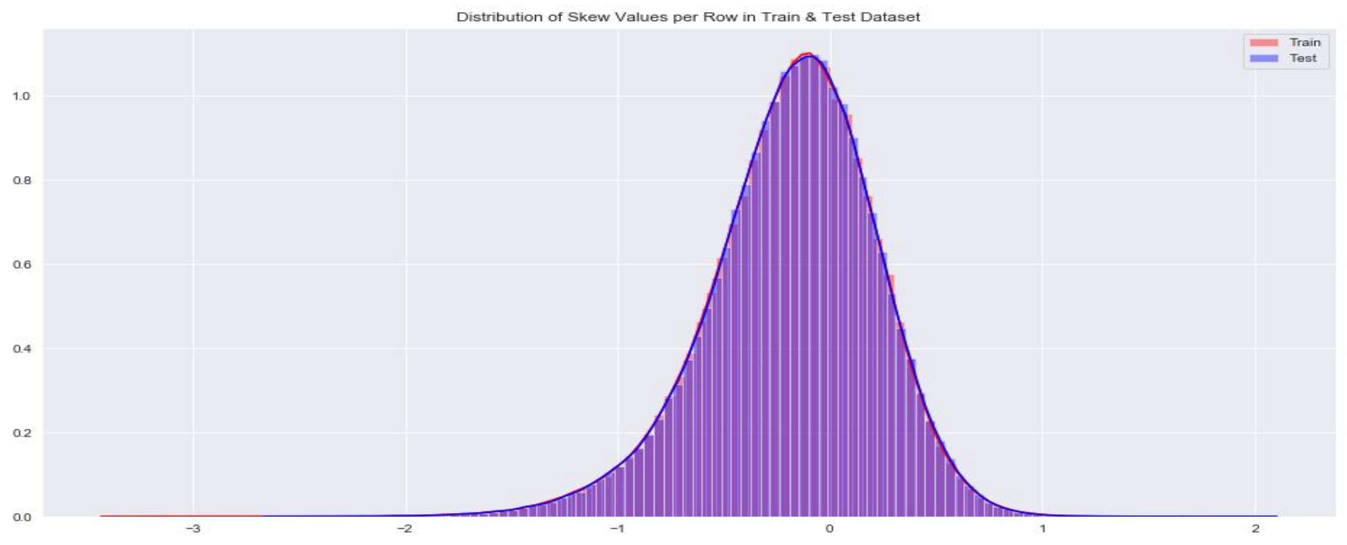
2.1.2.5 Distribution Plot of Skew Values



Distribution Plot of Skew Values per Column in Train & Test Dataset

Following observation is made on Skew Values for column of data

- The Column wise distribution does not follows the Gaussian distribution
- Around 90% of features have the Skew Values around in the range of -0.2 to 0.2.

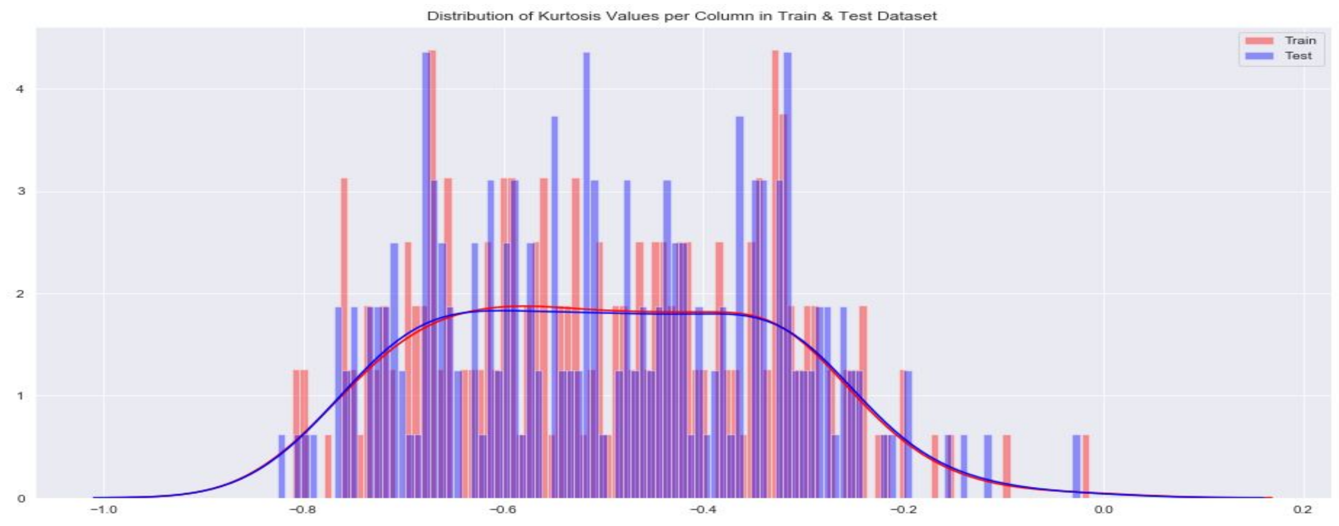


Distribution Plot of Skew Values per Row in Train & Test Dataset

Following observation is made on Skew Values for row of data

- The Row wise distribution follows the Gaussian distribution
- Around 90% of features have the Skew Values around in the range of -1 to 1..

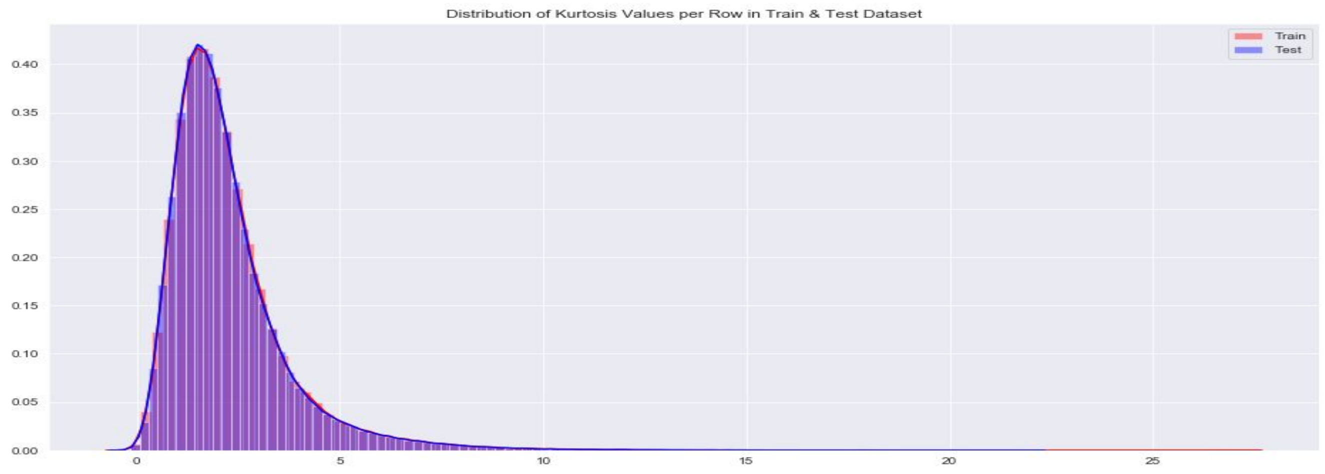
2.1.2.6 Distribution Plot of Kurtosis Values



Distribution Plot of Kurtosis Values per Row in Train & Test Dataset

Following observation is made on Kurtosis Values for column of data

- The Column wise distribution does not follows the Gaussian distribution
- Around 90% of features have the Kurtosis Values around in the range of -0.8 to 0.2.



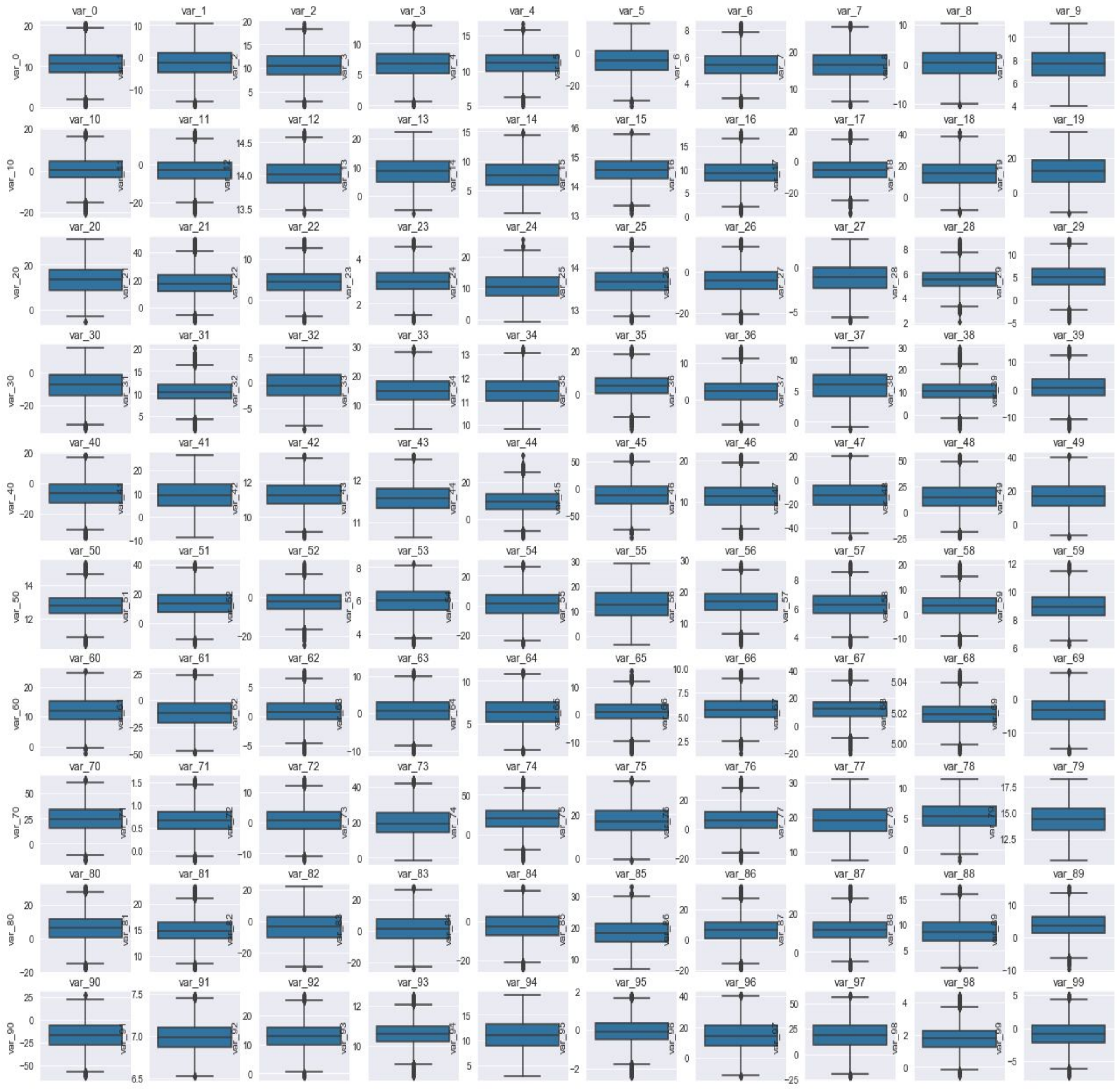
Distribution Plot of Kurtosis Values per Row in Train & Test Dataset

