

Prediction of Santander Customer Transaction

Vanusha Baregal

11/22/19



Contents

1 Introduction	
1.1 Problem Statement	2
1.2 Data	2
2 Methodology	
2.1 Exploratory Data Analysis	4
2.1.2 Missing value analysis	11
2.1.1 Attributes Distributions and trends	6
2.1.3 Outlier Analysis	11
2.1.4 Feature Selection	14
2.1.5 Feature Engineering	16
2.2 Modeling	18
2.2.1 Model Selection	18
2.2.2 Logistic Regression	18
2.2.3 SMOTE or ROSE	22
2.2.4 LightGBM & XGboost.....	23
3 Conclusion	
3.1 Model Evaluation	28
3.1.1 Confusion Matrix	29
3.1.2 ROC_AUC_score	32
3.2 Model Selection	41
Appendix – Complete Python and R Code	
Python Code	
R code	
References	

Chapter 1

Introduction

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.

Santander Bank's data science team wants to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted. The bank is continually challenging its machine learning algorithms to make sure they can more accurately identify new ways to solve its most common challenges such as:

- Is a customer satisfied?
- Will a customer buy this product?
- Can a customer pay this loan?

1.1 Problem Statement

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

1.2 Data

Our task is to build Classification model to predict whether the customer will make the transaction in the future. Given below is a sample of Santander Customer Transaction data set:

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

	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
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...	4.4354	3.9642	3.1364	1.6910	18.5227	-2.3978	7.8784
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	...	7.6421	7.7214	2.5837	10.9516	15.4305	2.0339	8.1267
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	...	2.9057	9.7905	1.6704	1.6858	21.6042	3.1417	-6.5213
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	...	4.4666	4.7433	0.7178	1.4214	23.0347	-1.2706	-2.9275
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	...	-1.4905	9.5214	-0.1508	9.1942	13.2876	-1.5121	3.9267
5 rows × 202 columns																		

Table 1.2

	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
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	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
2	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	...	-0.7484	10.9935	1.9803	2.1800	12.9813	2.1281	-7.1086
3	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	...	9.5702	9.0766	1.6580	3.5813	15.1874	3.1656	3.9567
4	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	...	4.2259	9.1723	1.2835	3.3778	19.5542	-0.2860	-5.1612
5 rows × 201 columns																		

In the train dataset there are 200000 observations and 202 variables using which we have to predict the customer transaction and the test data has 200000 observations and 201 variables

train contains:

- ID_code (string)
- target
- 200 numerical variables, named from var_0 to var_199

test contains:

- ID_code (string)
- 200 numerical variables, named from var_0 to var_199

Chapter 2

Methodology

2.1 Pre Processing

Any predictive modeling requires that we look at the data before we start modeling. However, in data mining terms looking at data refers to so much more than just looking. Looking at data refers to exploring the data, cleaning the data as well as visualizing the data through graphs and plots. This is often called as Exploratory Data Analysis. Most analysis like regression, require the data to be normally distributed. We can visualize that in a glance by looking at the probability distributions or probability density functions of the variable. To start this process, we will first try and look at all the probability distributions of the variables.

Class distribution

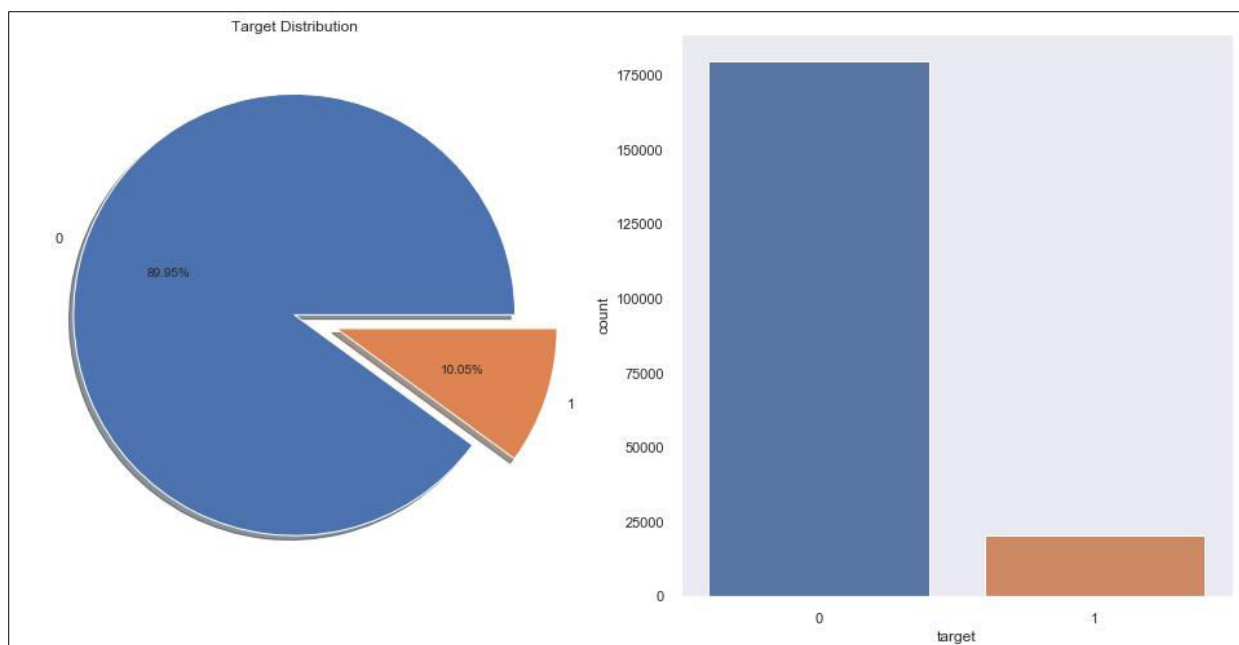


Fig 2.1 Distribution of dependent variable “target”

Findings:

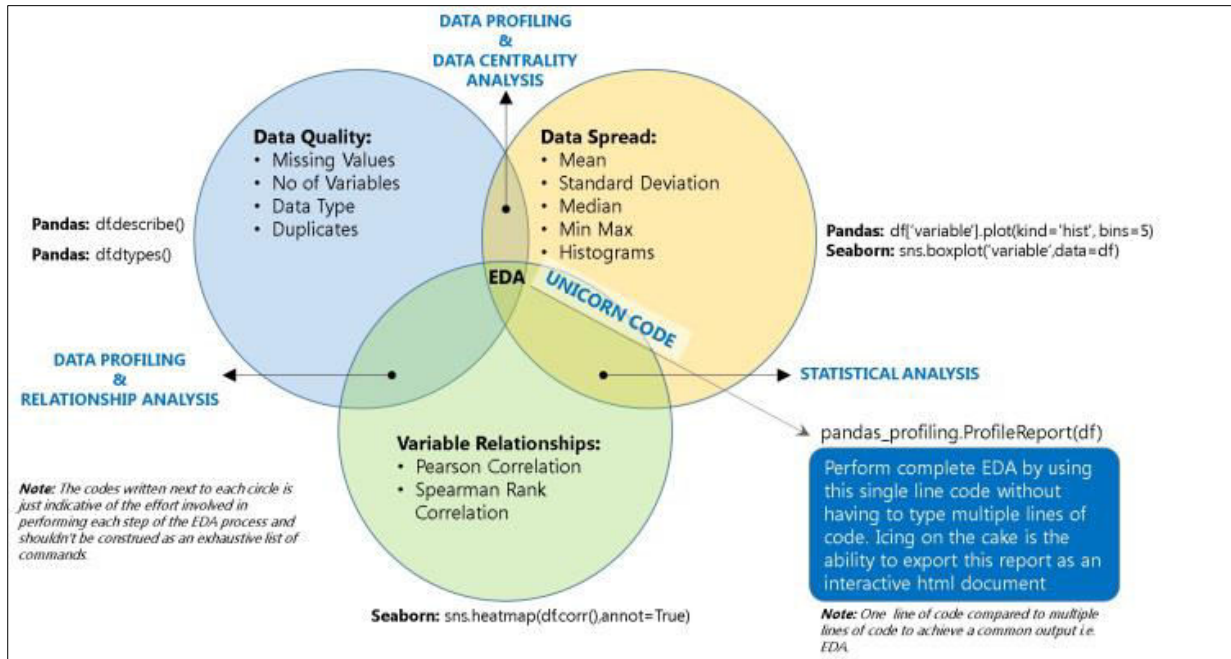
- Target variable has imbalanced class distribution where the number of observations belonging to one class is significantly lower than those belonging to the other class.
- Above visualisation shows that **89.95%** of customers who will not make the transaction (belong to "0") and only **10.05%** of customers make the transaction (belong to "1") at Santander.

Python Pandas Profiling

The pandas-profiling Python package is a great tool to create HTML profiling reports. For a given dataset, it computes the following statistics:

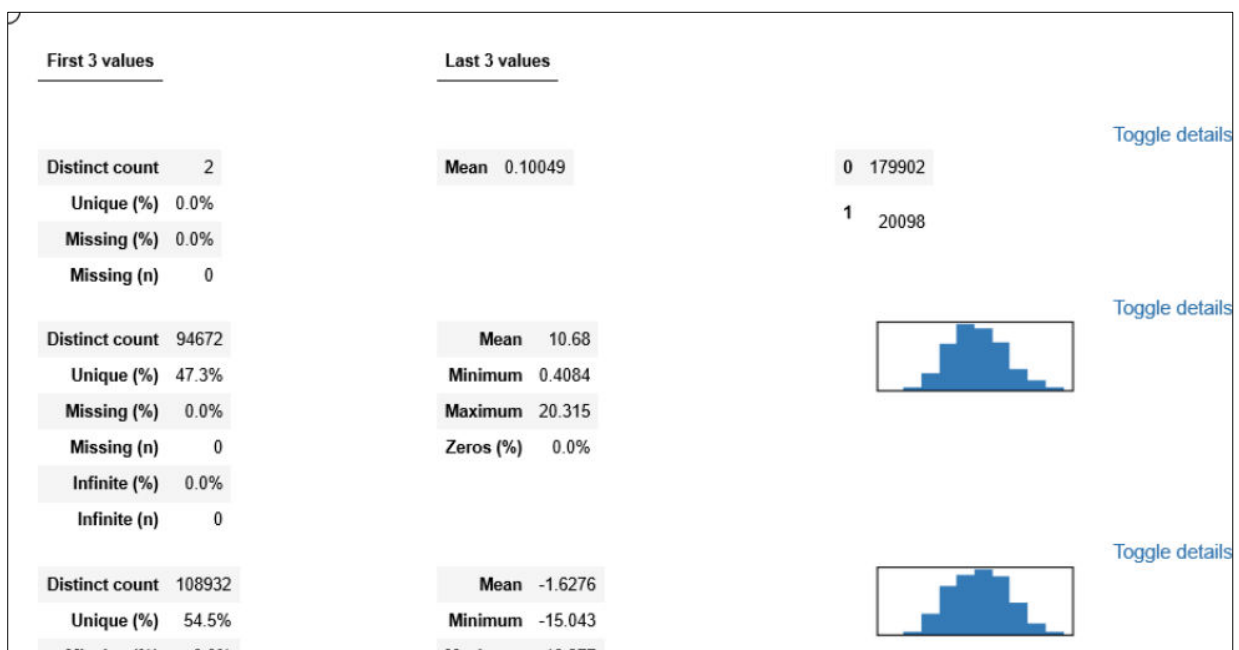
- Essentials: type, unique values, missing values.

- Quantile statistics like minimum value, Q1, median, Q3, maximum, range, interquartile range.
- Descriptive statistics like mean, mode, standard deviation, sum, median absolute deviation, coefficient of variation, kurtosis, skewness.
- Most frequent values.
- Histogram.
- Correlations highlighting of highly correlated variables, Spearman and Pearson matrixes.



Source: zone.com/articles/pandas-one-line-magical-code-for-eda-pandas-profil

Output of Pandas Profiling in Python



Out[4]:

Overview

Dataset info

Number of variables	202
Number of observations	200000
Total Missing (%)	0.0%
Total size in memory	308.2 MiB
Average record size in memory	1.6 KiB

Variables types

Numeric	200
Categorical	0
Boolean	1
Date	0
Text (Unique)	1
Rejected	0
Unsupported	0

Variables

ID_code

Categorical, Unique

target

Boolean

First 3 values

Distinct count	2
Unique (%)	0.0%
Missing (%)	0.0%
Missing (n)	0

Last 3 values

Mean	0.10049
0	179902
1	20098

Below table gives us glance on highly correlated variables

Correlations

Sample

	ID_code	target	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
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.7470	2.9252	3.1821	14.0137	0.5745	8.7989	14.5691
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	3.1468	8.0851	-0.4032	8.0585	14.0239	8.4135	5.4345	13.7003
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.9525	-0.3249	-11.2648	14.1929	7.3124	7.5244	14.6472
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.2450	2.3061	2.8102	13.8463	11.9704	6.4569	14.8372
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.6784	-9.4458	-12.1419	13.8481	7.8895	7.7894	15.0553
<																		

Distribution of Predictor variables (train dataset)

Train dataset consists of one string variable- Id_code, target variable and 200 numerical variables. The distribution of predictor variables is plotted below in two sets (1-99) and (100-199). Figures 2.2 & 2.3 (Appendix) mapped in Python. The distributions of many variables can be seen either very closely, or somewhat imitating the normal distribution.

Findings:

- As observed from below fig 2.2 & 2.3 it can be seen that considerable number of variables are following same distribution for both the classes of target variables like var_3, var_4, var_10, var_11, var_171, var_185 etc.
- Variables like var_0, var_1, var_2, var_9, var_180, var_98 etc. followed different distribution for both the classes of target.



Fig 2.2 Distribution plots of predictor variables (var_0 to var_99)

Distribution of Predictor variables (test dataset)

Test dataset consists of one string model- Id_code and 200 numerical variables. The distribution of test variables is plotted in python as shown in below fig 2.4.