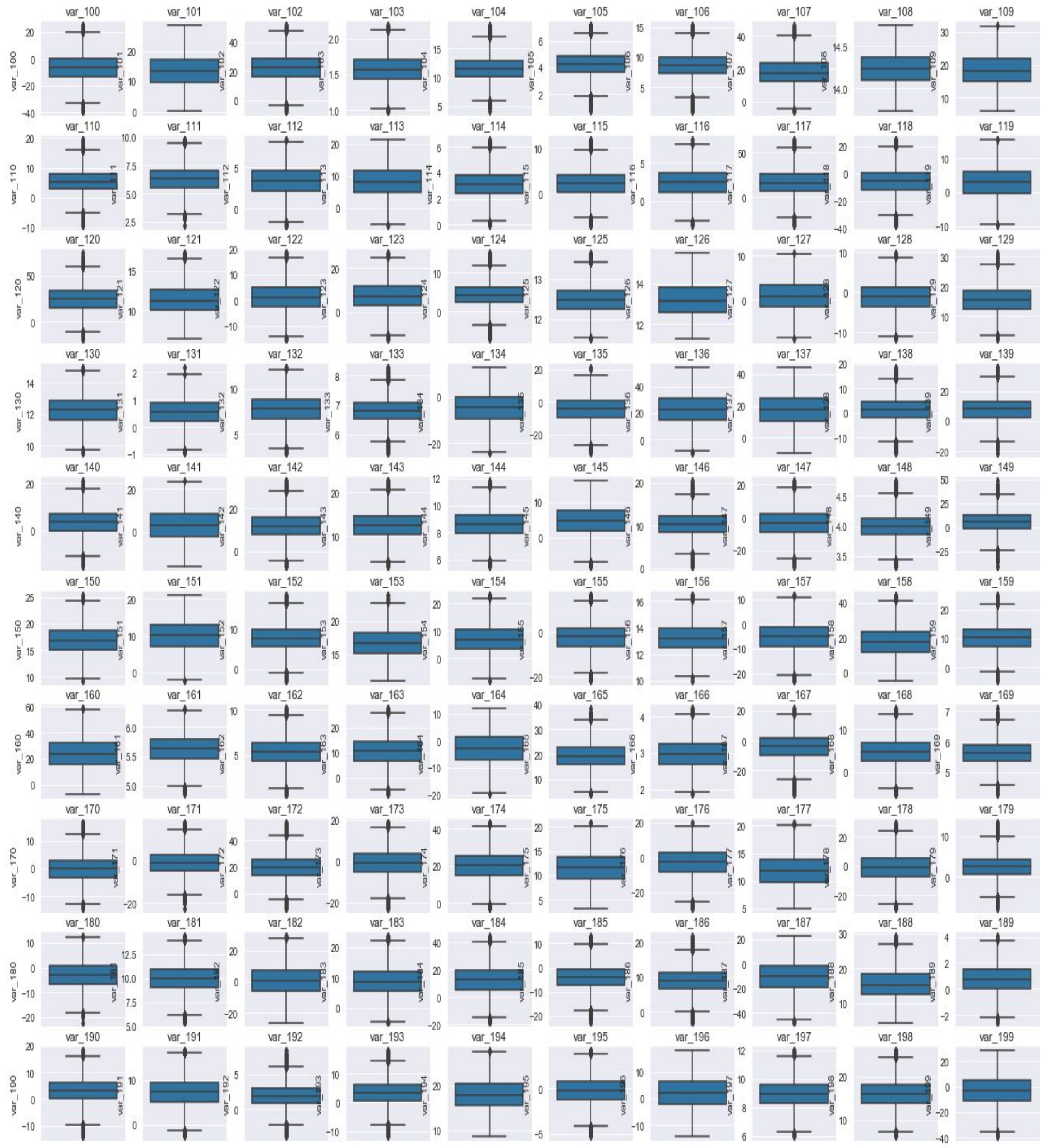
Following observation is made on Kurtosis Values for column of data

- The Column wise distribution does not follows the Gaussian distribution
- Around 90% of features have the Kurtosis Values around in the range of -0.8 to 0.2.

2.1.3 Outlier Analysis

Outliers are the object that deviates significantly from the rest of the objects. They can be caused by measurement or execution error. The analysis of outlier data is referred to as outlier analysis or outlier mining.





Boxplot Plot of Features of Train Dataset

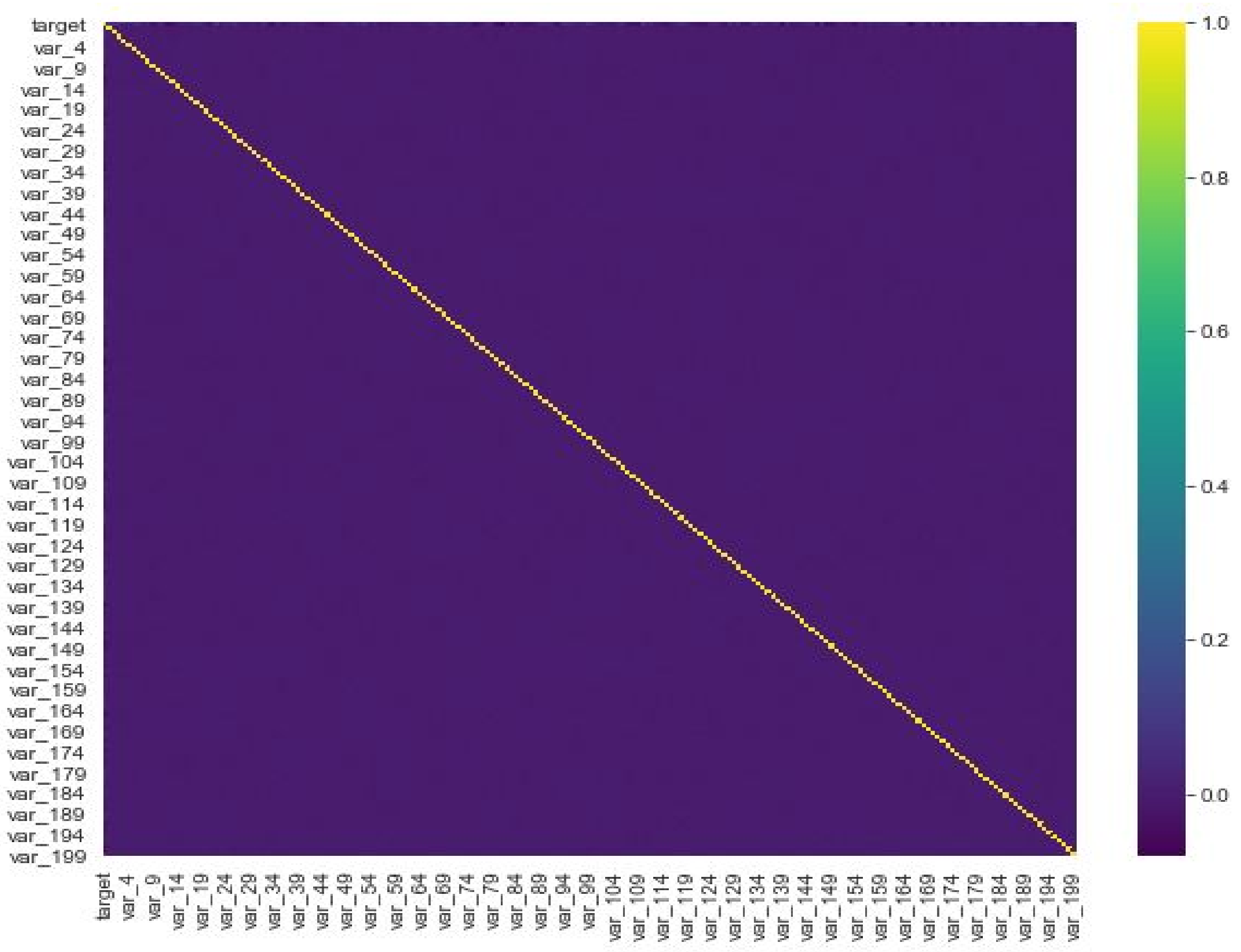
Following observation is made on the train dataset

- Around 90% of features are having outliers on the lower and higher quantiles.

2.1.4 Feature Selection

Feature selection is the process of reducing the number of input variables when developing a predictive model.It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model.

There are several methods of feature selection.We have used Correlation matrix for feature selection**.**



Following observation is made by heatmap of variables-

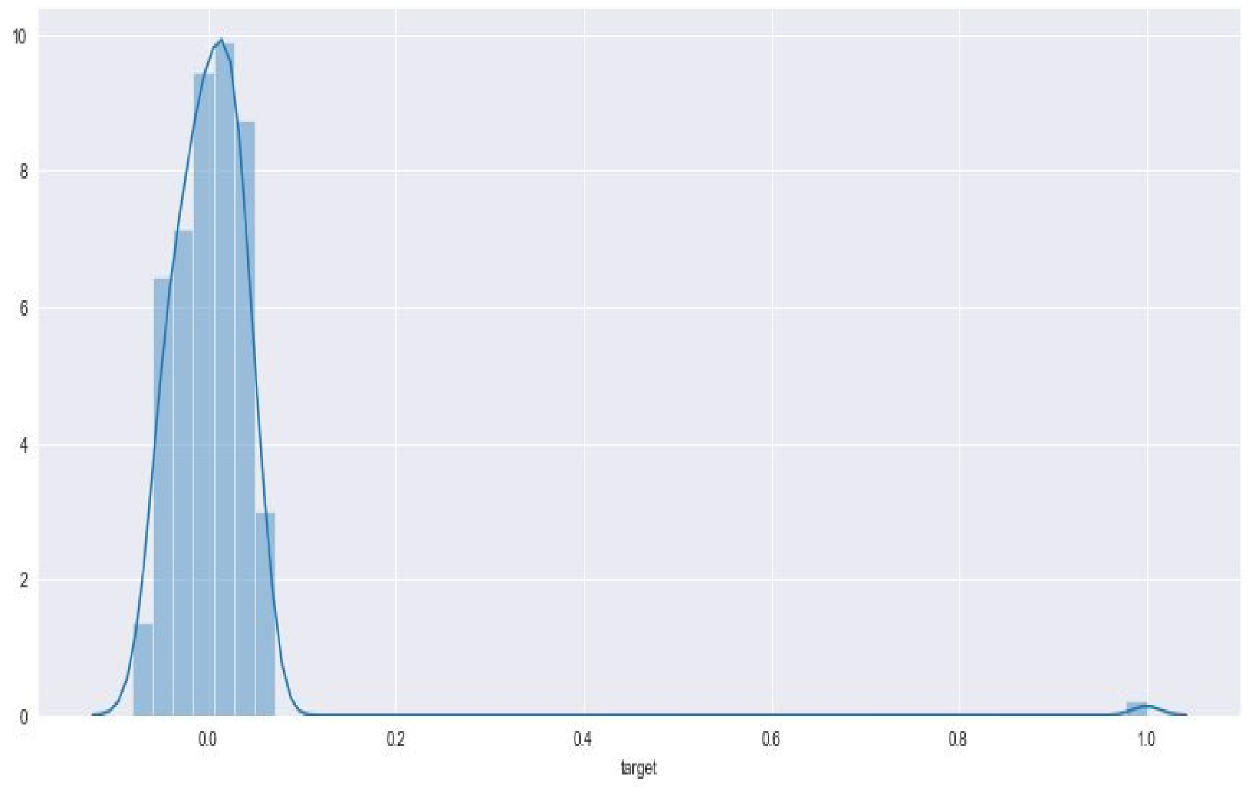
- All the variables have very negligible correlation with each other.

2.1.4.1 Correlation Matrix

Correlation is any statistical relationship, whether causal or not, between two random variables or bivariate data. In the broadest sense correlation is any statistical association, though it commonly refers to the degree to which a pair of variables are linearly related.

Top 10 correlated features of the train dataset are as below-

Column	**Correlation Score**
var_81	-0.080917
var_139	-0.074080
var_12	-0.069489
var_6	0.066731
var_110	0.064275
var_146	-0.063644
var_53	0.063399
var_26	0.062422
var_76	-0.061917
var_22	0.060558



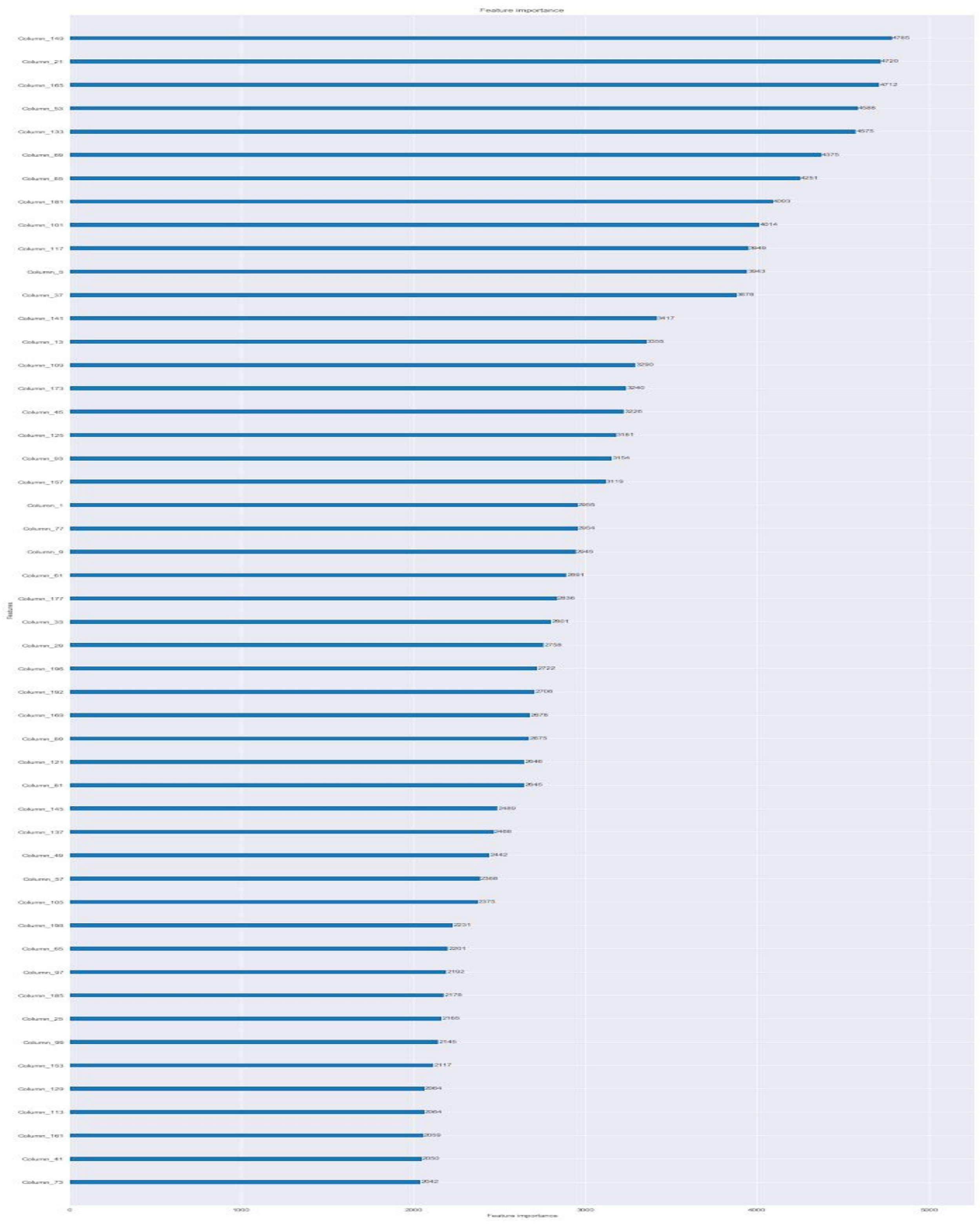
Following observation is made by correlation graph of features with target-

- Most of the features have very small correlation scores with target between -0.081 to 0.067.

Almost all the features have almost the same correlation score so we will select all the 200 features for the model.

2.1.5 Feature Importance

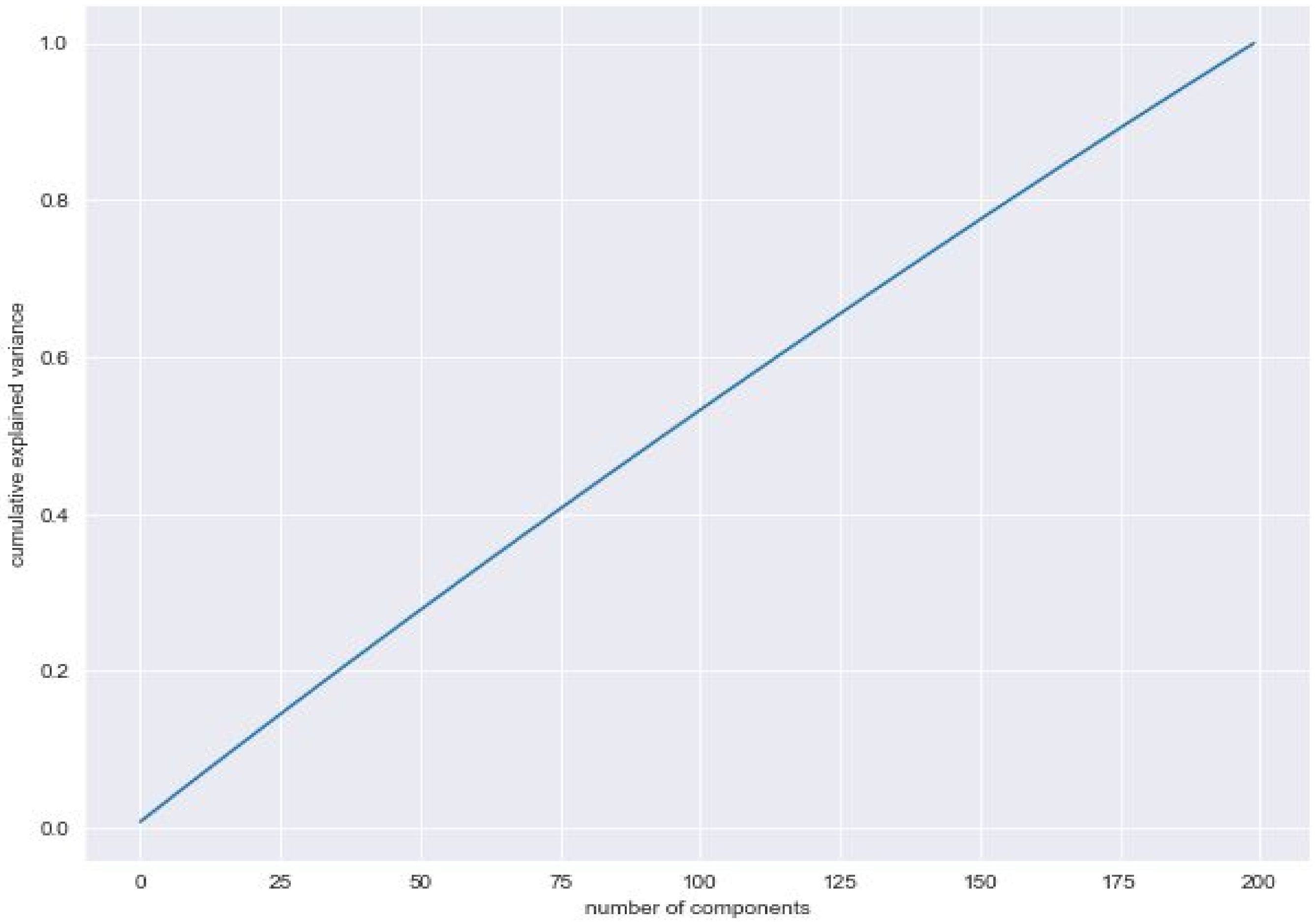
Feature importance refers to a class of techniques for assigning scores to input features to a predictive model that indicates the relative importance of each feature when making a prediction. There are many types and sources of feature importance scores, although popular examples include statistical correlation scores, coefficients calculated as part of linear models, decision trees, and permutation importance scores.



According var_149, var_21, var_165, var_53 and var_133 are the most important features whereas var_73, var_41, var_161, var_113, var_129, and var_153 are the least important features

2.1.6 Principle Component Analysis

Principal Component Analysis is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.



By observing the graph the to explain 90% of the variance of the data we have to choose 180 components which makes dimensionality reduction redundant so we will not use Principal Component Analysis in the problem

2.2 Modeling

2.2.1 Model Selection

Model selection is the process of choosing one among many candidate models for a predictive modeling problem.

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

1. Nominal
2. Ordinal
3. Interval
4. Ratio

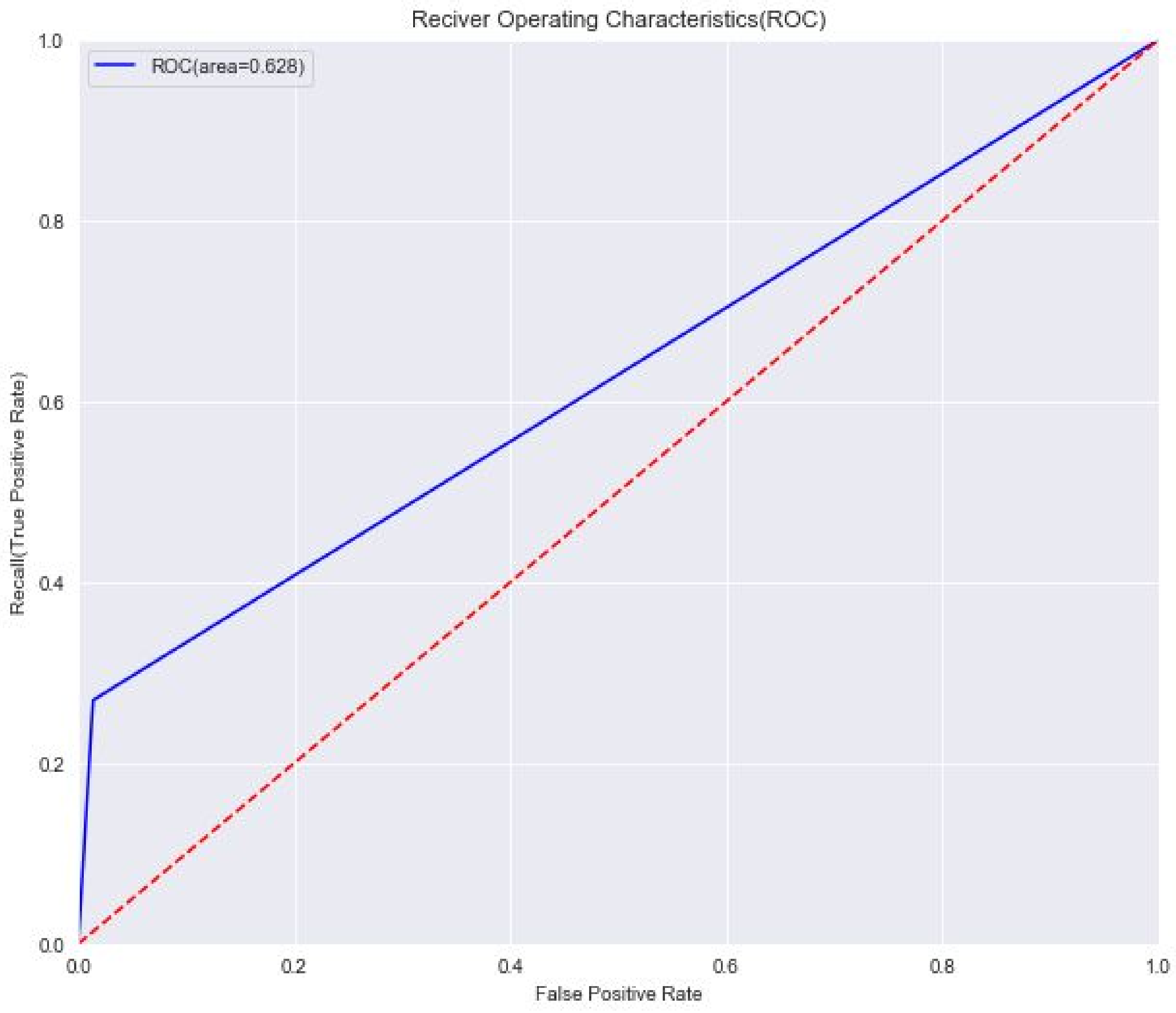
If the dependent variable is Nominal the only predictive analysis that we can perform is Classification, and if the dependent variable is Interval or Ratio, the normal method is to do a Regression analysis, or classification after binning. But the dependent variable we are dealing with is nominal, for which classification can be done. You always start your model building from the most simplest to more complex.