Findings:

- Variables like var_4, var_11, var_12 can be seen following normal distribution
- Variables like var_2, var_3, var_13 etc. can be seen having skewed distribution from the distribution plots



Fig 2.4 Distribution of predictor variables from test dataset

Distribution of mean values in both train and test dataset

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

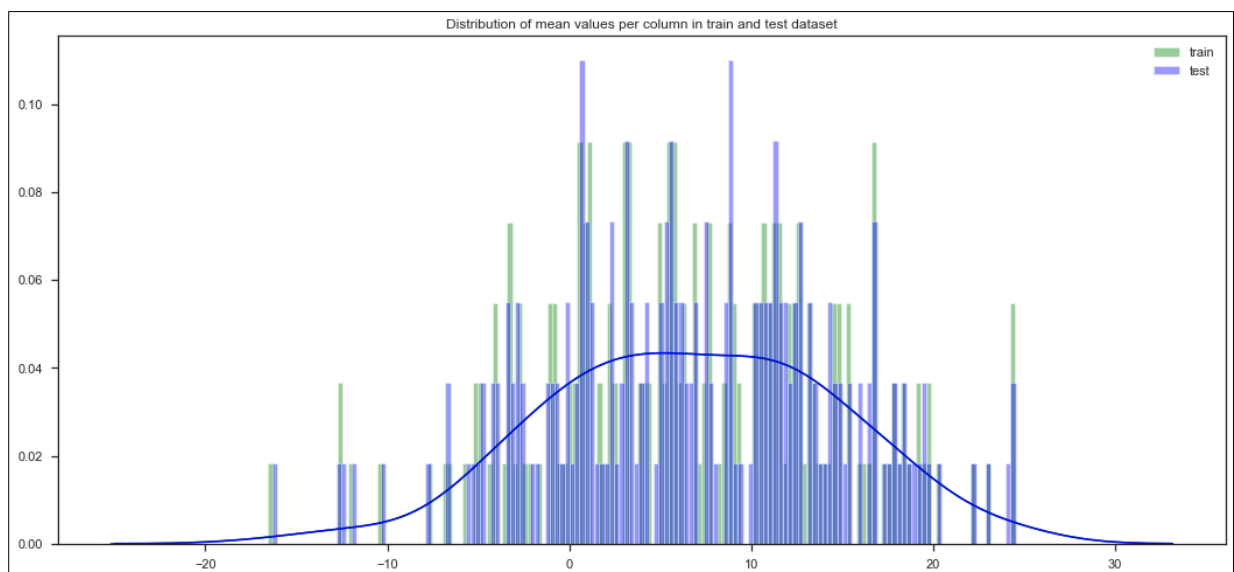


Fig 2.5 Distribution of mean values per column in train and test dataset

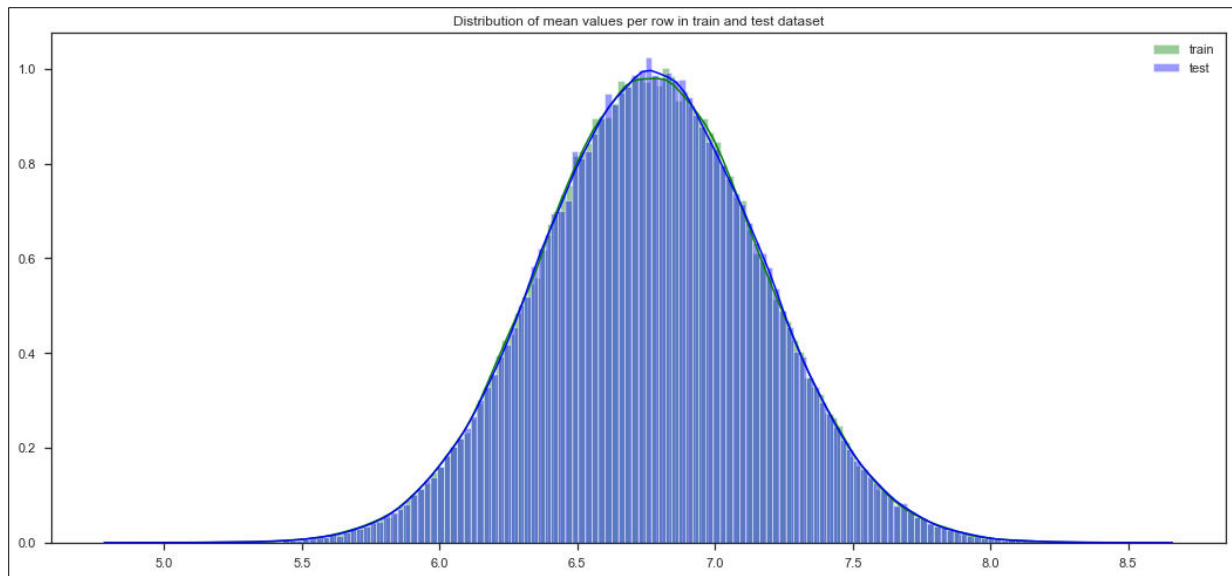


Fig 2.6 Distribution of mean values per row in train and test dataset

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

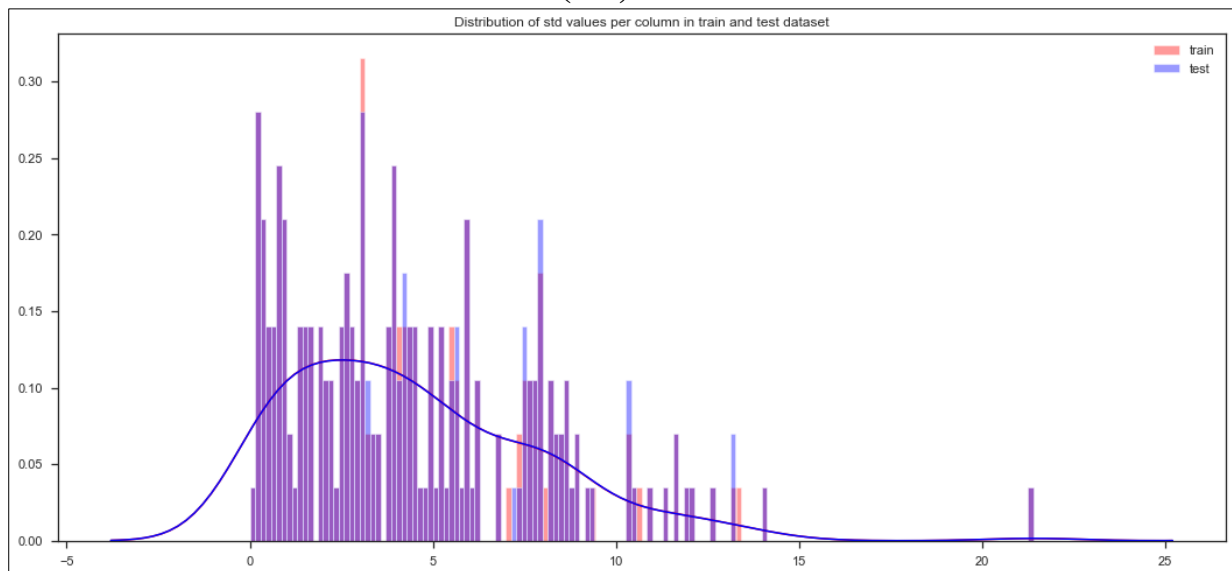


Fig 2.7 Distribution of Standard deviation values per column in train and test dataset

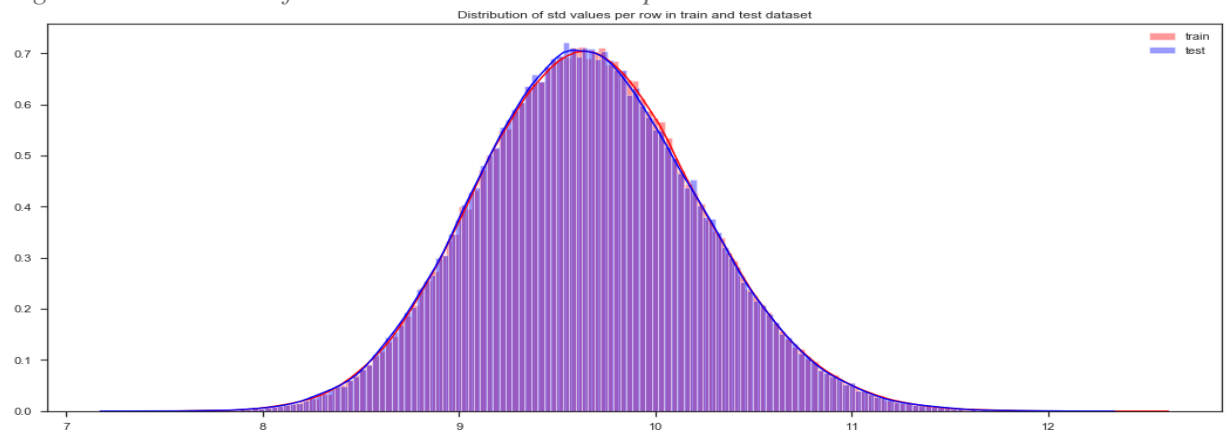


Fig 2.8 Distribution of Standard deviation values per row in train and test dataset

Distribution of skewness values in train and test dataset

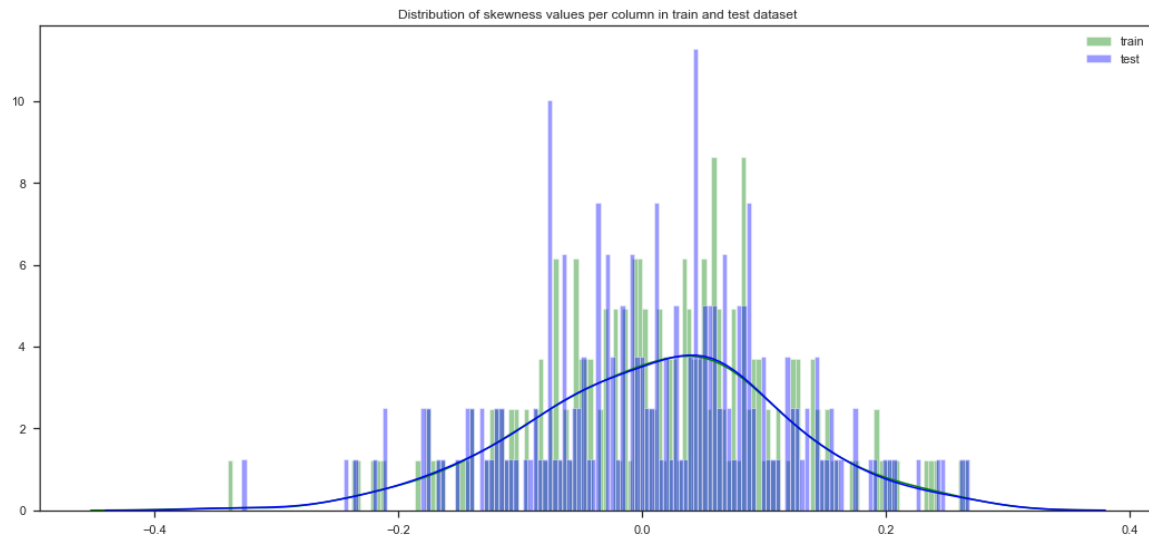


Fig 2.9 Distribution of Skewness values per column in train and test dataset

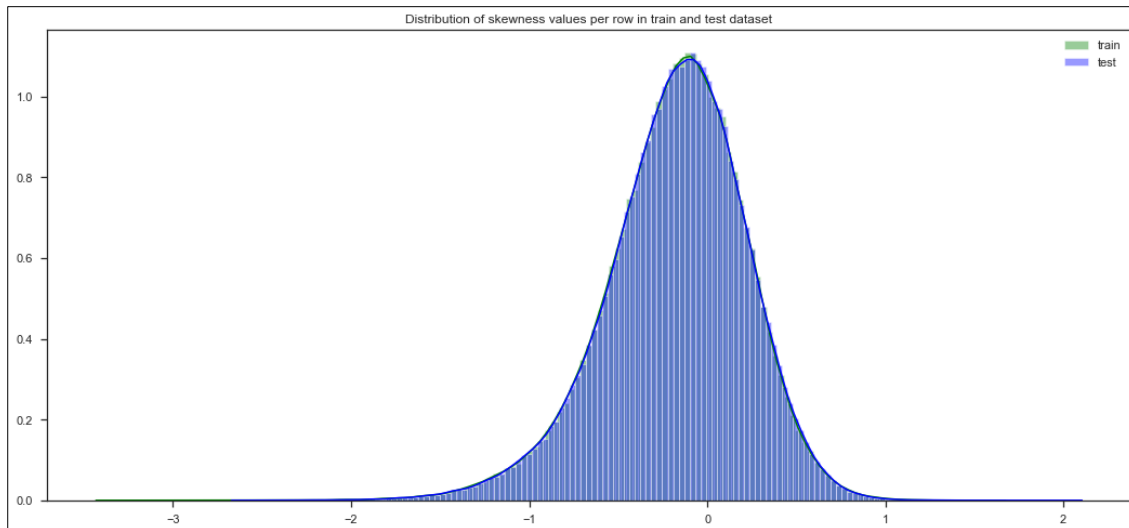


Fig 2.10 Distribution of Skewness values per row in train and test dataset

Distribution of kurtosis values in train and test dataset

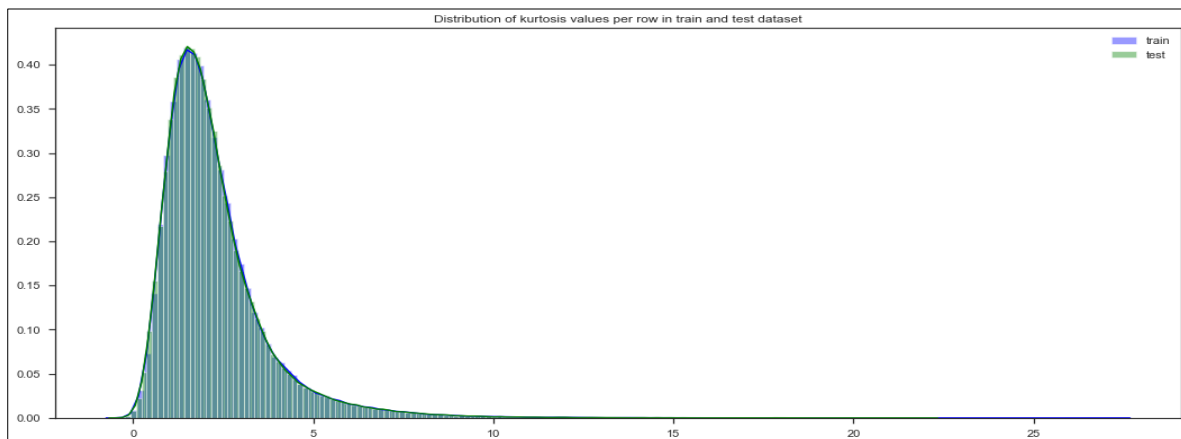


Figure 2.11 Distribution of Kurtosis values per row in train and test dataset

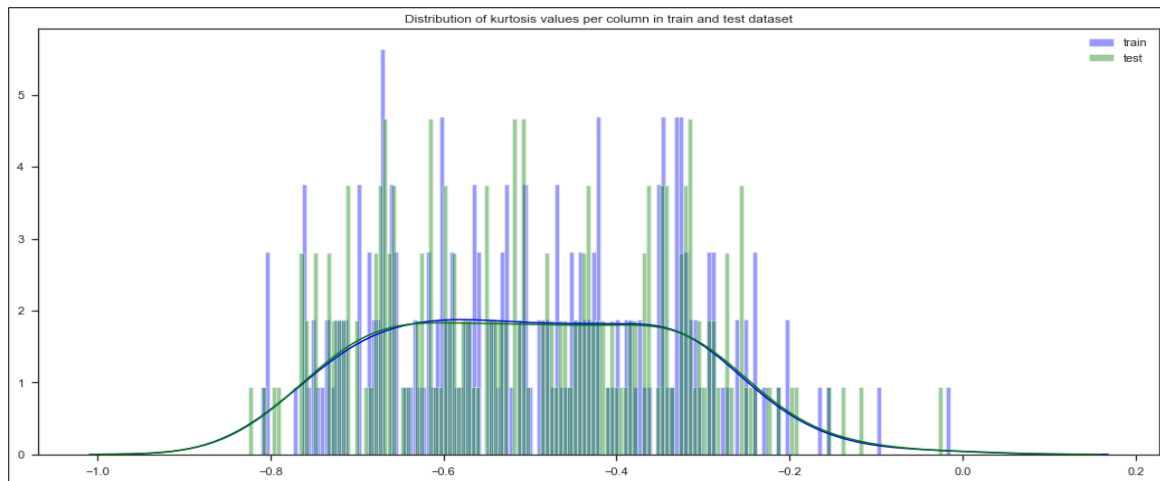


Figure 2.12 Distribution of Kurtosis values per column in train and test dataset

2.1.1 Missing value analysis

In statistics, **missing data**, or **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. If missing values are present in the data, then according to the percentage of missing we can either delete the variable or impute the values using mean, median and KNN imputation method.