2.2.2 Logistic Regression

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

Reason To use Logistic Regression

- Easier to inspect and less complex.
- Robust algorithm as the independent variables need not have equal variance or normal distribution.
- These algorithms do not assume a linear relationship between the dependent and independent variables and hence can also handle non-linear effects.
- Controls confounding and tests interaction.



Reason To reject Logistic Regression-

- Low AUC Score
- Low Precision Score
- Low Recall Score
- Low F1 Score
- Low Accuracy

2.2.3 Logistic Regression with Synthetic MInority Oversampling Technique

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis.

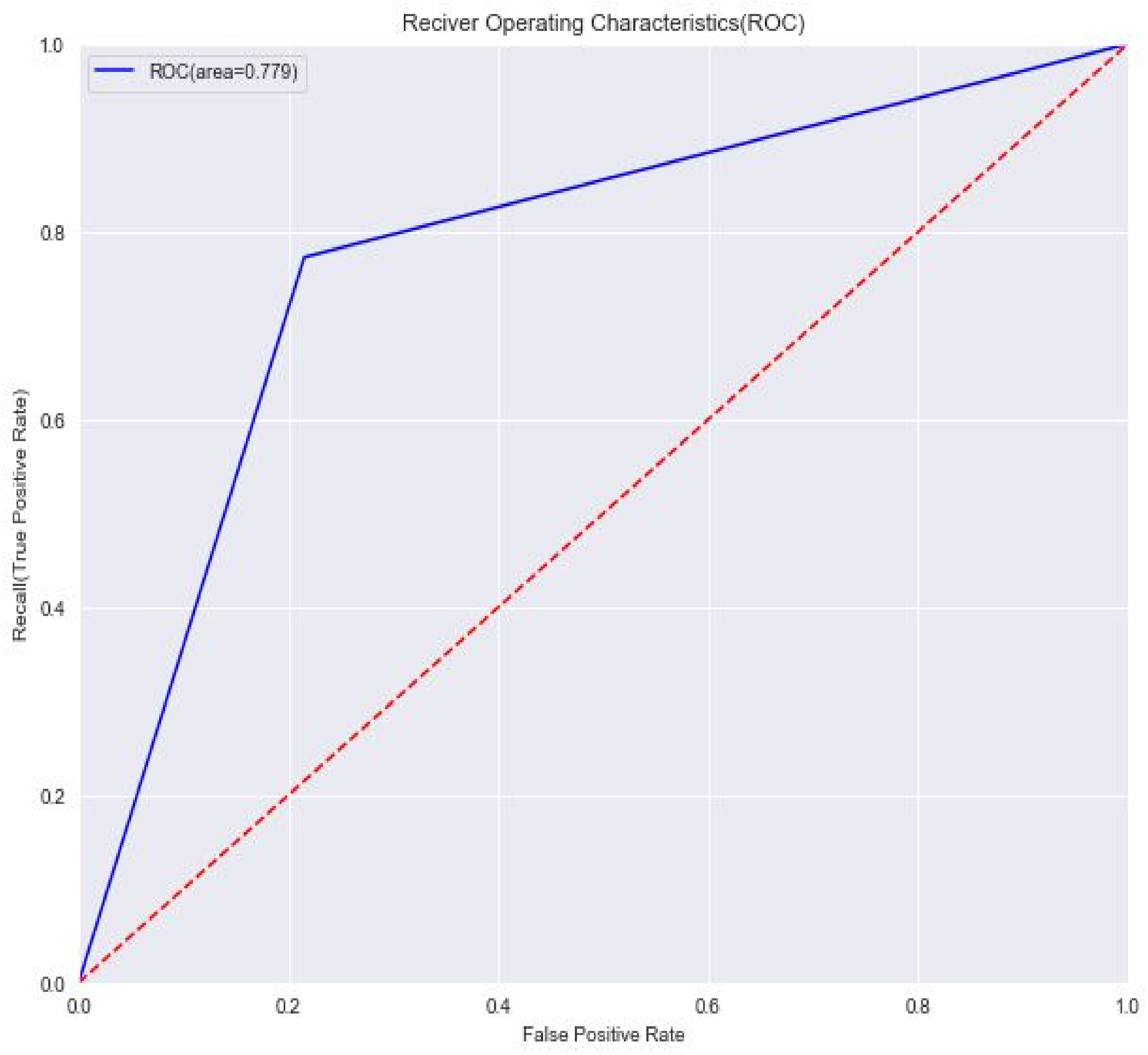
2.2.3.1 Synthetic MInority Oversampling Technique(SMOTE)

SMOTE is followed to avoid overfitting which occurs when exact replicas of minority instances are added to the main dataset. A subset of data is taken from the minority class as an example and the new Synthetics instances are then added to the original dataset. The new dataset is used as a sample to train the classification models.

Reason To Use Logistic Regression with SMOTE Model

●Mitigates the problem of overfitting caused by random oversampling as synthetic examples are generated rather than replication of instances

●No loss of useful information



Reason To Reject Logistic Regression with SMOTE Model

●While generating synthetic examples SMOTE does not take into consideration neighboring examples from other classes.This can result in increase in overlapping of classes and can introduce additional noise

●SMOTE is not very effective for high dimensional data

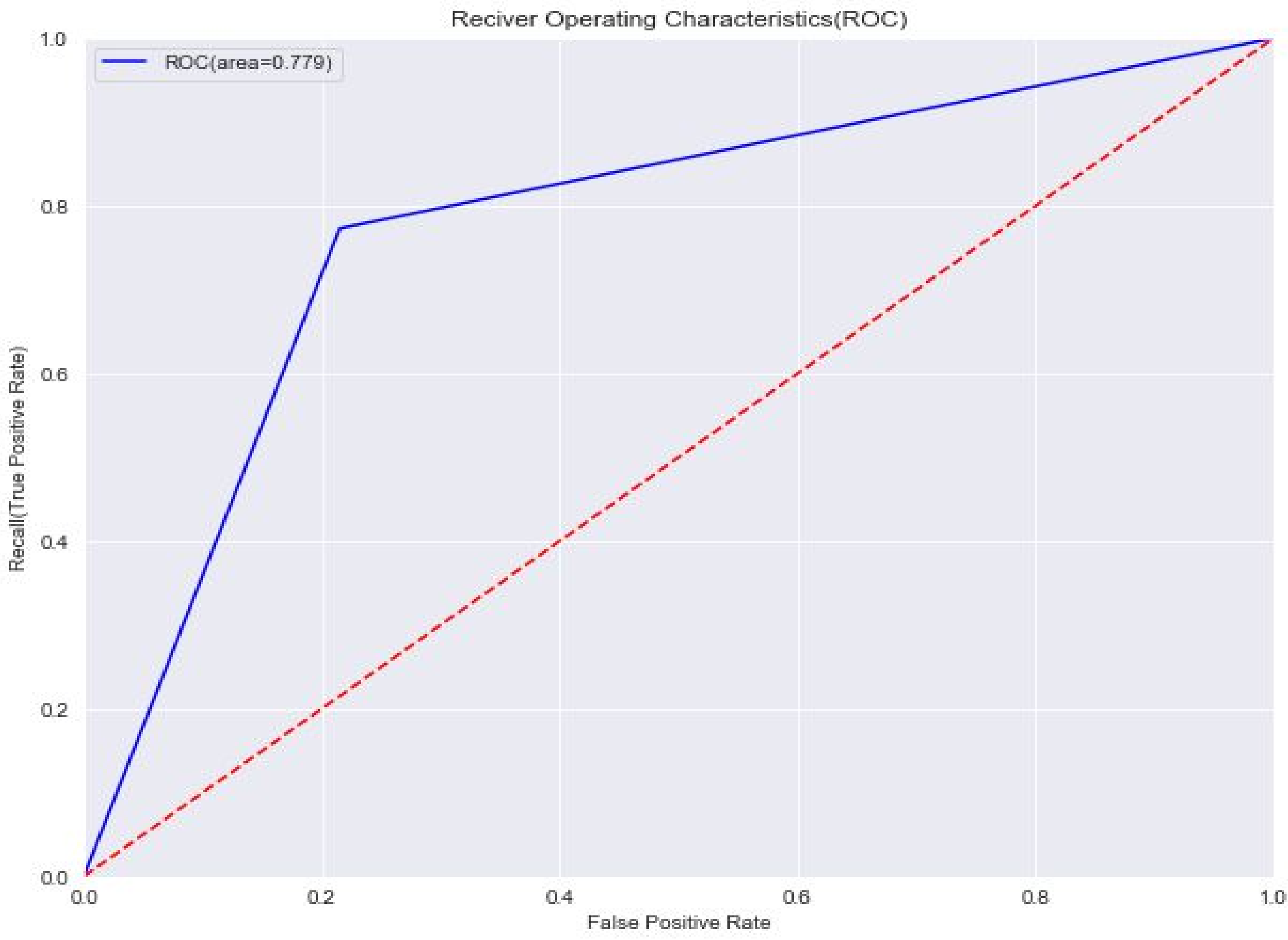
2.2.4 Light Gradient Boosting Machine

Light GBM is a fast, distributed, high-performance gradient boosting framework based on a decision tree algorithm, used for ranking, classification and many other machine learning tasks.

Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms. Also, it is surprisingly very fast, hence the word 'Light'.

Reason To use LGBM-

- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy than any other boosting algorithm
- Compatibility with Large Datasets
- Parallel learning supported


Reciver Operating Characteristics(ROC)
ROC(area=0.779)
Recall(True Positive Rate)
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

CHAPTER 3

CONCLUSION

3.1 MODEL EVALUATION

Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future.

Evaluating model performance with the data used for training is not acceptable in data science because it can easily generate overoptimistic and overfitted models.

There are two methods of evaluating models in data science, Hold-Out and Cross-Validation. To avoid overfitting, both methods use a test set (not seen by the model) to evaluate model performance.

Now that we have a few models for predicting the target variable, we need to decide which one to choose. There are several criteria that exist for evaluating and comparing models. We can compare the models using any of the following criteria:

1. Predictive Performance
2. Interpretability
3. Computational Efficiency

3.1.1 Precision

Precision is the ratio of system generated results that correctly *predicted positive* observations (**True Positives**) to the system's total *predicted positive* observations, both correct (**True Positives**) and incorrect (**False Positives**).

Model	**Logistic Regression**	**Logistic Regression With SMOTE**	**LGBM**
Precision in Python	0.671	0.272	0.477

Model	**Logistic Regression**	**Logistic Regression With SMOTE**	**LGBM**
Precision in R	0.923	1	0.434

3.1.2 Recall

Recall is the ratio of system generated results that correctly predicted positive observations (**True Positives**) to **all** observations in the actual malignant class (**Actual Positives**).

Model	**Logistic Regression**	**Logistic Regression With SMOTE**	**LGBM**
Recall in Python	0.27	0.771	0.626

Model	Logistic Regression	Logistic Regression With SMOTE	LGBM
Recall in R	0.267	1	0.645

3.1.3 AUC Score

The **Receiver Operator Characteristic (ROC)** curve is an evaluation metric for binary classification problems. It is a probability curve that plots the **TPR** against **FPR** at various threshold values and essentially **separates the 'signal' from the 'noise'**. The **Area Under the Curve (AUC)** is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

Model	Logistic Regression	Logistic Regression With SMOTE	LGBM
AUC Score in Python	0.628	0.778	0.777

Model	Logistic Regression	Logistic Regression With SMOTE	LGBM
AUC Score in R	0.617	1	0.767

3.1.4 F1 Score

The F1 Score is the weighted average (or harmonic mean) of Precision and Recall. Therefore, this score takes both **False Positives** and **False Negatives** into account to strike a balance between precision and Recall.

Model	Logistic Regression	Logistic Regression With SMOTE	LGBM
F1 Score in Python	0.385	0.402	0.514

Model	Logistic Regression	Logistic Regression With SMOTE	LGBM
F1 Score in R	0.414	1	0.518

3.2 MODEL SELECTION

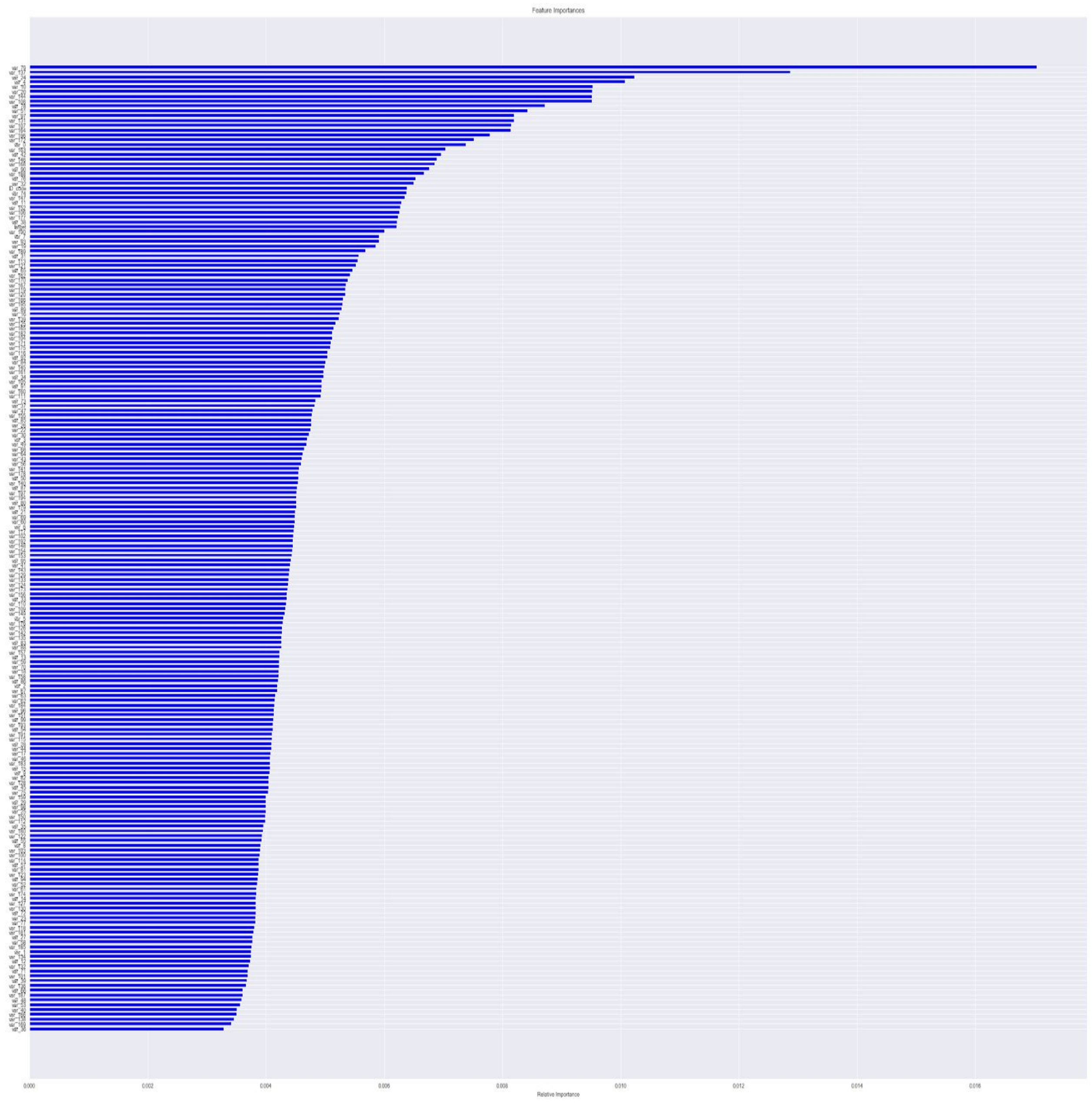
Model selection is the process of selecting one final machine learning model from among a collection of candidate machine learning models for a training dataset.

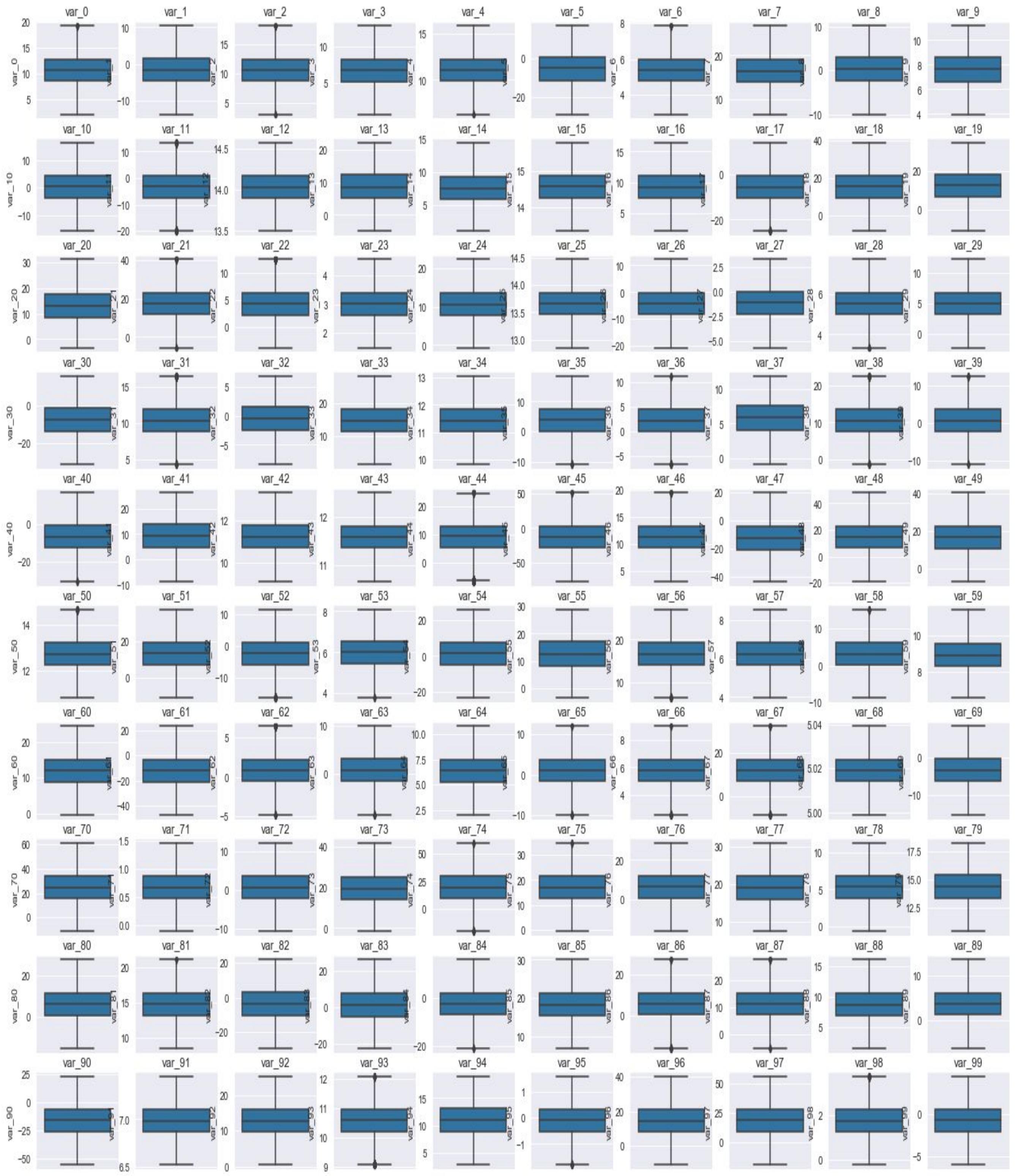
Model	Precision Score in Python	Recall Score in Python	AUC Score in Python	F1 Score in Python
Logistic Regression	0.671	0.269	0.627	0.384
Logistic Regression With SMOTE	0.272	0.772	0.778	0.403
LGBM	0.477	0.626	0.777	0.541

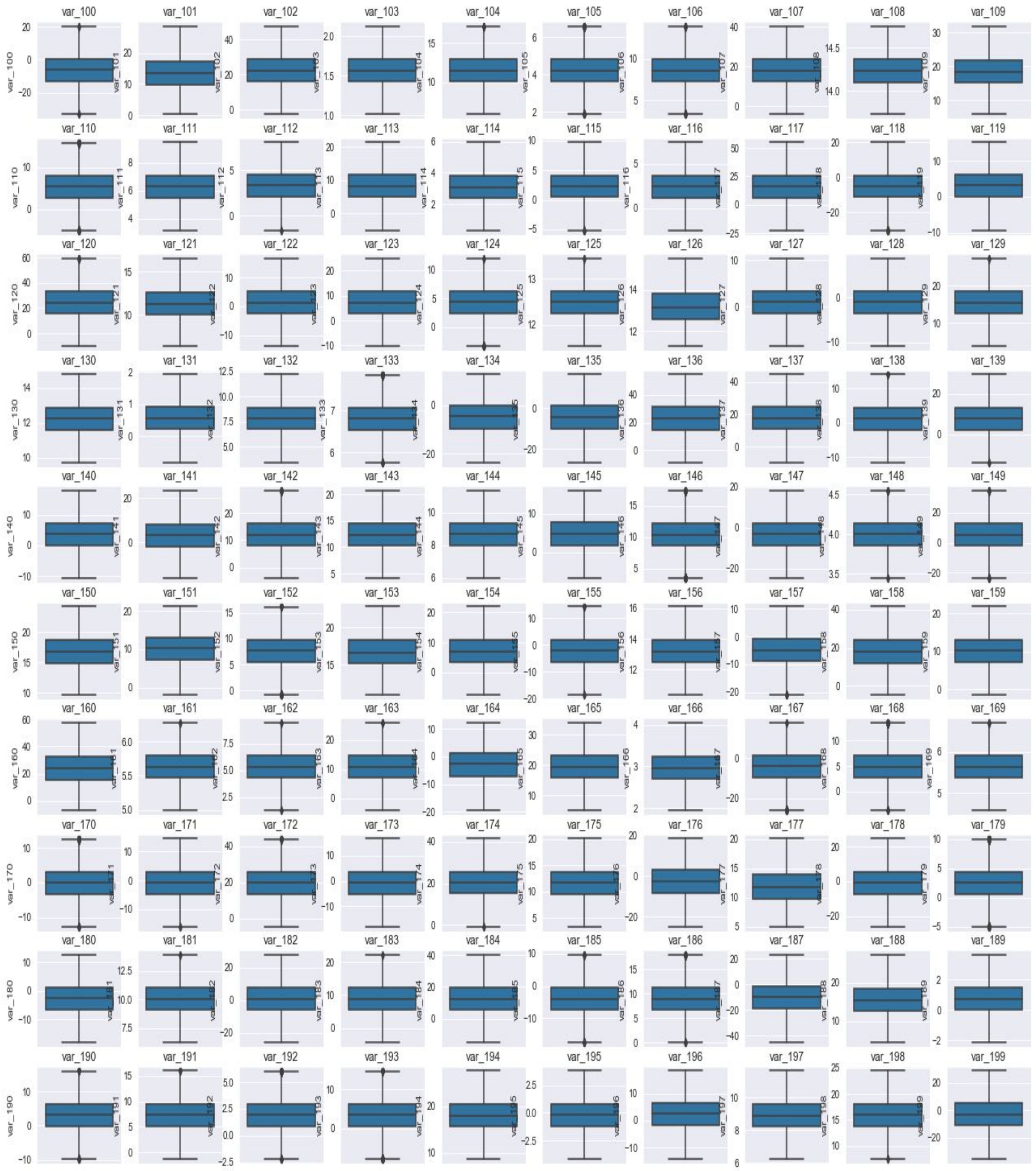
Model	Precision Score in R	Recall Score in R	AUC Score in R	F1 Score in R
Logistic Regression	0.923	0.267	0.617	0.414
Logistic Regression With SMOTE	1	1	1	1
LGBM	0.434	0.8645	0.767	0.518