Missing value analysis is done on both train and test dataset in this project. After the analysis, it was found that there were no missing values in both train and test data.

Missing value analysis in R and Python is pasted below:

```
# R code
missing_val<-data.frame(missing_val=apply(train,2,function(x){sum(is.na(x))}))
missing_val<-sum(missing_val)
[1] 0
missing_val<-data.frame(missing_val=apply(test,2,function(x){sum(is.na(x))}))
missing_val<-sum(missing_val)
[1] 0

#Python code

missingvalue_train=pd.DataFrame(train.isnull().sum())
missingvalue_test=pd.DataFrame(test.isnull().sum())

print (missingvalue_train)
print (missingvalue_test)
```

2.1.2 Outlier Analysis

Outliers are extreme values that deviate from other observations on data, they may indicate a variability in a measurement, experimental errors or a novelty. In other words, an outlier is an observation that diverges from an overall pattern on a sample.

Outlier analysis on train data using box plots is shown in below fig 2.12. for only (var_0 to var_49) variables. Boxplots of remaining variables will be attached in the Appendix.

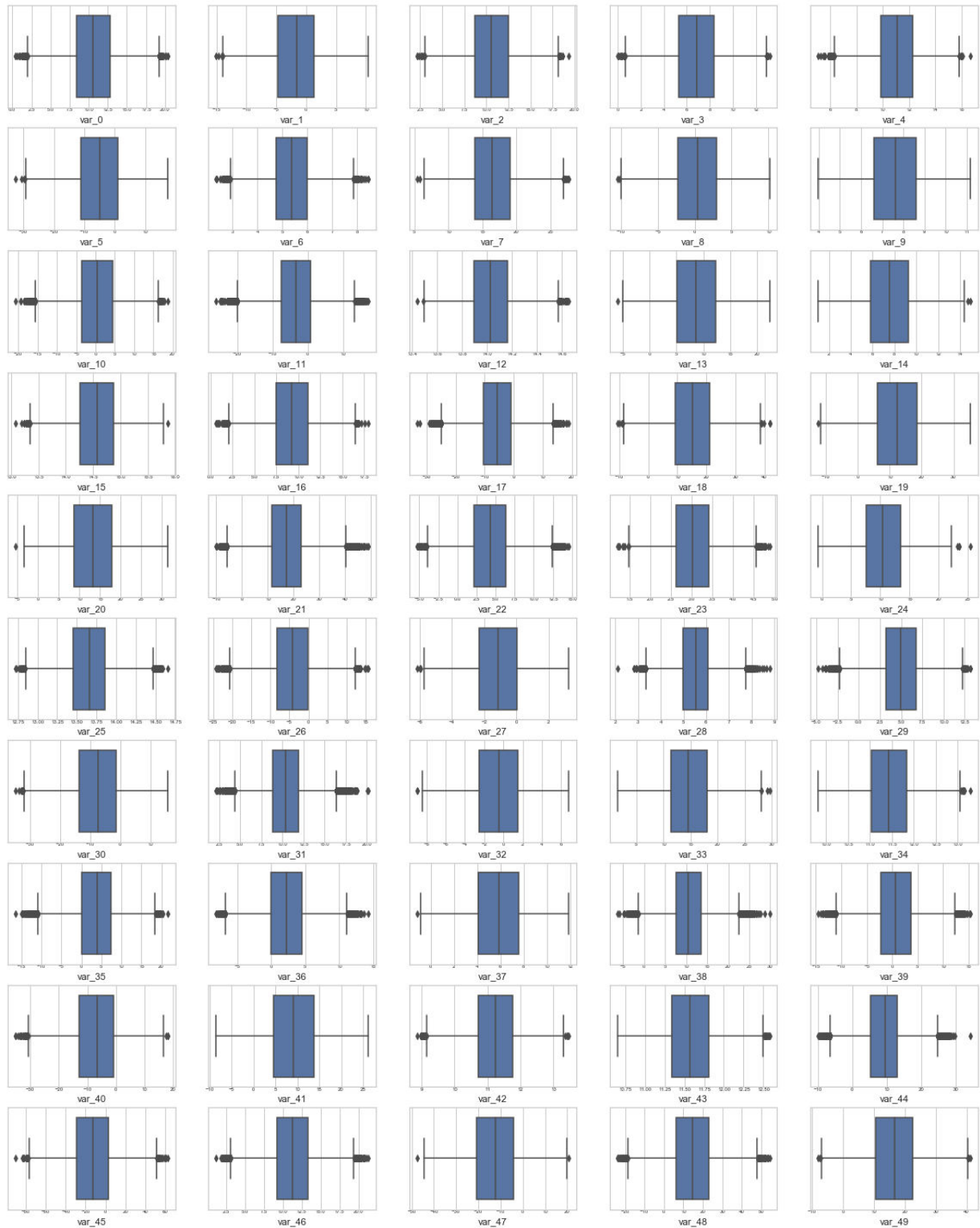


Figure 2.13 Outlier analysis using boxplots on train_dataset (var_0 to var_49)

Findings:

- As from the boxplots it is clear that almost all the variables have outliers

- After separating outliers and inliers with IQR method we found that all the target variables **with label as one** are outliers. As we have imbalanced data, it is bound to happen that the observations having minority class (target=1) will be shown as outliers.
- Outliers present in our data, are meaningful and contains important information so they can't be removed.

Python code

```
#1. %%time
def plot_feature_boxplot(df, features):
    i = 0
    sns.set_style('whitegrid')
    plt.figure()
    fig, ax = plt.subplots(10,5,figsize=(18,24))
    for feature in features:
        i += 1
        plt.subplot(10,5,i)
        sns.boxplot(df[feature])
        plt.xlabel(feature, fontsize=11)
        locs, labels = plt.xticks()
        plt.tick_params(axis='x', labelsize=6, pad=-6)
        plt.tick_params(axis='y', labelsize=6)
    plt.show()

#2. %%time
features = train.columns.values[2:52]
plot_feature_boxplot(train, features)

#3. print("df.shape:",train.shape)
df_in = train[~((train < (Q1 - 1.5 * IQR)) |(train > (Q3 + 1.5 * IQR))).any(axis=1)]
df_out = train[((train < (Q1 - 1.5 * IQR)) |(train > (Q3 + 1.5 * IQR))).any(axis=1)]
print("df_in.shape:",df_in.shape)
print("df_out.shape:",df_out.shape)

output: df.shape: (200000, 202)
        df_in.shape: (157999, 202)
        df_out.shape: (42001, 202)

#4. df_in['target'].value_counts()

output: 0 – 157999

#5. df_out['target'].value_counts()

output: 0- 21903
        1- 20098

#6. train['target'].value_counts()

output: 0- 179902
        1 -20098
```

Outlier analysis in R and Python is pasted below:

2.1.3 Feature selection

Feature Selection is the process where you automatically or manually select those features which contribute most to your prediction variable or output in which you are interested

in. Having irrelevant features in your data can decrease the accuracy of the models and make your model learn based on irrelevant features.

- **Reduces Overfitting:** Less redundant data means less opportunity to make decisions based on noise.
- **Improves Accuracy:** Less misleading data means modeling accuracy improves.
- **Reduces Training Time:** fewer data points reduce algorithm complexity and algorithms train faster.

There are few Feature selection techniques that can be used viz., Correlation analysis, Chi-Square test, ANOVA, Univariate analysis, Feature Importance. In this project as there are no categorical variables we will perform Correlation analysis and Feature importance.

Correlation analysis

Correlation is a technique for investigating the linear relationship between two quantitative, continuous variables.

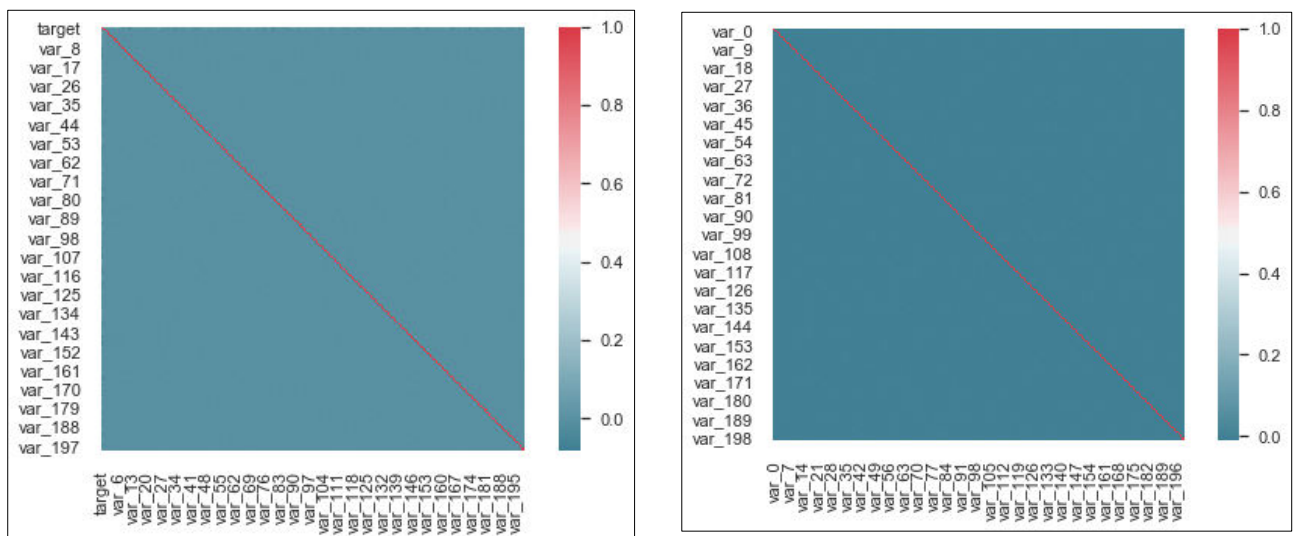


Figure 2.13 Correlation_train dataset and test dataset

Correlation analysis on test attributes

```
#Correlations in test attributes
test_attributes=test.columns.values[1:201]
test_correlations=test[test_attributes].corr().abs().unstack().sort_values(kind='quicksort').reset_index()
test_correlations=test_correlations[test_correlations['level_0']!=test_correlations['level_1']]
print(test_correlations.head(10))
print(test_correlations.tail(10))
```

```
level_0 level_1 0
0 var_154 var_175 1.477268e-07
1 var_175 var_154 1.477268e-07
2 var_188 var_113 1.639749e-07
3 var_113 var_188 1.639749e-07
4 var_131 var_8 4.695407e-07
5 var_8 var_131 4.695407e-07
6 var_60 var_189 9.523709e-07
7 var_189 var_60 9.523709e-07
8 var_159 var_96 1.147835e-06
9 var_96 var_159 1.147835e-06
level_0 level_1 0
39790 var_122 var_164 0.008513
39791 var_164 var_122 0.008513
39792 var_164 var_2 0.008614
39793 var_2 var_164 0.008614
39794 var_31 var_132 0.008714
39795 var_132 var_31 0.008714
39796 var_96 var_143 0.008829
39797 var_143 var_96 0.008829
39798 var_139 var_75 0.009868
39799 var_75 var_139 0.009868
Wall time: 21.3 s
```

Correlation analysis on train attributes

```
#Correlations in train attributes
train_attributes=train.columns.values[2:202]
train_correlations=train[train_attributes].corr().abs().unstack().sort_values(kind='quicksort').reset_index()
train_correlations=train_correlations[train_correlations['level_0']!=train_correlations['level_1']]
print(train_correlations.head(10))
print(train_correlations.tail(10))
```

```
level_0 level_1 0
0 var_75 var_191 2.703975e-08
1 var_191 var_75 2.703975e-08
2 var_173 var_6 5.942735e-08
3 var_6 var_173 5.942735e-08
4 var_126 var_109 1.313947e-07
5 var_109 var_126 1.313947e-07
6 var_144 var_27 1.772502e-07
7 var_27 var_144 1.772502e-07
8 var_177 var_100 3.116544e-07
9 var_100 var_177 3.116544e-07
level_0 level_1 0
39790 var_183 var_189 0.009359
39791 var_189 var_183 0.009359
39792 var_174 var_81 0.009490
39793 var_81 var_174 0.009490
39794 var_81 var_165 0.009714
39795 var_165 var_81 0.009714
39796 var_53 var_148 0.009788
39797 var_148 var_53 0.009788
39798 var_26 var_139 0.009844
39799 var_139 var_26 0.009844
Wall time: 22.2 s
```

Findings:

- As from the heatmap, variables in train and test data are independent of each other.
- Numerical analysis has given results of correlation values of less than (<0.005) in both train and test variables indicating the independence of predictor variables.

2.1.4 Feature Engineering

Feature engineering is about **creating new input features** from your existing ones. In general, you can think of data cleaning as a process of subtraction and feature engineering as a process of addition.

Feature importance using random forest

We'll use a random forest model to estimate the important features. Based on the importance, we'll use a cut of to choose the top features for training our model. Here while estimating the importance of a variable, each time a feature is dropped and the loss of accuracy is used to

#R code

#Split the training data

```
train_index<-sample(1:nrow(train),0.75*nrow(train))
```

```
train_data<-train[train_index,]
```

```
valid_data<-train[-train_index,]
```

#Training the Random forest classifier

```
set.seed(2732)
```

```
train_data$target<-as.factor(train_data$target)
```

```
mtry<-floor(sqrt(200))
```

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

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

#Variable importance

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

estimate the importance of that particular feature.

	Mean Decrease Gini
var_0	152.17478
var_1	161.68201
var_2	185.55470
var_3	109.88496
var_4	109.04578
var_5	146.64082
var_6	212.00188
var_7	102.89189
var_8	96.61942
var_9	163.15381
var_10	108.03386
var_11	112.77791
var_12	260.73034
var_13	183.27466
var_14	101.42348
var_15	117.01380
var_16	118.63159
var_17	111.39040
var_18	157.34826
var_19	105.87776

Findings:

var_12, var_26, var_22, var_174, var_198 and so on are the top important features based on Mean decrease Gini.

Permutation importance

Let's look at the permutation importance method of variable importance that changes the order of a column and measures the loss of accuracy of the model to estimate the importance of the feature set. Instead of dropping a feature like in a random forest method, the feature column is randomized. Intuitively, we should see same/similar set of features from both the techniques, ordered differently, and the performance shouldn't differ drastically. We'll use eli5 to compute importance.

```
#training data
X=train.drop(columns=['ID_code','target'],axis=1)
test=test.drop(columns=['ID_code'],axis=1)
y=train['target']
#Split the training data
X_train,X_valid,y_train,y_valid=train_test_split(X,y,random_state=42)
#Random forest classifier
rf_model=RandomForestClassifier(n_estimators=10,random_state=42)
#fitting the model
rf_model.fit(X_train,y_train)
#Let us calculate weights and show important features using eli5 library.
from eli5.sklearn import PermutationImportance
perm_imp=PermutationImportance(rf_model,random_state=42)
#fitting the model
perm_imp.fit(X_valid,y_valid)
#Important features
eli5.show_weights(perm_imp,feature_names=X_valid.columns.tolist(),top=200)
```

Python Code

Out[17]:		
	Weight	Feature
	0.0004 ± 0.0002	var_81
	0.0003 ± 0.0002	var_146
	0.0003 ± 0.0002	var_109
	0.0003 ± 0.0002	var_12
	0.0002 ± 0.0001	var_110
	0.0002 ± 0.0000	var_173
	0.0002 ± 0.0001	var_174
	0.0002 ± 0.0002	var_0
	0.0002 ± 0.0002	var_26
	0.0001 ± 0.0001	var_166
	0.0001 ± 0.0001	var_169
	0.0001 ± 0.0001	var_22
	0.0001 ± 0.0001	var_99
	0.0001 ± 0.0001	var_53
	0.0001 ± 0.0001	var_8
	0.0001 ± 0.0001	var_1
	0.0001 ± 0.0000	var_37
	0.0001 ± 0.0003	var_133
	0.0001 ± 0.0000	var_152
	0.0001 ± 0.0001	var_175
	0.0001 ± 0.0001	var_88

Findings:

- The variables in green rows have positive impact on our prediction
- The variables in white rows have no impact on our prediction
- The variables in red rows have negative impact on our prediction

2.1 Modeling

In our early stages of analysis during pre-processing we have come to understand that our dataset is imbalanced. So first of all imbalanced data should be handled properly and then different models should be tried out to find the best fit.

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.

First step: Ways to handle imbalanced data

Use the right evaluation metrics: accuracy is not the best metric to use when evaluating imbalanced datasets as it can be very misleading. Metrics that can provide better insight include:

- **Confusion Matrix:** a table showing correct predictions and types of incorrect predictions.
- **Precision:** the number of true positives divided by all positive predictions. Precision is also called Positive Predictive Value. It is a measure of a classifier's exactness. Low precision indicates a high number of false positives.
- **Recall:** the number of true positives divided by the number of positive values in the test data. Recall is also called Sensitivity or the True Positive Rate. It is a measure of a classifier's completeness. Low recall indicates a high number of false negatives.
- **F1 Score:** the weighted average of precision and recall.

Resample the training set

Apart from using different evaluation criteria, one can also work on getting different dataset. Two approaches to make a balanced dataset out of an imbalanced one are under-sampling and over-sampling.

- **Under-sampling:** Under-sampling balances the dataset by reducing the size of the abundant class. This method is used when quantity of data is sufficient. By keeping all samples in the rare class and randomly selecting an equal number of samples in the abundant class, a balanced new dataset can be retrieved for further modelling.
- **Over-sampling:** On the contrary, oversampling is used when the quantity of data is insufficient. It tries to balance dataset by increasing the size of rare samples. Rather than getting rid of abundant samples, new rare samples are generated by using e.g. repetition, bootstrapping or SMOTE (Synthetic Minority Over-Sampling Technique) [1].
- **Use Stratified K-fold Cross-Validation:** Stratification is the process of rearranging the data as to ensure each fold is a good representative of the whole. For example, in a binary classification problem where each class comprises 50% of the data, it is best to arrange the data such that in every fold, each class comprises around half the instances. It is noteworthy that cross-validation should be applied properly while using over-sampling method to address imbalance problems. Keep in mind that over-sampling takes observed rare samples and applies bootstrapping to generate new random data based on a distribution function. If cross-validation is applied after over-sampling, basically what we are doing is overfitting our model to a specific artificial bootstrapping result. That is why cross-validation should always be done before over-sampling the data, just as how feature selection should be implemented.

Change the algorithm

While in every machine learning problem, it's a good rule of thumb to try a variety of algorithms, it can be especially beneficial with imbalanced datasets.

(Source: <https://towardsdatascience.com/methods-for-dealing-with-imbalanced-data-5b761be45a18&https://www.kdnuggets.com/2017/06/7-techniques-handle-imbalanced-data.html>)

We are going to build models using above techniques to predict the target variable. You always start your model building from the simplest to more complex. Therefore, we use Logistic regression.

2.1.1 Logistic regression using K-fold Stratified Cross Validation

Logistic Regression: Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable (or output), y , can take only discrete values for given set of features (or inputs), X .

Contrary to popular belief, logistic regression IS a regression model. The model builds a regression model to predict the probability that a given data entry belongs to the category numbered as "1". Just like Linear regression assumes that the data follows a linear function, Logistic regression models the data using the sigmoid function

Python Code

Training dataset for modelling

```
#Training data
```

```
X=train.drop(['ID_code','target'],axis=1)
```

```
Y=train['target']
```

StratifiedKFold cross validator

```
cv=StratifiedKFold(n_splits=5,random_state=42,shuffle=True)
```

```
for train_index,valid_index in cv.split(X,Y):
```

```
X_train, X_valid=X.iloc[train_index], X.iloc[valid_index]
```

```
train, y_valid=Y.iloc[train_index], Y.iloc[valid_index]
```

Logistic regression model

```
lr_model=LogisticRegression(random_state=42)
```

fitting the lr model

```
lr_model.fit(X_train,y_train)
```

Accuracy of the model

```
lr=lr_model.score(X_train,y_train)
```

```
Accuracy of the model : 0.9148
```

Cross validation prediction

```
cv_predict=cross_val_predict(lr_model,X_valid,y_valid,cv=5)
```

Cross validation score

```
cv_score=cross_val_score(lr_model,X_valid,y_valid,cv=5)
```

```
print('cross_val_score : ',np.average(cv_score))
```

```
Cross_val_score : 0.9116
```

```
#Predicting the model
```

```
X_test=test.drop(['ID_code'],axis
```

```
r_pred=lr_model.predict(X_test)
```

R code

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

#Split the data using CreateDataPartition

```
train.index<-createDataPartition(train$target,p=0.8,list=FALSE)
```

```
train.data<-train[train.index,]
```

```
valid.data<-train[-train.index,]
```

#Training dataset

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

```
y_t<-as.matrix(train.data$target)
```

#validation dataset

```
X_v<-as.matrix(valid.data[, -c(1,2)])
```

```
y_v<-as.matrix(valid.data$target)
```

#test dataset

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

#Logistic regression model

```
set.seed(667)
```

```
lr_model <- glmnet(X_t,y_t, family = "binomial")
```

```
summary(lr_model)
```

#Cross validation prediction

```
set.seed(8909)
```

```

cv_lr <- cv.glmnet(X_t,y_t,family = "binomial", type.measure = "class")
#Plotting the missclassification error vs log(lambda) where lambda is regularization parameter
#Minimum lambda
cv_lr$lambda.min
#plot the auc score vs log(lambda)
plot(cv_lr)
#Model performance on validation dataset
set.seed(5363)
cv_predict.lr<-predict(cv_lr,X_v,s = "lambda.min", type = "class")
#Confusion matrix
set.seed(689)
#actual target variable
target<-valid.data$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.lr<-as.factor(cv_predict.lr)
confusionMatrix(data=cv_predict.lr,reference=target)
#ROC_AUC score and curve
set.seed(892)
cv_predict.lr<-as.numeric(cv_predict.lr)
roc(data=valid.data[, -c(1,2)],response=target,predictor=cv_predict.lr, auc=TRUE,
plot=TRUE)
#predict the model
lr_pred<-predict(lr_model,test[, -c(1)],type='class')

```

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

Synthetic Minority Oversampling Technique (SMOTE)

SMOTE uses a nearest neighbour's algorithm to generate new and synthetic data to use for training the model. In order to balance imbalanced data, we are going to use SMOTE sampling method.

Python code

```

from imblearn.over_sampling import SMOTE
#Synthetic Minority Oversampling Technique
sm = SMOTE(random_state=42, ratio=1.0)
#Generating synthetic data points
X_smote,y_smote=sm.fit_sample(X_train,y_train)
X_smote_v,y_smote_v=sm.fit_sample(X_valid,y_valid)
#Logistic regression model for SMOTE
smote=LogisticRegression(random_state=42)
#fitting the smote model
smote.fit(X_smote,y_smote)
smote_score=smote.score(X_smote,y_smote)
print('Accuracy of the smote_model :',smote_score)

```

Accuracy of the model : 0.798

```
#Cross validation prediction
cv_pred=cross_val_predict(smote,X_smote_v,y_smote_v,cv=5)
#Cross validation score
cv_score=cross_val_score(smote,X_smote_v,y_smote_v,cv=5)
print('cross_val_score : ',np.average(cv_score))
cross_val_score : 0.806
#Predicting the model
X_test=test.drop(['ID_code'],axis=1)
smote_pred=smote.predict(X_test)
```

ROC score : 0.8006114508060033

R code**Random Oversampling Examples (ROSE)**

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

#Random Oversampling Examples(ROSE)

```
set.seed(699)
#train.data$target<-as.factor(train.data$target)
train.rose <- ROSE(target~., data =train.data[,-c(1)],seed=32)$data
table(train.rose$target)
valid.rose <- ROSE(target~., data =valid.data[,-c(1)],seed=42)$data
table(valid.rose$target)
```

#Baseline logistic regression model

```
set.seed(462)
lr_rose <-glmnet(as.matrix(train.rose),as.matrix(train.rose$target),
family = "binomial")
summary(lr_rose)
#Cross validation prediction
set.seed(473)
cv_rose = cv.glmnet(as.matrix(valid.rose),as.matrix(valid.rose$target),
family = "binomial", type.measure = "class")
```

#Minimum lambda

```
cv_rose$lambda.min
#plot the auc score vs log(lambda)
plot(cv_rose)
```

#Model performance on validation dataset

```
set.seed(442)
cv_predict.rose<-predict(cv_rose,as.matrix(valid.rose),s = "lambda.min",
type = "class")
cv_predict.rose
```

#Confusion matrix

```

set.seed(478)
#actual target variable
target<-valid.rose$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.rose<-as.factor(cv_predict.rose)
confusionMatrix(data=cv_predict.rose,reference=target)
#ROC_AUC score and curve
set.seed(843)
cv_predict.rose<-as.numeric(cv_predict.rose)
roc(data=valid.rose[, -c(1,2)],response=target,predictor=cv_predict.rose, auc=TRUE,
plot=TRUE)
#predict the model
set.seed(6543)
rose_pred<-predict(lr_rose,test[, -c(1)],type='class')

```

LightGBM

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

```

Python code
Let us build LightGBM model
#Training the model
#training data
lgb_train=lgb.Dataset(X_train,label=y_train)
#validation data
lgb_valid=lgb.Dataset(X_valid,label=y_valid)
#Selecting best hyper parameters by tuning of different parameters
params={'boosting_type': 'gbdt',
'max_depth': -1, #no limit for max_depth if <0
'objective': 'binary',
'boost_from_average':False,
'nthread': 8,
'metric': 'auc',
'num_leaves': 100,
'learning_rate': 0.03,
'max_bin': 950, #default 255
'subsample_for_bin': 200,
'subsample': 1,
'subsample_freq': 1,
'colsample_bytree': 0.8,
'reg_alpha': 1.2, #L1 regularization(>0)
'reg_lambda': 1.2, #L2 regularization(>0)
'min_split_gain': 0.5, #>0
'min_child_weight': 1,
'min_child_samples': 5,

```

```

'is_unbalance':True,
}
num_rounds=3000
lgbm= lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_valid],
verbose_eval=100,early_stopping_rounds = 1000)
X_test=test.drop(['ID_code'],axis=1)
#predict the model
#probability predictions
lgbm_predict_prob=lgbm.predict(X_test,random_state=42,
num_iteration=lgbm.best_iteration)
#Convert to binary output 1 or 0
lgbm_predict=np.where(lgbm_predict_prob>=0.5,1,0)

```

```

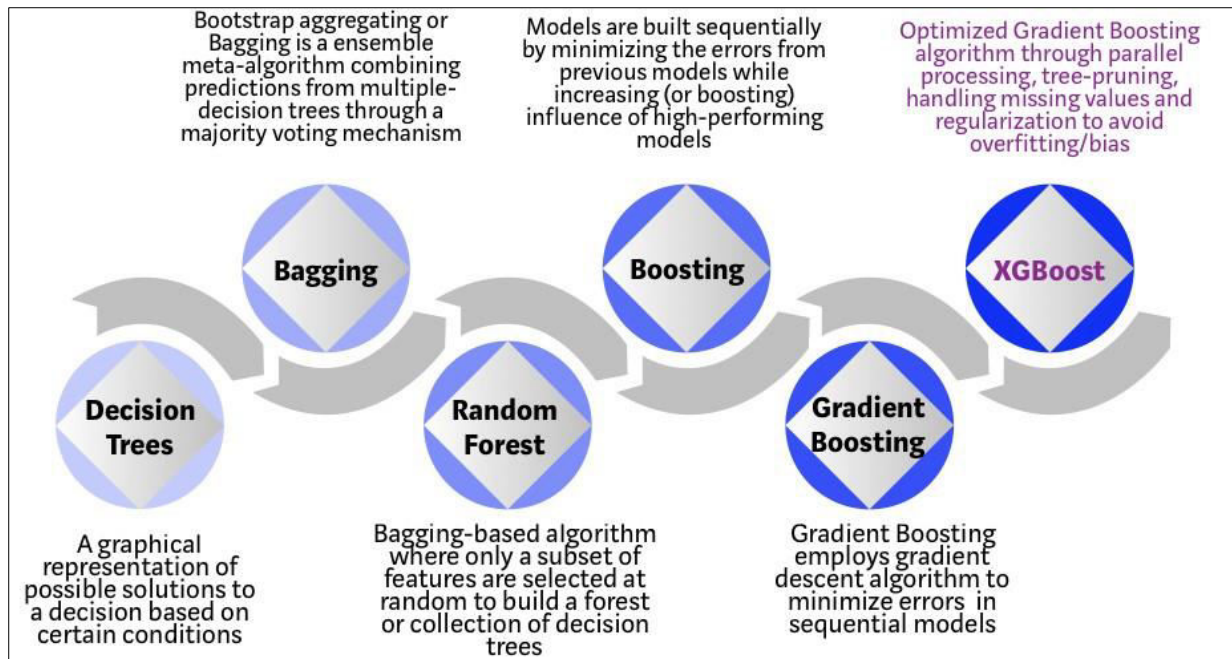
Training until validation scores don't improve for 5000 rounds
[1000] training's auc: 0.938996      valid_1's auc: 0.885963
[2000] training's auc: 0.958629      valid_1's auc: 0.890769
[3000] training's auc: 0.972001      valid_1's auc: 0.89195
[4000] training's auc: 0.981625      valid_1's auc: 0.892447
[5000] training's auc: 0.988357      valid_1's auc: 0.892444
[6000] training's auc: 0.992858      valid_1's auc: 0.892633
[7000] training's auc: 0.995834      valid_1's auc: 0.892332
[8000] training's auc: 0.997652      valid_1's auc: 0.89205
[9000] training's auc: 0.99874      valid_1's auc: 0.891803
[10000] training's auc: 0.999366      valid_1's auc: 0.891481
Did not meet early stopping. Best iteration is:
[10000] training's auc: 0.999366      valid_1's auc: 0.891481

<lightgbm.basic.Booster at 0x1bd0b3fa668>