We chose Light Gradient Boosting Machines (LGBM) basis on its AUC Score, Precision, Recall and F1 Score

APPENDIX A EXTRA FIGURES




var_0
var_1
var_2
var_3
var_4
var_5
var_6
var_7
var_8
var_9
var_10
var_11
var_12
var_13
var_14
var_15
var_16
var_17
var_18
var_19
var_20
var_21
var_22
var_23
var_24
var_25
var_26
var_27
var_28
var_29
var_30
var_31
var_32
var_33
var_34
var_35
var_36
var_37
var_38
var_39
var_40
var_41
var_42
var_43
var_44
var_45
var_46
var_47
var_48
var_49
var_50
var_51
var_52
var_53
var_54
var_55
var_56
var_57
var_58
var_59
var_60
var_61
var_62
var_63
var_64
var_65
var_66
var_67
var_68
var_69
var_70
var_71
var_72
var_73
var_74
var_75
var_76
var_77
var_78
var_79
var_80
var_81
var_82
var_83
var_84
var_85
var_86
var_87
var_88
var_89
var_90
var_91
var_92
var_93
var_94
var_95
var_96
var_97
var_98
var_99


var_100
var_101
var_102
var_103
var_104
var_105
var_106
var_107
var_108
var_109
var_110
var_111
var_112
var_113
var_114
var_115
var_116
var_117
var_118
var_119
var_120
var_121
var_122
var_123
var_124
var_125
var_126
var_127
var_128
var_129
var_130
var_131
var_132
var_133
var_134
var_135
var_136
var_137
var_138
var_139
var_140
var_141
var_142
var_143
var_144
var_145
var_146
var_147
var_148
var_149
var_150
var_151
var_152
var_153
var_154
var_155
var_156
var_157
var_158
var_159
var_160
var_161
var_162
var_163
var_164
var_165
var_166
var_167
var_168
var_169
var_170
var_171
var_172
var_173
var_174
var_175
var_176
var_177
var_178
var_179
var_180
var_181
var_182
var_183
var_184
var_185
var_186
var_187
var_188
var_189
var_190
var_191
var_192
var_193
var_194
var_195
var_196
var_197
var_198
var_199

APPENDIX B CODES

PYTHON CODES

Importing The Libraries

```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
#Machine Learning Algorithms
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import lightgbm as lgb
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.utils import resample
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split,cross_val_predict,cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score,auc,roc_auc_score,roc_curve,precisio
n_score,recall_score,f1_score
```

Setting UP Working Directory

```
os.chdir("C:/users/Raaz/Documents/Data Science/Santander Customer TransactionPrediction")

os.getcwd()
```

Importing The Dataset

```
train = pd.read_csv('train.csv')

test = pd.read_csv('test.csv')

train.head()

train.info()

train.describe()

test.head()

test.info()

test.describe()
```

Missing Value Analysis

```
Train_Missing_Values = pd.DataFrame(train.isna().sum(),columns=['No Of Missing Values'])

Train_Missing_Values

Test_Missing_Values = pd.DataFrame(test.isna().sum(),columns=['No Of Missing Values'])

Test_Missing_Values

print('No of Missing Value in Train Set:',Train_Missing_Values['No Of Missing Values'].sum())

print('No of Missing Value in Test Set:',Test_Missing_Values['No Of Missing Values'].sum())
```

Data Visualisation

```
plt.figure(figsize=(10,8))

sns.countplot(train['target'])

numerical_features = train.columns[2:102]

print('Distributions columns')

plt.figure(figsize=(22,18))

sns.set_style('darkgrid')

for i, col in enumerate(numerical_features):
  plt.subplot(10, 10, i + 1)
  sns.distplot(train[col])
  plt.title(col)
numerical_features = train.columns[102:]
print('Distributions columns')
plt.figure(figsize=(22,18))
for i, col in enumerate(numerical_features):
  plt.subplot(10, 10, i + 1)
  sns.distplot(train[col])
  plt.title(col)
numerical_features = test.columns[1:101]
print('Distributions columns')
plt.figure(figsize=(22,18))
for i, col in enumerate(numerical_features):
  plt.subplot(10, 10, i + 1)
  sns.distplot(test[col])
  plt.title(col)
numerical_features = test.columns[101:]
print('Distributions columns')
plt.figure(figsize=(22,18))
for i, col in enumerate(numerical_features):
  plt.subplot(10, 10, i + 1)
  sns.distplot(test[col])
  plt.title(col)
plt.figure(figsize=(16,8))
#Train Features-
train_features=train.columns.values[2:]

#Test Features-
```

```
test_features=test.columns.values[1:]

#Distribution plot for mean values per column in Train Features:
sns.distplot(train[train_features].mean(axis=0),color='red',kde=True,bins=100,label='Train')

#Distribution plot for mean values per column in Test Features:
sns.distplot(test[test_features].mean(axis=0),color='blue',kde=True,bins=100,label='Test')

plt.title('Distribution of Mean Values per Column in Train & Test Dataset')
plt.legend()
plt.show()
plt.figure(figsize=(16,8))

#Distribution plot for mean values per Row in Train Features:
sns.distplot(train[train_features].mean(axis=1),color='red',kde=True,bins=100,label='Train')

#Distribution plot for mean values per Row in Test Features:
sns.distplot(test[test_features].mean(axis=1),color='blue',kde=True,bins=100,label='Test')

plt.title('Distribution of Mean Values per Row in Train & Test Dataset')
plt.legend()
plt.show()

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

#Distribution plot for Standard Deviation per column in Train Features:
sns.distplot(train[train_features].std(axis=0),color='red',kde=True,bins=100,label='Train')

#Distribution plot for Standard Deviation per column in Test Features:
```

```
sns.distplot(test[test_features].std(axis=0),color='blue',kde=True,bins=100,label='Test')

plt.title('Distribution of Standard Deviation per Column in Train & Test Dataset')
plt.legend()
plt.show()
plt.figure(figsize=(16,8))

#Distribution plot for Standard Deviation per Row in Train Features:
sns.distplot(train[train_features].std(axis=1),color='red',kde=True,bins=100,label='Train')

#Distribution plot for Standard Deviation per Row in Test Featuress:
sns.distplot(test[test_features].std(axis=1),color='blue',kde=True,bins=100,label='Test')

plt.title('Distribution of Standard Deviation per Row in Train & Test Dataset')
plt.legend()
plt.show()
plt.figure(figsize=(16,8))

#Distribution plot for Skew Values per column in Train Features::
sns.distplot(train[train_features].skew(axis=0),color='red',kde=True,bins=100,label='Train')

#Distribution plot for Skew Values per column in Test Features::
sns.distplot(test[test_features].skew(axis=0),color='blue',kde=True,bins=100,label='Test')

plt.title('Distribution of Skew Values per Column in Train & Test Dataset')
plt.legend()
plt.show()
plt.figure(figsize=(16,8))
```

```
#Distribution plot for mean values per Row in Train Features::
sns.distplot(train[train_features].skew(axis=1),color='red',kde=True,bins=100,label='Train')

#Distribution plot for mean values per Row in Test Features::
sns.distplot(test[test_features].skew(axis=1),color='blue',kde=True,bins=100,label='Test')

plt.title('Distribution of Skew Values per Row in Train & Test Dataset')
plt.legend()
plt.show()
plt.figure(figsize=(16,8))

#Distribution plot for Kurtosis Values per column in Train Features:
sns.distplot(train[train_features].kurtosis(axis=0),color='red',kde=True,bins=100,label='Train
')

#Distribution plot for Kurtosis Values per column in Test Features:
sns.distplot(test[test_features].kurtosis(axis=0),color='blue',kde=True,bins=100,label='Test')

plt.title('Distribution of Kurtosis Values per Column in Train & Test Dataset')
plt.legend()
plt.show()
plt.figure(figsize=(16,8))

#Distribution plot for Kurtosis Values per Row in Train Features:
sns.distplot(train[train_features].kurtosis(axis=1),color='red',kde=True,bins=100,label='Train
')

#Distribution plot for Kurtosis Values per Row in Test Features:
sns.distplot(test[test_features].kurtosis(axis=1),color='blue',kde=True,bins=100,label='Test')
```

```
plt.title('Distribution of Kurtosis Values per Row in Train & Test Dataset')
plt.legend()
plt.show()
```

Feature Selection

```
train_corr = train.corr()
train_corr
test_corr = test.corr()
test_corr
plt.figure(figsize=(15,6))
sns.distplot(train_corr['target'])
train_corr['target'].nlargest(6)
train_corr['target'][1:].sort_values()
plt.figure(figsize=(10,8))
sns.heatmap(train_corr,cmap='viridis')
```

Outlier Analysis

```
#Train Set Outlier Analysis from var_0 to var_99
numerical_features = train.columns[2:102]
print('Boxplot of Features')
plt.figure(figsize=(22,18))
sns.set_style('darkgrid')
for i, col in enumerate(numerical_features):
  plt.subplot(10, 10, i + 1)
  sns.boxplot(y=train[col])
  plt.title(col)
#Train Set Outlier Analysis from var_100 to var_199
numerical_features = train.columns[102:]
print('Boxplot of Features')
```

```
plt.figure(figsize=(22,18))
sns.set_style('darkgrid')
for i, col in enumerate(numerical_features):
  plt.subplot(10, 10, i + 1)
  sns.boxplot(y=train[col])
  plt.title(col)
#Test Set Outlier Analysis from var_0 to var_99
numerical_features = test.columns[1:101]
print('Boxplot of Features')
plt.figure(figsize=(22,18))
sns.set_style('darkgrid')
for i, col in enumerate(numerical_features):
  plt.subplot(10, 10, i + 1)
  sns.boxplot(y=test[col])
  plt.title(col)
#Test Set Outlier Analysis from var_100 to var_199
numerical_features = test.columns[101:]
print('Boxplot of Features')
plt.figure(figsize=(22,18))
sns.set_style('darkgrid')
for i, col in enumerate(numerical_features):
  plt.subplot(10, 10, i + 1)
  sns.boxplot(y=test[col])
  plt.title(col)
```

Removing Outlier

```
columns = train.columns[2:]
for i in columns:
  q75,q25 = np.percentile(train.loc[:,i],[75,25])
  iqr = q75-q25
  min = q25 - (iqr*1.5)
  max = q75 + (iqr*1.5)
  train = train.drop(train[train.loc[:,i] < min].index)
  train = train.drop(train[train.loc[:,i] > max].index)
#Train Set Outlier Analysis from var_0 to var_99 after Removing Outliers
numerical_features = train.columns[2:102]
print('Boxplot of Features')
plt.figure(figsize=(22,18))
sns.set_style('darkgrid')
for i, col in enumerate(numerical_features):
  plt.subplot(10, 10, i + 1)
  sns.boxplot(y=train[col])
  plt.title(col)
#Train Set Outlier Analysis from var_100 to var_199 after Removing Outliers
numerical_features = train.columns[102:]
print('Boxplot of Features')
plt.figure(figsize=(22,18))
sns.set_style('darkgrid')
for i, col in enumerate(numerical_features):
  plt.subplot(10, 10, i + 1)
  sns.boxplot(y=train[col])
  plt.title(col)
```