```

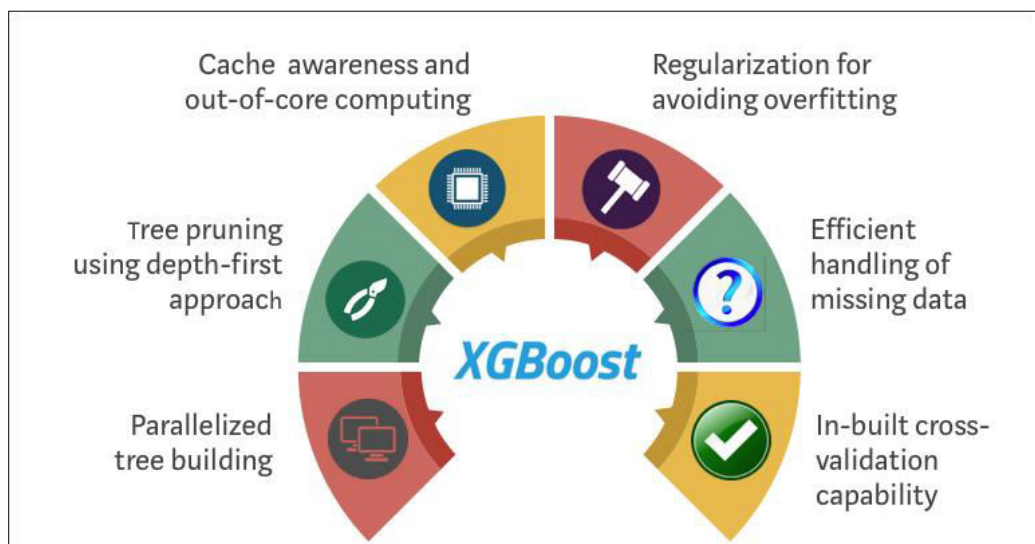
XG Boost in R

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree based algorithms are considered best-in-class right now. Please see the chart below for the evolution of tree-based algorithms over the years.



Evolution of XGBoost Algorithm from Decision Trees

XGBoost and Gradient Boosting Machines (GBMs) are both ensemble tree methods that apply the principle of boosting weak learners ([CARTs](#) generally) using the gradient descent architecture. However, XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements.



R code

#Loading Libraries

```
library(data.table)
library(caret)
library(xgboost)
```

```

library(pROC)

#setting working directory

setwd("F:/EdwisorVanusha/Project/Santander")

#loading the data

train=read.csv("train.csv")

test=read.csv("test.csv")

#Let's check the dimension of train and test sets. Also check what are the variables that are
there in train but not in test. Also let's have a look at the head of the data sets

dim(train) ; dim(test) ; setdiff(colnames(train) , colnames(test)) ; head(train) ; head(test)

#It seems like the variables have no names as such and the only variable that is missing in the
test set is the target column which we need to predict. Let's check the sample submission file.
We will check if the ids in sample sub and the test file are in same order or not as well.

#training and testing data

trainX=as.matrix(train[-c(1,2)])

trainY=as.matrix(train$target)

testX=as.matrix(test[-c(1)])

# preparing XGB matrix

dtrain <- xgb.DMatrix(data = as.matrix(trainX), label = as.matrix(trainY))

# parameters

params <- list(booster = "gbtree",
               objective = "binary:logistic",
               eta=0.02,
               #gamma=80,
               max_depth=2,
               min_child_weight=1,
               subsample=0.5,
               colsample_bytree=0.1,
               scale_pos_weight = round(sum(!trainY) / sum(trainY), 2))

# CV

set.seed(123)

xgbcv <- xgb.cv(params = params,
               data = dtrain,
               nrounds = 30000,
               nfold = 5,
               showsd = F,
               stratified = T,
               print_every_n = 100,

```

```

        early_stopping_rounds = 500,
        maximize = T,
        metrics = "auc")
cat(paste("Best iteration:", xgbcv$best_iteration))

# train final model

set.seed(123)
xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = xgbcv$best_iteration,
  print_every_n = 100,
  maximize = T,
  eval_metric = "auc")

#view variable importance plot

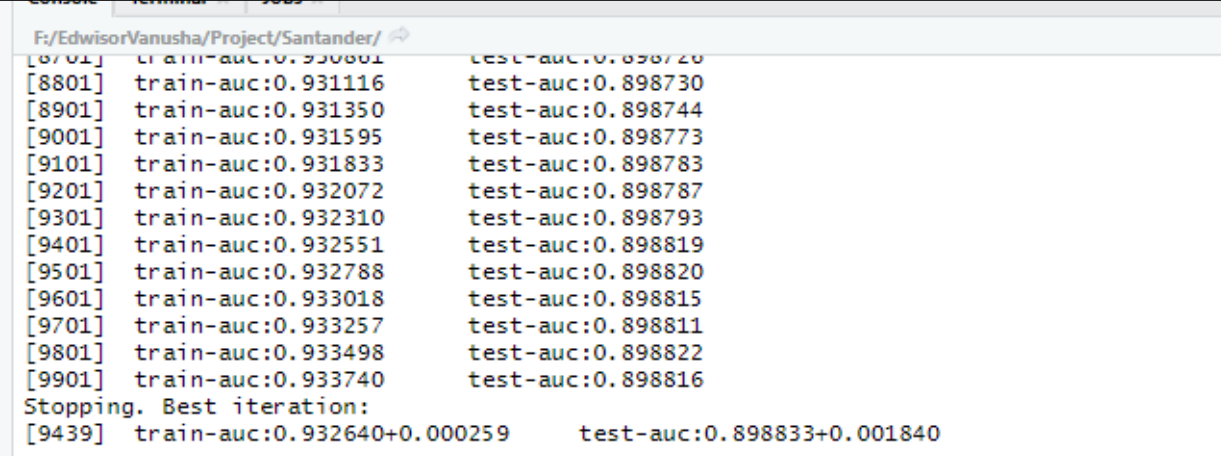
imp_mat <- xgb.importance(feature_names = colnames(trainX), model = xgb_model)

xgb.plot.importance(importance_matrix = imp_mat[1:30])

#prediction

pred_sub <- predict(xgb_model, newdata=as.matrix(testX), type="response")

```



```

F:/EdwisorVanusha/Project/Santander/
[8701] train-auc:0.930801    test-auc:0.898720
[8801] train-auc:0.931116    test-auc:0.898730
[8901] train-auc:0.931350    test-auc:0.898744
[9001] train-auc:0.931595    test-auc:0.898773
[9101] train-auc:0.931833    test-auc:0.898783
[9201] train-auc:0.932072    test-auc:0.898787
[9301] train-auc:0.932310    test-auc:0.898793
[9401] train-auc:0.932551    test-auc:0.898819
[9501] train-auc:0.932788    test-auc:0.898820
[9601] train-auc:0.933018    test-auc:0.898815
[9701] train-auc:0.933257    test-auc:0.898811
[9801] train-auc:0.933498    test-auc:0.898822
[9901] train-auc:0.933740    test-auc:0.898816
Stopping. Best iteration:
[9439] train-auc:0.932640+0.000259    test-auc:0.898833+0.001840

```

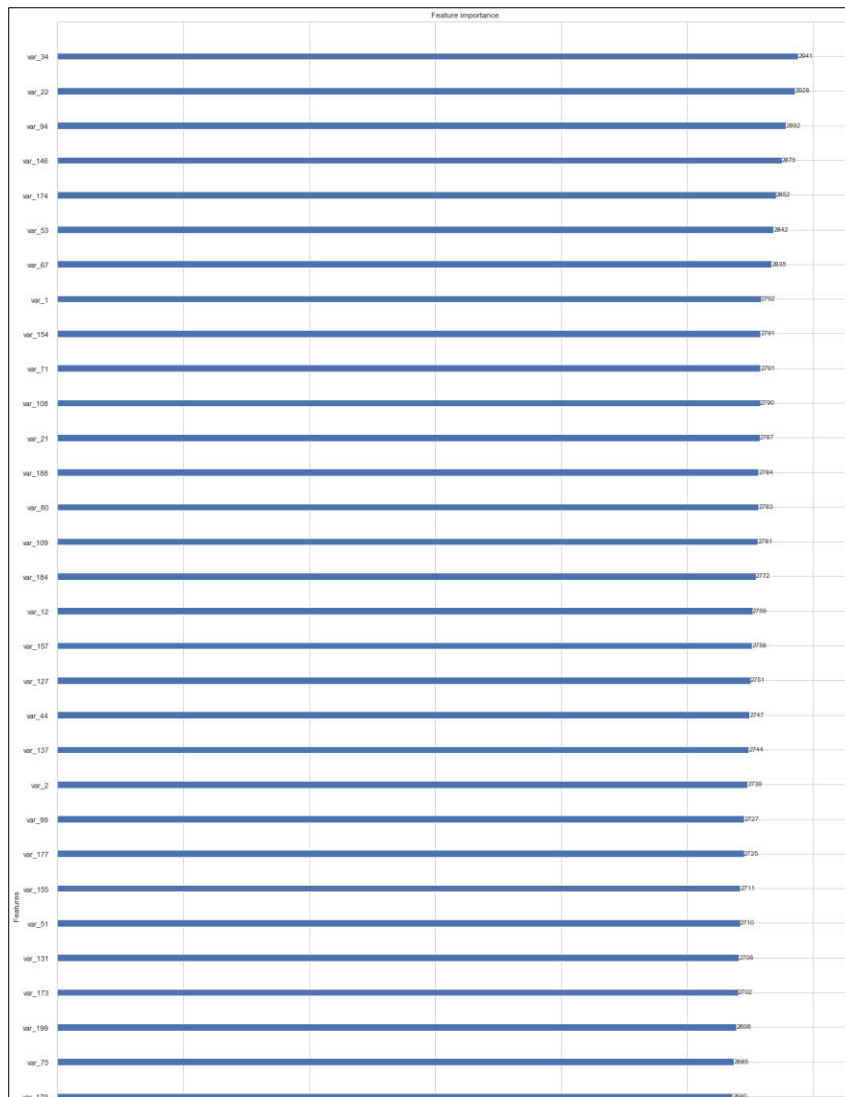
Important features plot

Python code

```

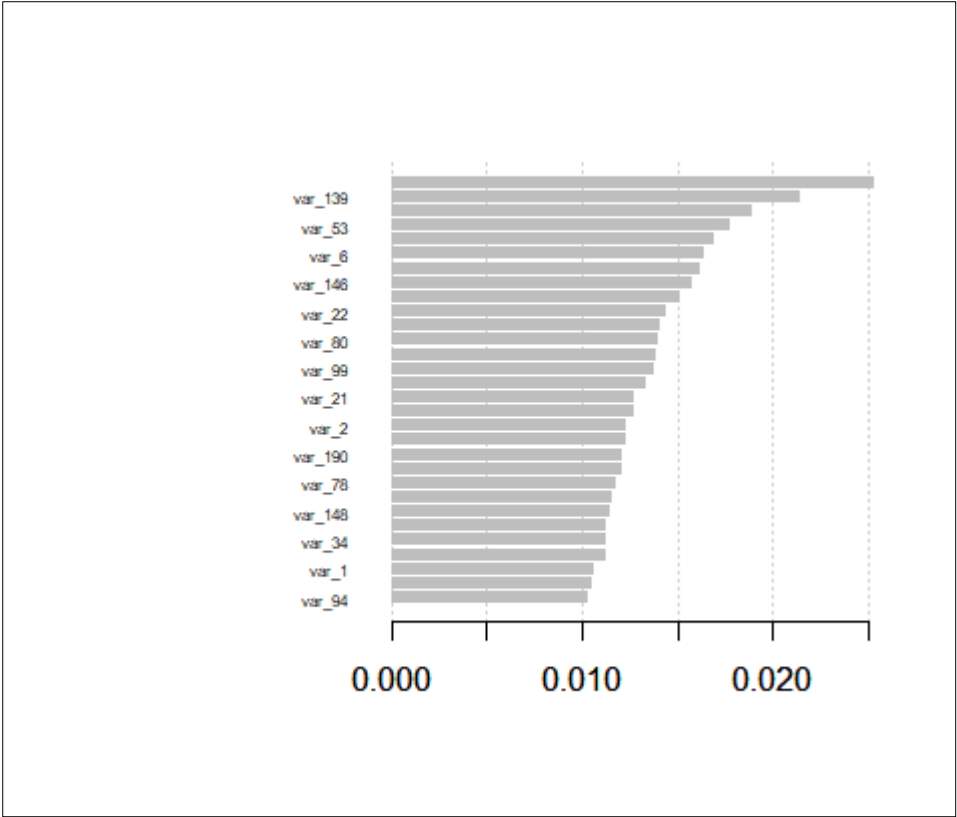
#plot the important features
lgb.plot_importance(lgbm,max_num_features=150,importance_type="split",figsize=(20,50))

```



R code

```
#view variable importance plot
imp_mat <- xgb.importance(feature_names = colnames(trainX), model = xgb_model)
xgb.plot.importance(importance_matrix = imp_mat[1:30])
```



Chapter 3

Conclusion

3.1 Model Evaluation

Now that we have built 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.

1. Predictive Performance
2. Interpretability
3. Computational Efficiency

In our case of transaction prediction, with imbalanced classes, it's easy to get a high accuracy without actually making useful predictions. So, accuracy as an evaluation metrics makes sense only if the class labels are uniformly distributed. In case of imbalanced classes confusion-matrix is good technique to summarizing the performance of a classification algorithm.

For Imbalanced data classification problem, Precision & Recall, ROC_AUC_Score is used for model evaluation.

3.1.1 Confusion Matrix

Confusion Matrix is a performance measurement for a classification algorithm where output can be two or more classes. A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known

- **True positives (TP):** These are cases in which we predicted yes (they have the disease), and they do have the disease.
- **True negatives (TN):** We predicted no, and they don't have the disease.
- **False positives (FP):** We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- **False negatives (FN):** We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

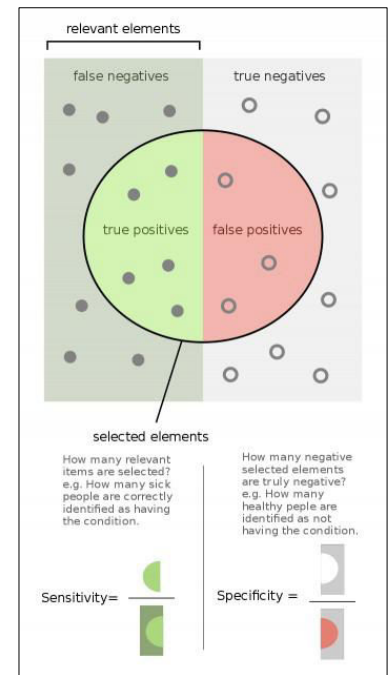
	Actual = Yes	Actual = No
Predicted = Yes	TP	FP
Predicted = No	FN	TN

- **Accuracy:** Percentage of total items classified correctly- $(TP+TN)/(\text{Total Predictions})$
- **Misclassification Error:** The ratio of incorrect predictions to total predictions

Error rate= $(FN+FP)/\text{Total Predictions}$

- **Recall or Sensitivity or TPR (True Positive Rate):** Number of items correctly identified as positive out of total true positives- $TP/(TP+FN)$
- **Specificity or TNR (True Negative Rate):** Number of items correctly identified as negative out of total negatives- $TN/(TN+FP)$
- **Precision:** Number of items correctly identified as positive out of total items identified as positive- $TP/(TP+FP)$
- **False Positive Rate or Type I Error:** Number of items wrongly identified as positive out of total true negatives- $FP/(FP+TN)$
- **False Negative Rate or Type II Error:** Number of items wrongly identified as negative out of total true positives- $FN/(FN+TP)$
- **F1 Score:** It is a harmonic mean of precision and recall given by-

$$F1 = 2 * Precision * Recall / (Precision + Recall)$$



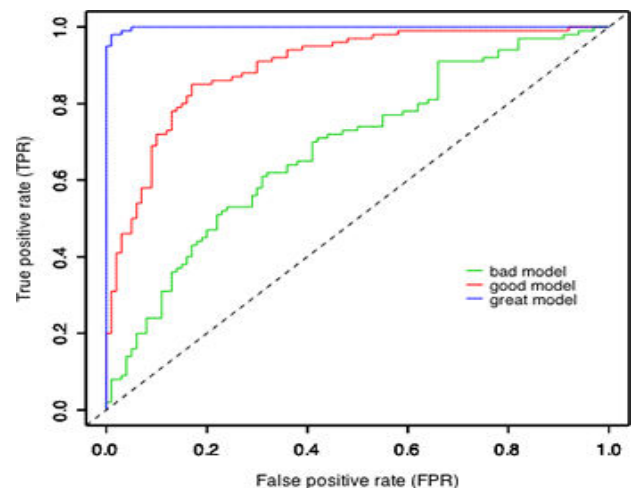
Source of Image: Wikipedia

3.1.2 Receiver operating characteristics (ROC)_Area under curve(AUC) Score

The probabilistic interpretation of ROC-AUC score is that if you randomly choose a positive case and a negative case, the probability that the positive case outranks the negative case according to the classifier is given by the AUC. Here, rank is determined according to order by predicted values.

Mathematically, it is calculated by area under curve of sensitivity (TPR) vs.

FPR(1-specificity). Ideally, we would like to have high sensitivity & high specificity, but in real-world scenarios, there is always a tradeoff between sensitivity & specificity.



Some important characteristics of ROC-AUC are-

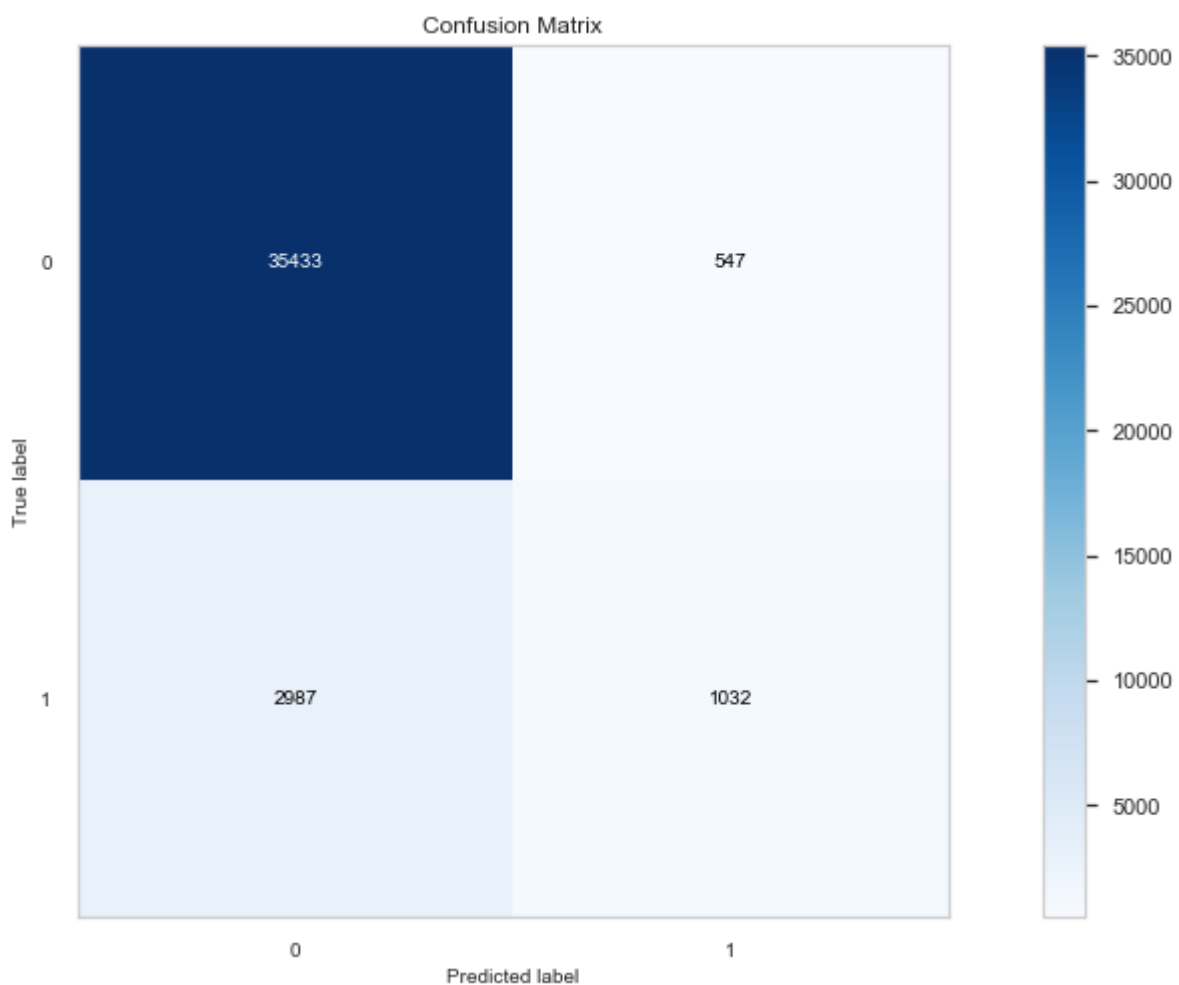
- The value can range from 0 to 1. However, auc score of a random classifier for balanced data is 0.5

Source of Image: UNC Lecture

- ROC-AUC score is independent of the threshold set for classification because it only considers the rank of each prediction and not its absolute value. The same is not true for F1 score which needs a threshold value in case of probabilities output.

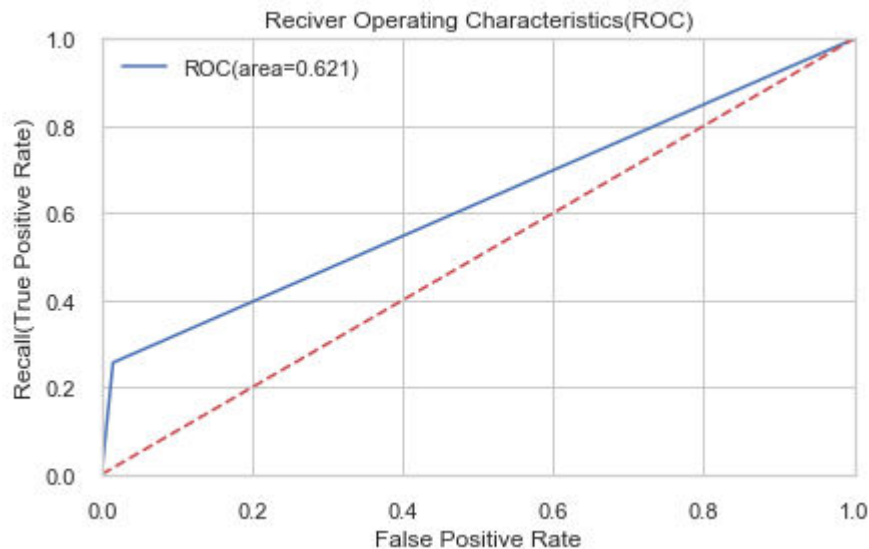
Logistic Regression

```
#Confusion matrix
cm=confusion_matrix(y_valid,cv_predict)
#Plot the confusion matrix
plot_confusion_matrix(y_valid,cv_predict,normalize=False,figsize=(15,8))
```



```
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_valid,cv_predict)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area=%0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1], 'r--')
```

```
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
```



AUC: 0.6207887015553276

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

Classification report

```
#Classification report
scores=classification_report(y_valid,cv_predict)
print(scores)
```

	precision	recall	f1-score	support
0	0.92	0.98	0.95	35980
1	0.65	0.26	0.37	4019
accuracy			0.91	39999
macro avg	0.79	0.62	0.66	39999
weighted avg	0.90	0.91	0.89	39999

We have observed that F1 score is high for number of customers those who will not make a transaction than who will make a transaction. So, we are going to change the algorithm.

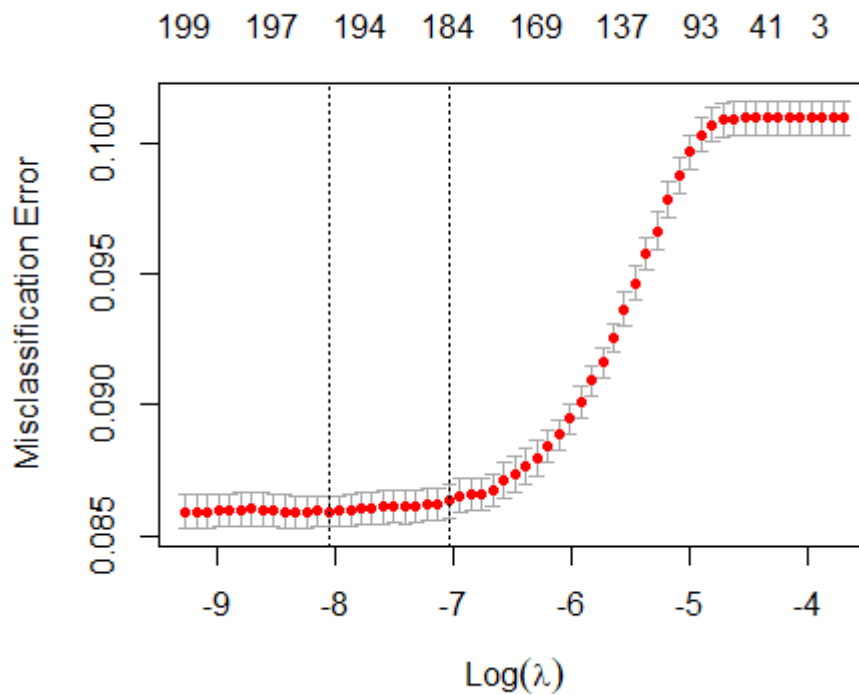
R code

Logistic Regression

```

#Cross validation prediction
set.seed(8909)
cv_lr <- cv.glmnet(X_t,y_t,family = "binomial", type.measure = "class")
#Plotting misclassification error vs log(lambda)
#Minimum lambda-Regularization parameter
cv_lr$lambda.min
#plot the auc score vs log(lambda)
plot(cv_lr)

```



Miss classification error increases as increasing the log(Lambda).

```

#Confusion matrix
set.seed(689)
#actual target variable
target<-valid.data$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.lr<-as.factor(cv_predict.lr)
confusionMatrix(data=cv_predict.lr,reference=target)

```

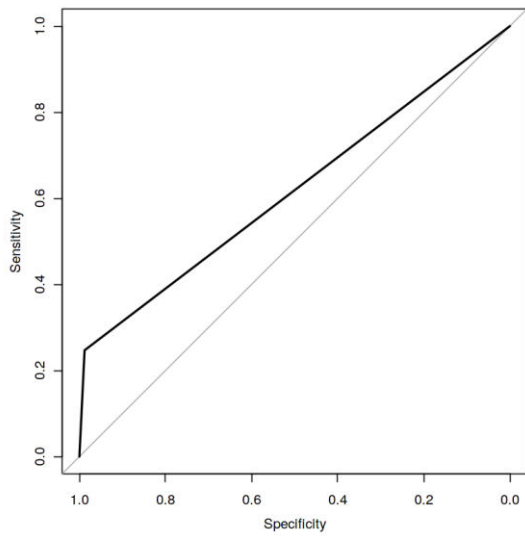
F:/EdwisorVanusha/Project/Santander/ ↗			
Confusion Matrix and Statistics			
	Reference		
Prediction	0	1	
0	35618	2973	
1	434	975	
Accuracy : 0.9148			
95% CI : (0.912, 0.9175)			
No Information Rate : 0.9013			
P-Value [Acc > NIR] : < 2.2e-16			
Kappa : 0.3292			
McNemar's Test P-Value : < 2.2e-16			
Sensitivity : 0.9880			
Specificity : 0.2470			
Pos Pred Value : 0.9230			
Neg Pred Value : 0.6920			
Prevalence : 0.9013			
Detection Rate : 0.8904			
Detection Prevalence : 0.9648			
Balanced Accuracy : 0.6175			
'Positive' Class : 0			
>			

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

```
#ROC_AUC score and curve
set.seed(892)
cv_predict.lr<-as.numeric(cv_predict.lr)
roc(data=valid.data[, -c(1,2)],response=target,predictor=cv_predict.lr, auc=TRUE,
     plot=TRUE)
```