Splitting The Dataset

```
X = train.drop(['ID_code','target'],axis=1)
y = train['target']
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Feature Scaling

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
```

Feature Importance

```
rfr = RandomForestClassifier(n_estimators = 5, criterion = 'entropy', random_state = 42)
rfr.fit(X_train,y_train)
features=train.columns
importances = rfr.feature_importances_
indices = np.argsort(importances)
plt.figure(figsize=(40,30))
plt.figure(1)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), features[indices])
plt.xlabel('Relative Importance')
```

Principle Component Analysis

```
pca = PCA().fit(X_train)
plt.figure(figsize=(10,8))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
```

Logistic Regression

```
#Hyperparameter Tunning
model = LogisticRegression()
solvers = ['newton-cg', 'lbfgs', 'liblinear']
```

```
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
# define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv,
scoring='accuracy',error_score=0)
grid_result = grid_search.fit(X_train, y_train)
# summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print("%f (%f) with: %r" % (mean, stdev, param))
lr = LogisticRegression(penalty='l2',C=0.1,solver='newton-cg')
lr.fit(X_train,y_train)
pred_lr = lr.predict(X_val)
print('Classification Report\n\n',classification_report(y_val,pred_lr))
print('\n')
print('Confusion matrix\n\n',confusion_matrix(y_val,pred_lr))
print('\n')
print('Accuracy : ',accuracy_score(y_val,pred_lr))
print('\n')
print('AUC Score : ',roc_auc_score(y_val,pred_lr))
print('\n')
print('Precision Score : ',precision_score(y_val,pred_lr))
print('\n')
print('Recall Score : ',recall_score(y_val,pred_lr))
print('\n')
print('F1 Score : ',f1_score(y_val,pred_lr))
print('\n')
print('Confusion matrix\n\n',confusion_matrix(y_val,pred_lr))
print('\nTrue Positives(TP) : ', confusion_matrix(y_val,pred_lr)[0,0])
print('\nTrue Negatives(TN) : ', confusion_matrix(y_val,pred_lr)[1,1])
print('\nFalse Positives(FP) : ', confusion_matrix(y_val,pred_lr)[0,1])
print('\nFalse Negatives(FN) : ', confusion_matrix(y_val,pred_lr)[1,0])
#ROC_AUC_Curve:-
sns.set_style('darkgrid')
plt.figure(figsize=(10,8))
false_positive_rate,recall,thresholds=roc_curve(y_val,pred_lr)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
```

Logistic Regression with Class Balancing(SMOTE)

SMOTE(Synthetic Minority Over-sampling Technique)

```
sm = SMOTE()
X_train_smote,y_train_smote = sm.fit_sample(X_train,y_train)
y_train_smote.value_counts()
```

```
lr = LogisticRegression()
lr.fit(X_train_smote,y_train_smote)
pred_lr_cb = lr.predict(X_val)
print('Classification Report\n\n',classification_report(y_val,pred_lr_cb))
print('\n')
print('Confusion matrix\n\n',confusion_matrix(y_val,pred_lr_cb))
print('\n')
print('Accuracy : ',accuracy_score(y_val,pred_lr_cb))
print('\n')
print('AUC Score : ',roc_auc_score(y_val,pred_lr_cb))
print('\n')
print('Precision Score : ',precision_score(y_val,pred_lr_cb))
print('\n')
print('Recall Score : ',recall_score(y_val,pred_lr_cb))
print('\n')
print('F1 Score : ',f1_score(y_val,pred_lr_cb))
print('\n')
print('Confusion matrix\n\n',confusion_matrix(y_val,pred_lr_cb))
print('\nTrue Positives(TP) : ', confusion_matrix(y_val,pred_lr_cb)[0,0])
print('\nTrue Negatives(TN) : ', confusion_matrix(y_val,pred_lr_cb)[1,1])
print('\nFalse Positives(FP) : ', confusion_matrix(y_val,pred_lr_cb)[0,1])
print('\nFalse Negatives(FN) : ', confusion_matrix(y_val,pred_lr_cb)[1,0])
#ROC_AUC_Curve:-
sns.set_style('darkgrid')
plt.figure(figsize=(10,8))
false_positive_rate,recall,thresholds=roc_curve(y_val,pred_lr_cb)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
```

LightBGM

```
#Training data-
lgb_train=lgb.Dataset(X_train,label=y_train)
lgb_val = lgb.Dataset(X_val,label=y_val)
params={'boosting_type': 'gbdt',
        'max_depth' : -1, #no limit for max_depth if <0
        'objective': 'binary',
        'boost_from_average':False,
        'nthread': 20,
        'metric':'auc',
        'num_leaves': 50,
        'learning_rate': 0.01,
        'max_bin': 100,      #default 255
        'subsample_for_bin': 100,
        'subsample': 1,
        'subsample_freq': 1,
        'colsample_bytree': 0.8,
        'bagging_fraction':0.5,
        'bagging_freq':5,
        'feature_fraction':0.08,
        'min_split_gain': 0.45, #>0
        'min_child_weight': 1,
        'min_child_samples': 5,
        'is_unbalance':True,
        }
num_round = 1000000
lgbm = lgb.train(params, lgb_train, num_round, valid_sets = [lgb_train, lgb_val],
              verbose_eval=1000, early_stopping_rounds = 2000)
lgb.plot_importance(lgbm,max_num_features=50,importance_type="split",figsize=(20,50))
#probability predictions
lgbm_predict_prob=lgbm.predict(X_val,random_state=42,num_iteration=lgbm.best_iteratio
n)
#Convert to binary output 1 or 0
pred_lgbm=np.where(lgbm_predict_prob>=0.5,1,0)
print('Classification Report\n\n',classification_report(y_val,pred_lgbm))
print('\n')
print('Confusion matrix\n\n',confusion_matrix(y_val,pred_lgbm))
print('\n')
print('Accuracy : ',accuracy_score(y_val,pred_lgbm))
print('\n')
print('AUC Score : ',roc_auc_score(y_val,pred_lgbm))
print('\n')
print('Precision Score : ',precision_score(y_val,pred_lgbm))
print('\n')
print('Recall Score : ',recall_score(y_val,pred_lgbm))
print('\n')
print('F1 Score : ',f1_score(y_val,pred_lgbm))
print('\n')
print('Confusion matrix\n\n',confusion_matrix(y_val,pred_lgbm))
print('\nTrue Positives(TP) : ', confusion_matrix(y_val,pred_lgbm)[0,0])
print('\nTrue Negatives(TN) : ', confusion_matrix(y_val,pred_lgbm)[1,1])
print('\nFalse Positives(FP) : ', confusion_matrix(y_val,pred_lgbm)[0,1])
print('\nFalse Negatives(FN) : ', confusion_matrix(y_val,pred_lgbm)[1,0])
#ROC_AUC_Curve:-
plt.figure(figsize=(10,8))
false_positive_rate,recall,thresholds=roc_curve(y_val,pred_lgbm)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1])
plt.ylim([0.0,1])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
```

Selecting Final Model

```
#After looking at the precision score,recall score,f1 score,auc score and accuracy
#it turned out that theLGBM is the best available model.
#Hence We will use the LGBM model to predict on the Test Dataset
```

Predicting on Test Set

```
test_set = scaler.transform(test.iloc[:,1:])
test['target'] = lgbm.predict(test_set,random_state=42,num_iteration=lgbm.best_iteration)
test['target']=np.where(test['target']>=0.5,1,0)
test['target'].value_counts()
test.head()
```

R CODES

```
#Clearing Objects:-
rm(list=ls(all=T))

#Importing the Libraries
library(tidyverse)
library(moments)
library(DataExplorer)
library(caret)
library(Matrix)
library(pdp)
library(mlbench)
library(caTools)
library(randomForest)
library(glmnet)
library(mlr)
library(vita)
library(rBayesianOptimization)
library(lightgbm)
library(pROC)
library(DMwR)
library(ROSE)
library(yardstick)
library(caTools)

#Setting Directory:-
```

```
setwd("C:/Users/Raaz/Documents/Data Science/Santander Customer
TransactionPrediction")



#Importing the training Data:-
train=read.csv("train.csv")
head(train)

#Importing the Test Data:-
test=read.csv("test.csv")
head(test)

#Dimension of the train data:-
dim(test)

#Summary of the train dataset:-
str(train)

#Dimension of the test data:-
dim(test)

#Summary of the test dataset:-
str(test)

#Target class count in train data:-
table(train$target)

#Missing Value Analysis:-
#Finding the missing values in train data
missing_val<-data.frame(missing_val=apply(train,2,function(x){sum(is.na(x))}))
```

```
missing_val<-sum(missing_val)

missing_val


#Finding the missing values in test data

missing_val<-data.frame(missing_val=apply(test,2,function(x){sum(is.na(x))}))

missing_val<-sum(missing_val)

missing_val


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

ggplot(train,aes(target))+theme_bw()+geom_bar(stat='count',fill='blue')


#Dataset is unbalanced data, where 90% of the data is no. of customers who will not
make a transaction & 10 % of  the data are those who will make a transaction.


#Distribution of train attributes from 3 to 102:-

for (var in names(train)[c(3:102)]){

  target<-train$target

  plot<-ggplot(train, aes(train[[var]])) +

           geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()

  print(plot)

}


#Distribution of train attributes from 103 to 202:-

for (var in names(train)[c(103:202)]){

  target<-train$target

  plot<-ggplot(train, aes(train[[var]])) +

           geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()

  print(plot)

}
```

```
#Distribution of test attributes from 2 to 101-
for (var in names(test)[c(2:101)]){
  plot<-ggplot(test, aes(test[[var]])) +
          geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()
  print(plot)
}



#Distribution of test attributes from 102 to 201:-
for (var in names(test)[c(102:201)]){
  plot<-ggplot(test, aes(test[[var]])) +
          geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()
  print(plot)
}

#Mean value per rows and columns in train & test dataset:-
#Applying the function to find mean values per row in train and test data.
train_mean<-apply(train[,-c(1,2)],MARGIN=1,FUN=mean)
test_mean<-apply(test[,-c(1)],MARGIN=1,FUN=mean)
ggplot()+
  #Distribution of mean values per row in train data

geom_density(data=train[,-c(1,2)],aes(x=train_mean),kernel='gaussian',show.legend=T
RUE,color='blue')+theme_classic()+
  #Distribution of mean values per row in test data

geom_density(data=test[,-c(1)],aes(x=test_mean),kernel='gaussian',show.legend=TRU
E,color='red')+
  labs(x='mean values per row',title="Distribution of mean values per row in train and
test dataset")
```

```
#Applying the function to find mean values per column in train and test data.
train_mean<-apply(train[,-c(1,2)],MARGIN=2,FUN=mean)
test_mean<-apply(test[,-c(1)],MARGIN=2,FUN=mean)
ggplot()+
  #Distribution of mean values per column in train data

geom_density(aes(x=train_mean),kernel='gaussian',show.legend=TRUE,color='blue')+
theme_classic()+
  #Distribution of mean values per column in test data
  geom_density(aes(x=test_mean),kernel='gaussian',show.legend=TRUE,color='red')+
  labs(x='mean values per column',title="Distribution of mean values per column in
train and test dataset")



#Applying the function to find standard deviation values per row in train and test
data.
train_sd<-apply(train[,-c(1,2)],MARGIN=1,FUN=sd)
test_sd<-apply(test[,-c(1)],MARGIN=1,FUN=sd)
ggplot()+
  #Distribution of standard deviation values per row in train data

geom_density(data=train[,-c(1,2)],aes(x=train_sd),kernel='gaussian',show.legend=TRU
E,color='red')+theme_classic()+
  #Distribution of standard deviation values per row in test data

geom_density(data=test[,-c(1)],aes(x=test_sd),kernel='gaussian',show.legend=TRUE,co
lor='blue')+
  labs(x='sd values per row',title="Distribution of sd values per row in train and test
dataset")


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