Area under the curve: 0.5006

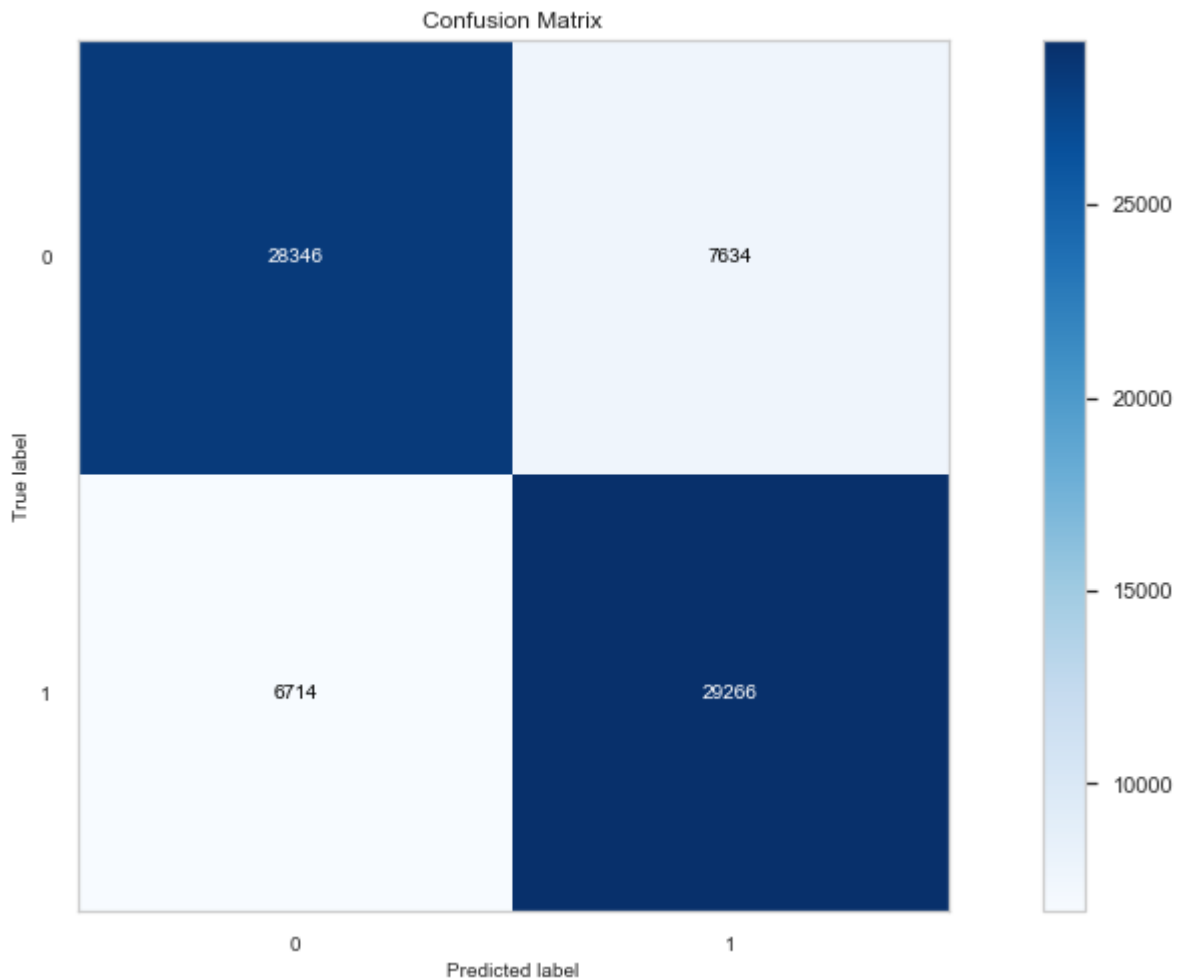
As from the results the auc score is very less and the specificity is also less so we do not accept this algorithm



Python code

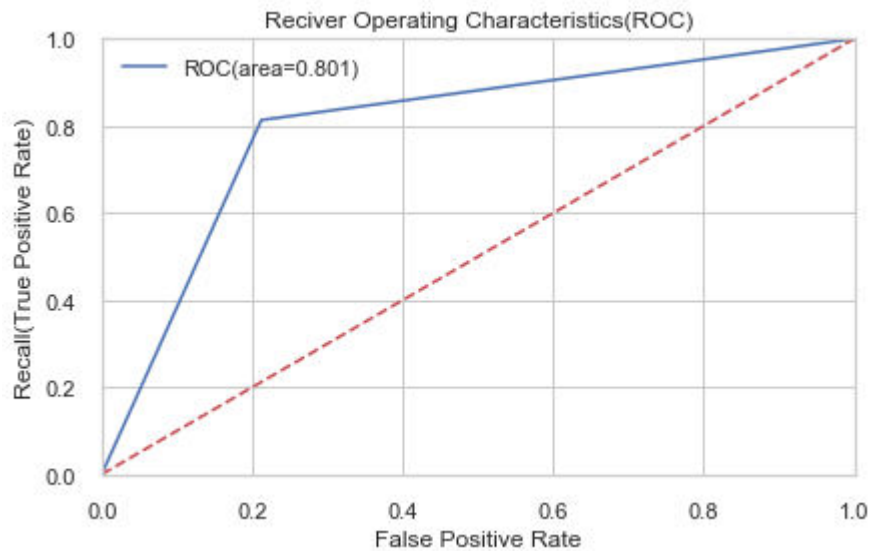
Synthetic Minority Oversampling Technique (SMOTE)

```
#Confusion matrix
cm=confusion_matrix(y_smote_v,cv_pred)
#Plot the confusion matrix
plot_confusion_matrix(y_smote_v,cv_pred,normalize=False,figsize=(15,8))
```



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

```
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_smote_v,cv_pred)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area= %0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
```



ROC score: 0.8006114508060033

Classification report

```
#Classification report
scores=classification_report(y_smote_v,cv_pred)
print(scores)
```

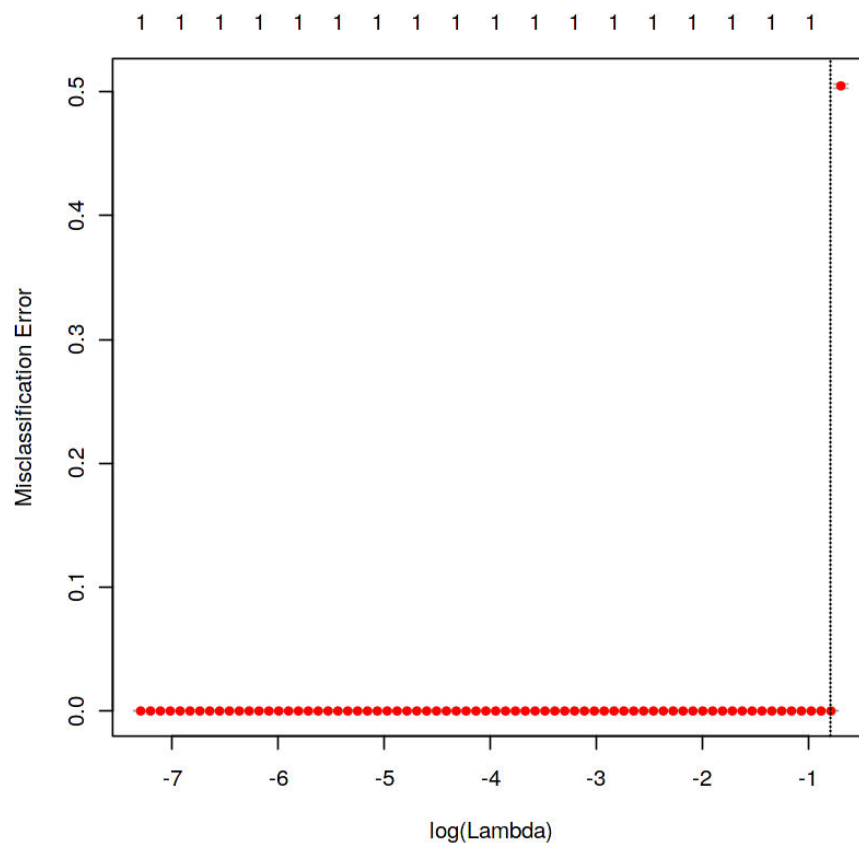
	precision	recall	f1-score	support
0	0.81	0.79	0.80	35980
1	0.79	0.81	0.80	35980
accuracy			0.80	71960
macro avg	0.80	0.80	0.80	71960
weighted avg	0.80	0.80	0.80	71960

We have observed that smote model is performing well on imbalance data compare to baseline logistic regression as the recall value for 1 has improved to 0.81

R code

Random Oversampling Examples (ROSE)

```
#Plotting misclassification error vs log(lambda)
#lambda-Regularization parameter
#Minimum lambda
cv_rose$lambda.min
#plot the auc score vs log(lambda)
plot(cv_rose)
```

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

Confusion Matrix and Statistics

	Reference	
Prediction	1	2
1	20012	0
2	0	19988

Accuracy : 1
95% CI : (0.9999, 1)

```
No Information Rate : 0.5003
P-Value [Acc > NIR] : < 2.2e-16
```

```
Kappa : 1
```

```
McNemar's Test P-Value : NA
```

```
Sensitivity : 1.0000
Specificity : 1.0000
Pos Pred Value : 1.0000
Neg Pred Value : 1.0000
Prevalence : 0.5003
Detection Rate : 0.5003
Detection Prevalence : 0.5003
Balanced Accuracy : 1.0000
```

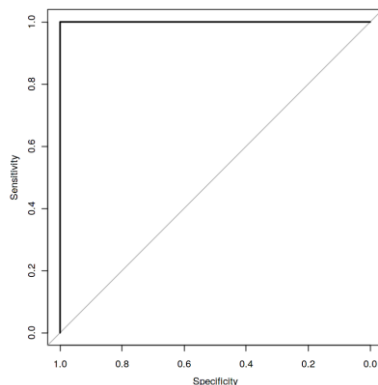
```
'Positive' Class : 1
```

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

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

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

Area under the curve: 1



I tried different ways to get good accuracy like changing count of one target class variable. Finally got area under ROC curve is 1 but this may not be possible.

3.2 Model Selection

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

1. Logistic regression model is not performed well on imbalanced data.
2. We balance the imbalanced data using resampling techniques like SMOTE in python and ROSE in R. and the recall for the minority class has improved over logistic regression
3. Baseline logistic regression model is performed well on balanced data.
4. LightGBM model performed well on imbalanced data in Python with **0.8914 auc score**

XGBoost model performed well on imbalanced data in R with **0.8988 auc score**

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

Model explaining with PDP plots

Partial dependence plots

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

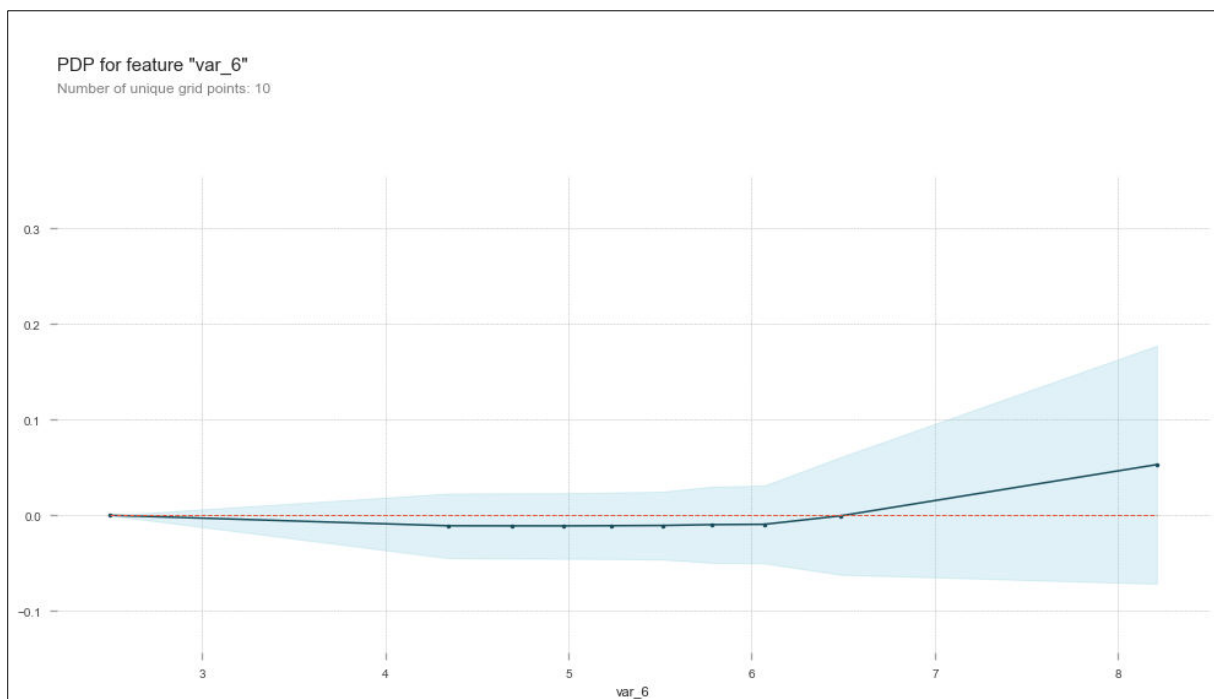
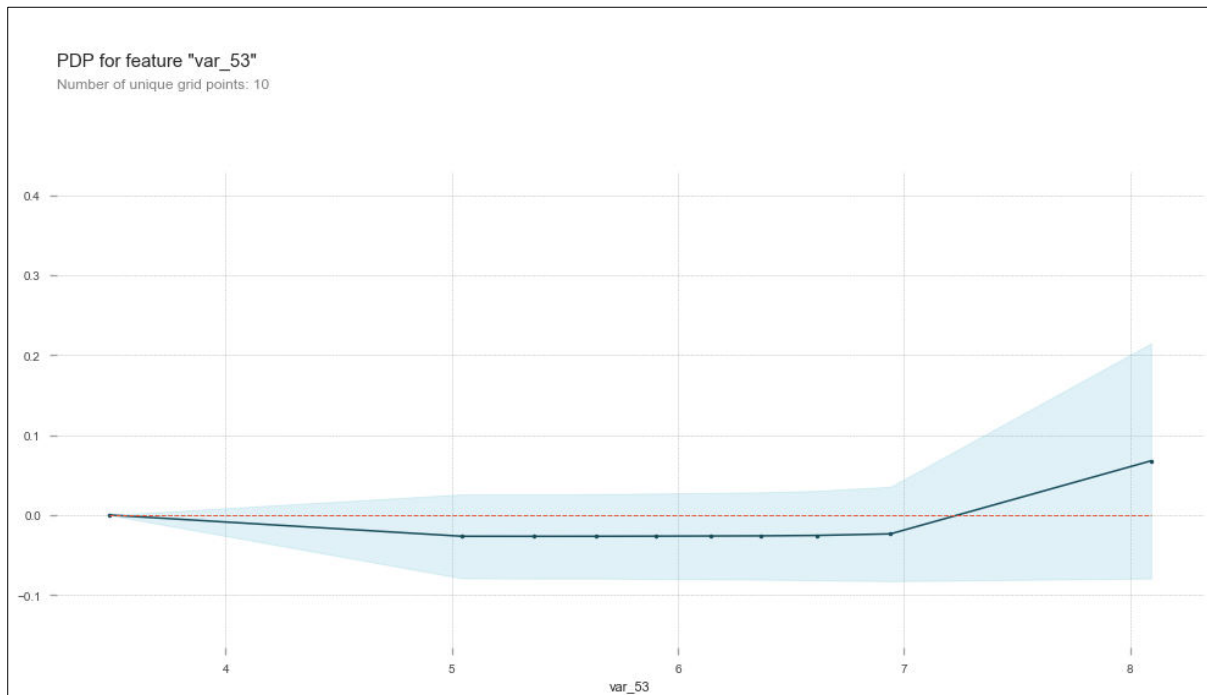
Python code

```
#Create the data we will plot 'var_53'
features=[v for v in X_valid.columns if v not in ['ID_code','target']]
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,feature='var_53')
#plot feature "var_53"
pdp.pdp_plot(pdp_data,'var_53')
plt.show()
```

Findings:

- The y_axis does not show the predictor value instead how the value changing with the change in given predictor variable.
- The blue shaded area indicates the level of confidence of 'var_53', 'var_6'

- On y-axis having a positive value means for that particular value of predictor variable it is less likely to predict the correct class and having a positive value means it has positive impact on predicting the correct class.