```
train_sd<-apply(train[,-c(1,2)],MARGIN=2,FUN=sd)
test_sd<-apply(test[,-c(1)],MARGIN=2,FUN=sd)
ggplot()+
  #Distribution of standard deviation values per column in train data

geom_density(aes(x=train_sd),kernel='gaussian',show.legend=TRUE,color='red')+the
me_classic()+
  #Distribution of standard deviation values per column in test data
  geom_density(aes(x=test_sd),kernel='gaussian',show.legend=TRUE,color='blue')+
  labs(x='sd values per column',title="Distribution of std values per column in train
and test dataset")

#Applying the function to find skewness values per row in train and test data.
train_skew<-apply(train[,-c(1,2)],MARGIN=1,FUN=skewness)
test_skew<-apply(test[,-c(1)],MARGIN=1,FUN=skewness)
ggplot()+
  #Distribution of skewness values per row in train data

geom_density(aes(x=train_skew),kernel='gaussian',show.legend=TRUE,color='red')+th
eme_classic()+
  #Distribution of skewness values per column in test data
  geom_density(aes(x=test_skew),kernel='gaussian',show.legend=TRUE,color='blue')+
  labs(x='skewness values per row',title="Distribution of skewness values per row in
train and test dataset")

#Applying the function to find skewness values per column in train and test data.
train_skew<-apply(train[,-c(1,2)],MARGIN=2,FUN=skewness)
test_skew<-apply(test[,-c(1)],MARGIN=2,FUN=skewness)
ggplot()+
  #Distribution of skewness values per column in train data
```

```
geom_density(aes(x=train_skew),kernel='gaussian',show.legend=TRUE,color='red')+th
eme_classic()+
  #Distribution of skewness values per column in test data
  geom_density(aes(x=test_skew),kernel='gaussian',show.legend=TRUE,color='blue')+
  labs(x='skewness values per column',title="Distribution of skewness values per
column in train and test dataset")

#Applying the function to find kurtosis values per row in train and test data.
train_kurtosis<-apply(train[,-c(1,2)],MARGIN=1,FUN=kurtosis)
test_kurtosis<-apply(test[,-c(1)],MARGIN=1,FUN=kurtosis)
ggplot()+
  #Distribution of kurtosis values per row in train data

geom_density(aes(x=train_kurtosis),kernel='gaussian',show.legend=TRUE,color='blue'
)+theme_classic()+
  #Distribution of kurtosis values per row in test data

geom_density(aes(x=test_kurtosis),kernel='gaussian',show.legend=TRUE,color='red')+
  labs(x='kurtosis values per row',title="Distribution of kurtosis values per row in train
and test dataset")

#Applying the function to find kurtosis values per column in train and test data.
train_kurtosis<-apply(train[,-c(1,2)],MARGIN=2,FUN=kurtosis)
test_kurtosis<-apply(test[,-c(1)],MARGIN=2,FUN=kurtosis)
ggplot()+
  #Distribution of kurtosis values per column in train data

geom_density(aes(x=train_kurtosis),kernel='gaussian',show.legend=TRUE,color='blue'
)+theme_classic()+
  #Distribution of kurtosis values per column in test data
```

```
geom_density(aes(x=test_kurtosis),kernel='gaussian',show.legend=TRUE,color='red')+
  labs(x='kurtosis values per column',title="Distribution of kurtosis values per column in train and test dataset")

#Correlations in train data:-
#convert factor to int
train$target<-as.numeric(train$target)
train_correlation<-cor(train[,c(2:202)])
train_correlation
#Observation:- We can observe that correlation between train attributes is very small.

#Correlations in test data
test_correlation<-cor(test[,c(2:201)])
test_correlation
#Observation:- We can observe that correlation between test attributes is very small.

#Outlier Analysis
#BOx Plot of train attributes from 3 to 202:-
for (var in names(train)[c(3:202)]){
  target<-train$target
  plot<-ggplot(train, aes(train[[var]])) +
          geom_boxplot() + ggtitle(var)+theme_classic()
  print(plot)
}

#BOx Plot of test attributes from 2 to 201:-
for (var in names(test)[c(2:201)]){
  target<-test$target
  plot<-ggplot(test, aes(test[[var]])) +
          geom_boxplot() + ggtitle(var)+theme_classic()
  print(plot)
}
```

#Observation:- We can observe that there are outliers present in both of train and test set

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

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

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

```
#Split the training data using caTools
set.seed(123)
split = sample.split(train$target, SplitRatio = 0.80)
train_data = subset(train, split == TRUE)
valid_data = subset(train, split == FALSE)

#dimension of train and validation data
dim(train_data)
dim(valid_data)
```

#Random forest classifier:-

```
#Training the Random forest classifier
set.seed(2732)
#convert to int to factor
train_data$target<-as.factor(train_data$target)
```

```
#setting the mtry
mtry<-floor(sqrt(200))
#setting the tunegrid
tuneGrid<-expand.grid(.mtry=mtry)
#fitting the ranndom forest
rf<-randomForest(target~.,train_data[,-c(1)],mtry=mtry,ntree=10,importance=TRUE)

#Feature importance by random forest-
#Variable importance
VarImp<-importance(rf,type=2)
VarImp
```

#Observation:-We can observed that the top important features are var_12, var_26, var_22,v var_174, var_198 and so on based on Mean decrease gini.

#Partial dependence plots:-PDP gives a graphical depiction of marginal effect of a variable on the class probability or classification. It shows how a feature effects predictions.

#Calculation of partial dependence plots on random forest:-

#we are observing impact of main features which are discovered in previous section by using PDP Plot.

```
#We will plot "var_13"
par.var_13 <- partial(rf, pred.var = c("var_13"), chull = TRUE)
plot.var_13 <- autoplot(par.var_13, contour = TRUE)
plot.var_13

#We will plot "var_6"
par.var_6 <- partial(rf, pred.var = c("var_6"), chull = TRUE)
```

```
plot.var_6 <- autoplot(par.var_6, contour = TRUE)
plot.var_6

#Handling of imbalanced data- Now we are going to explore 5 different approaches
for dealing with imbalanced datasets.
#Change the performance metric
#Oversample minority class
#Undersample majority class
#ROSE
#LightGBM

#Logistic Regression Model:-
#Split the data using simple random sampling:-
set.seed(689)
split = sample.split(train$target, SplitRatio = 0.80)
train.data = subset(train, split == TRUE)
valid.data = subset(train, split == FALSE)
#dimension of train data
dim(train.data)
#dimension of validation data
dim(valid.data)
#target classes in train data
table(train.data$target)
#target classes in validation data
table(valid.data$target)

#Training and validation dataset

#Training dataset
X_t<-as.matrix(train.data[,-c(1,2)])
```

```
y_t<-as.matrix(train.data$target)
#validation dataset
X_v<-as.matrix(valid.data[,-c(1,2)])
y_v<-as.matrix(valid.data$target)
#test dataset
test<-as.matrix(test[,-c(1)])

#Logistic regression model
set.seed(667) # to reproduce results
lr_model <-glmnet(X_t,y_t, family = "binomial")
summary(lr_model)

#Cross validation prediction
set.seed(8909)
cv_lr <- cv.glmnet(X_t,y_t,family = "binomial", type.measure = "class")
cv_lr

#Plotting the missclassification error vs log(lambda) where lambda is regularization
parameter
#Minimum lambda
cv_lr$lambda.min
#plot the auc score vs log(lambda)
plot(cv_lr)

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

#Model performance on validation dataset
set.seed(5363)
cv_predict.lr<-predict(cv_lr,X_v,s = "lambda.min", type = "class")
```

cv_predict.lr

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

#Confusion Matrix:-

set.seed(689)

#actual target variable

target<-valid.data$target

#convert to factor

target<-as.factor(target)

#predicted target variable

#convert to factor

cv_predict.lr<-as.factor(cv_predict.lr)

confusionMatrix(data=cv_predict.lr,reference=target)

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

#ROC_AUC score and curve

set.seed(892)

cv_predict.lr<-as.numeric(cv_predict.lr)

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

#Oversample Minority Class:-

#-Adding more copies of minority class.

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

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

#Undersample Mojorityclass:-

#-Removing some copies of majority class.

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

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

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

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

```
#Random Oversampling Examples(ROSE)
set.seed(699)
train.rose <- ROSE(target~., data =train.data[,-c(1)],seed=32)$data
#target classes in balanced train data
table(train.rose$target)
valid.rose <- ROSE(target~., data =valid.data[,-c(1)],seed=42)$data
#target classes in balanced valid data
table(valid.rose$target)

#Logistic regression model
set.seed(462)
lr_rose <-glmnet(as.matrix(train.rose),as.matrix(train.rose$target), family = "binomial")
summary(lr_rose)

#Cross validation prediction
set.seed(473)
cv_rose = cv.glmnet(as.matrix(valid.rose),as.matrix(valid.rose$target),family =
"binomial", type.measure = "class")
```

```
cv_rose

#Plotting the missclassification error vs log(lambda) where lambda is regularization parameter:-
#Minimum lambda
cv_rose$lambda.min
#plot the auc score vs log(lambda)
plot(cv_rose)

#Model performance on validation dataset
set.seed(442)
cv_predict.rose<-predict(cv_rose,as.matrix(valid.rose),s = "lambda.min", type = "class")
cv_predict.rose

#Confusion matrix
set.seed(478)
#actual target variable
target<-valid.rose$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.rose<-as.factor(cv_predict.rose)
#Confusion matrix
confusionMatrix(data=cv_predict.rose,reference=target)

#ROC_AUC score and curve
set.seed(843)
#convert to numeric
cv_predict.rose<-as.numeric(cv_predict.rose)
```

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

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

#Training and validation dataset

#Convert data frame to matrix
set.seed(5432)
X_train<-as.matrix(train.data[,-c(1,2)])
y_train<-as.matrix(train.data$target)
X_valid<-as.matrix(valid.data[,-c(1,2)])
y_valid<-as.matrix(valid.data$target)
test_data<-as.matrix(test[,-c(1)])
head(test_data)
#training data
lgb.train <- lgb.Dataset(data=X_train, label=y_train)
#Validation data
lgb.valid <- lgb.Dataset(data=X_valid,label=y_valid)

#Choosing best hyperparameters

#Selecting best hyperparameters
set.seed(653)
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)

#Training the lgbm model

set.seed(7663)
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
set.seed(6532)
lgbm_pred_prob <- predict(lgbm.model,test_data)
print(lgbm_pred_prob)
#Convert to binary output (1 and 0) with threshold 0.5
lgbm_pred<-ifelse(lgbm_pred_prob>0.5,1,0)
print(lgbm_pred)
```

```
#Let us plot the important features
set.seed(6521)
#feature importance plot
tree_imp <- lgb.importance(lgbm.model, percentage = TRUE)
lgb.plot.importance(tree_imp, top_n = 50, measure = "Frequency", left_margin = 10)
```

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

```
#Final submission
sub_df<-data.frame(ID_code=test$ID_code,lgb_predict_prob=lgbm_pred_prob,lgb_p
redict=lgbm_pred)
write.csv(sub_df,'submission-R.CSV',row.names=F)
head(sub_df)
```

References

James, Gareth, Daniela Witten, Trevor Hastie, and Robert Tibshirani. 2013. An Introduction to Statistical Learning. Vol. 6. Springer. Wickham, Hadley. 2009. Ggplot2: Elegant Graphics for Data Analysis. Springer Science & Business Media.