Appendix – Complete Python and R Code

Python Code

Exploratory Data Analysis

```

# Ignore the warnings
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

# data visualisation and manipulation
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
from scipy import stats
import seaborn as sns
import missingno as msno
import pandas_profiling as pp
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

# configure
# sets matplotlib to inline and displays graphs below the corresponding cell.
%matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid',color_codes=True)

# import the necessary modelling algos.

#classification.
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectFromModel
from imblearn.over_sampling import SMOTE
import lightgbm as lgb
import eli5
from eli5.sklearn import PermutationImportance
from sklearn import tree
import graphviz
from pdpbox import pdp, get_dataset, info_plots

from scipy.stats import randint as sp_randint

import warnings
warnings.filterwarnings('ignore')

#model selection

```

```

from sklearn.model_selection import
train_test_split,cross_validate,cross_val_score,cross_val_predict
from sklearn.model_selection import StratifiedKFold
from sklearn.pipeline import Pipeline

#Imbalanced data handling
from imblearn.over_sampling import SMOTE, RandomOverSampler
from imblearn.under_sampling import ClusterCentroids,NearMiss, RandomUnderSampler

#evaluation metrics
from sklearn.metrics import
roc_auc_score,confusion_matrix,make_scorer,classification_report,roc_curve,auc
! pip install scikit-plot
import scikitplot as skplt
from scikitplot.metrics import plot_confusion_matrix,plot_precision_recall_curve

random_state=101
np.random.seed(random_state)

```

Target classes count

```

#target classes count

target_class=train['target'].value_counts()
print('Count of target classes :\n',target_class)
#Percentage of target classes count
per_target_class=train['target'].value_counts()/len(train)*100
print('percentage of count of target classes :\n',per_target_class)
#Countplot and violin plot for target classes
fig,ax=plt.subplots(1,2,figsize=(20,5))
sns.countplot(train.target.values,ax=ax[0],palette='husl')
sns.violinplot(x=train.target.values,y=train.index.values,ax=ax[1],
palette='husl')
sns.stripplot(x=train.target.values,y=train.index.values,jitter=True,
color=' black',linewidth=0.5,size=0.5,alpha=0.5,ax=ax[1],palette='husl')
ax[0].set_xlabel('Target')
ax[1].set_xlabel('Target')
ax[1].set_ylabel('Index')

```

Distribution of train attributes

```

def plot_train_attribute_distribution(t0,t1,label1,label2,train_attributes):
i=0
sns.set_style('whitegrid')

```

```

fig=plt.figure()
ax=plt.subplots(10,10,figsize=(22,18))
for attribute in train_attributes:
    i+=1
    plt.subplot(10,10,i)
    sns.distplot(t0[attribute],hist=False,label=label1)
    sns.distplot(t1[attribute],hist=False,label=label2)
    plt.legend()
    plt.xlabel('Attribute',)
    sns.set_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})
plt.show()
Let us see first 100 train attributes
%%time
t0=train[train.target.values==0]
t1=train[train.target.values==1]
train_attributes=train.columns.values[2:102]
plot_train_attribute_distribution(t0,t1,'0','1',train_attributes)
Next 100 train attributes
train_attributes=train.columns.values[102:203]
plot_train_attribute_distribution(t0,t1,'0','1',train_attributes)

```

Distribution of test attributes

```

#importing the test dataset
test=pd.read_csv('./input/test.csv')
test.head()
#Shape of the test dataset
test.shape
def plot_test_attribute_distribution(test_attributes):
    i=0
    sns.set_style('whitegrid')
    fig=plt.figure()
    ax=plt.subplots(10,10,figsize=(22,18))
    for attribute in test_attributes:
        i+=1
        plt.subplot(10,10,i)
        sns.distplot(test[attribute],hist=False)
        plt.xlabel('Attribute',)
        sns.set_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})
    plt.show()
#Let us see first 100 test attributes
test_attributes=test.columns.values[1:101]
plot_test_attribute_distribution(test_attributes)
#Next 100 test attributes
test_attributes=test.columns.values[101:202]
plot_test_attribute_distribution(test_attributes)

```

Distribution of test attributes

```

#importing the test dataset
test=pd.read_csv('./input/test.csv')
test.head()

```

```

#Shape of the test dataset
test.shape
def plot_test_attribute_distribution(test_attributes):
i=0
sns.set_style('whitegrid')
fig=plt.figure()
ax=plt.subplots(10,10,figsize=(22,18))
for attribute in test_attributes:
i+=1
plt.subplot(10,10,i)
sns.distplot(test[attribute],hist=False)
plt.xlabel('Attribute',)
sns.set_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})
plt.show()
#Let us see first 100 test attributes
test_attributes=test.columns.values[1:101]
plot_test_attribute_distribution(test_attributes)
#Next 100 test attributes
test_attributes=test.columns.values[101:202]
plot_test_attribute_distribution(test_attributes)
Distribution of mean values in train and test dataset
%%time
#Distribution of mean values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train.columns.values[2:202]
#test attributes
test_attributes=test.columns.values[1:201]
#Distribution plot for mean values per column in train attributes
sns.distplot(train[train_attributes].mean(axis=0),color='blue',kde=True,bins=150,label='train')
#Distribution plot for mean values per column in test attributes
sns.distplot(test[test_attributes].mean(axis=0),color='green',kde=True,bins=150,label='test')
plt.title('Distribution of mean values per column in train and test dataset')
plt.legend()
plt.show() 55

```



```
#Distribution of mean values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for mean values per row in train attributes
sns.distplot(train[train_attributes].mean(axis=1),color='blue',kde=True,bins=150,label='train')
#Distribution plot for mean values per row in test attributes
sns.distplot(test[test_attributes].mean(axis=1),color='green',kde=True, bins=150, label='test')
plt.title('Distribution of mean values per row in train and test dataset')
plt.legend()
plt.show()
```

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

```
%%time
#Distribution of std values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train.columns.values[2:202]
#test attributes
test_attributes=test.columns.values[1:201]
#Distribution plot for std values per column in train attributes
sns.distplot(train[train_attributes].std(axis=0),color='red',kde=True,
bins=150,label='train')
#Distribution plot for std values per column in test attributes
sns.distplot(test[test_attributes].std(axis=0),color='blue',kde=True,bins=150,
label='test')
plt.title('Distribution of std values per column in train and test dataset')
plt.legend()
plt.show()
#Distribution of std values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for std values per row in train attributes
sns.distplot(train[train_attributes].std(axis=1),color='red',kde=True,bins=150
, label='train')
#Distribution plot for std values per row in test attributes
sns.distplot(test[test_attributes].std(axis=1),color='blue',kde=True, bins=150
, label='test')
plt.title('Distribution of std values per row in train and test dataset')
plt.legend()
```

Distribution of skewness in train and test dataset

```
%%time
#Distribution of skew values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train.columns.values[2:202]
#test attributes
test_attributes=test.columns.values[1:201]
#Distribution plot for skew values per column in train attributes
sns.distplot(train[train_attributes].skew(axis=0),color='green',kde=True, 56
```

```

bins=150,label='train')
#Distribution plot for skew values per column in test attributes
sns.distplot(test[test_attributes].skew(axis=0),color='blue',kde=True,bins=150
,label='test')
plt.title('Distribution of skewness values per column in train and test dataset')
plt.legend()
plt.show()
#Distribution of skew values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for skew values per row in train attributes
sns.distplot(train[train_attributes].skew(axis=1),color='green',kde=True,
bins=150,label='train')
#Distribution plot for skew values per row in test attributes
sns.distplot(test[test_attributes].skew(axis=1),color='blue',kde=True,
bins=150, label='test')
plt.title('Distribution of skewness values per row in train and test dataset')
plt.legend()
plt.show()
Distribution of kurtosis values in train and test dataset
%%time
#Distribution of kurtosis values per column in train and test dataset
plt.figure(figsize=(16,8))
#train attributes
train_attributes=train.columns.values[2:202]
#test attributes
test_attributes=test.columns.values[1:201]
#Distribution plot for kurtosis values per column in train attributes
sns.distplot(train[train_attributes].kurtosis(axis=0),color='blue',kde=True,
bins=150,label='train')
#Distribution plot for kurtosis values per column in test attributes
sns.distplot(test[test_attributes].kurtosis(axis=0),color='red',kde=True,
bins=150,label='test')
plt.title('Distribution of kurtosis values per column in train and test dataset')
plt.legend()
plt.show()
#Distribution of kurtosis values per row in train and test dataset
plt.figure(figsize=(16,8))
#Distribution plot for kurtosis values per row in train attributes
sns.distplot(train[train_attributes].kurtosis(axis=1),color='blue',kde=True,
bins=150,label='train')
#Distribution plot for kurtosis values per row in test attributes
sns.distplot(test[test_attributes].kurtosis(axis=1),color='red',kde=True,
bins=150, label='test')
plt.title('Distribution of kurtosis values per row in train and test dataset')
plt.legend()
plt.show() 57

```

Missing value analysis and Correlations

```
%%time
#Finding the missing values in train and test data
train_missing=train.isnull().sum().sum()
test_missing=test.isnull().sum().sum()
print('Missing values in train data :',train_missing)
print('Missing values in test data :',test_missing)
%%time
#Correlations in train attributes
train_attributes=train.columns.values[2:202]
train_correlations=train[train_attributes].corr().abs().unstack().sort_values(kind='quicksort').reset_index()
train_correlations=train_correlations[train_correlations['level_0']!=train_correlations['level_1']]
]
print(train_correlations.head(10))
print(train_correlations.tail(10))
%%time
#Correlations in test attributes
test_attributes=test.columns.values[1:201]
test_correlations=test[test_attributes].corr().abs().unstack().sort_values(kind='quicksort').reset_index()
test_correlations=test_correlations[test_correlations['level_0']!=test_correlations['level_1']]
print(test_correlations.head(10))
print(test_correlations.tail(10))
#Correlation plot
%%time
#Correlations in train data
train_correlations=train[train_attributes].corr()
train_correlations=train_correlations.values.flatten()
train_correlations=train_correlations[train_correlations!=1]
test_correlations=test[test_attributes].corr()
#Correlations in test data
test_correlations=test_correlations.values.flatten()
test_correlations=test_correlations[test_correlations!=1]
plt.figure(figsize=(20,5))
#Distribution plot for correlations in train data
sns.distplot(train_correlations, color="Red", label="train")
#Distribution plot for correlations in test data
sns.distplot(test_correlations, color="Blue", label="test")
plt.xlabel("Correlation values found in train and test")
plt.ylabel("Density")
plt.title("Correlation distribution plot for train and test attributes")
plt.legend() 58
```

Feature engineering

```
#training data
X=train.drop(columns=['ID_code','target'],axis=1)
test=test.drop(columns=['ID_code'],axis=1)
y=train['target']
#Split the training data
X_train,X_valid,y_train,y_valid=train_test_split(X,y,random_state=42)
print('Shape of X_train :',X_train.shape)
print('Shape of X_valid :',X_valid.shape)
print('Shape of y_train :',y_train.shape)
print('Shape of y_valid :',y_valid.shape)
%%time
```

#Random forest classifier

```
rf_model=RandomForestClassifier(n_estimators=10,random_state=42)
#fitting the model
rf_model.fit(X_train,y_train)
#Permutation importance
%%time
from eli5.sklearn import PermutationImportance
perm_imp=PermutationImportance(rf_model,random_state=42)
#fitting the model
perm_imp.fit(X_valid,y_valid)
%%time
```

#Important features

```
eli5.show_weights(perm_imp,feature_names=X_valid.columns.tolist(),top=200)
#partial dependence plots
%%time
```

Handling of imbalanced data

```
#Training data
X=train.drop(['ID_code','target'],axis=1)
Y=train['target']
#StratifiedKFold cross validator
cv=StratifiedKFold(n_splits=5,random_state=42,shuffle=True)
for train_index,valid_index in cv.split(X,Y):
X_train, X_valid=X.iloc[train_index], X.iloc[valid_index]
y_train, y_valid=Y.iloc[train_index], Y.iloc[valid_index]
print('Shape of X_train :',X_train.shape)
print('Shape of X_valid :',X_valid.shape)
print('Shape of y_train :',y_train.shape)
print('Shape of y_valid :',y_valid.shape)
%%time

#Logistic regression model
lr_model=LogisticRegression(random_state=42)
#fitting the lr model
lr_model.fit(X_train,y_train)
#Accuracy of the model
lr_score=lr_model.score(X_train,y_train)
print('Accuracy of the lr_model :',lr_score)
%%time

#Cross validation prediction
cv_predict=cross_val_predict(lr_model,X_valid,y_valid,cv=5)
#Cross validation score
cv_score=cross_val_score(lr_model,X_valid,y_valid,cv=5)
print('cross_val_score :',np.average(cv_score))
#Confusion matrix
cm=confusion_matrix(y_valid,cv_predict)
#Plot the confusion matrix
plot_confusion_matrix(y_valid,cv_predict,normalize=False,figsize=(15,8))
#ROC_AUC score
roc_score=roc_auc_score(y_valid,cv_predict)
print('ROC score :',roc_score)
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_valid,cv_predict)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area= %0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1],r--)
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')
plt.xlabel('False Positive Rate') 60
```

```

plt.show()
print('AUC:',roc_auc)
#Classification report
scores=classification_report(y_valid,cv_predict)
print(scores)
%%time
#Predicting the model
X_test=test.drop(['ID_code'],axis=1)
lr_pred=lr_model.predict(X_test)
print(lr_pred)
#Synthetic Minority Oversampling Technique
sm = SMOTE(random_state=42, ratio=1.0)
#Generating synthetic data points
X_smote,y_smote=sm.fit_sample(X_train,y_train)
X_smote_v,y_smote_v=sm.fit_sample(X_valid,y_valid)
#Logistic regression model for SMOTE
smote=LogisticRegression(random_state=42)
#fitting the smote model
smote.fit(X_smote,y_smote)
#Accuracy of the model
smote_score=smote.score(X_smote,y_smote)
print('Accuracy of the smote_model :',smote_score)
#Cross validation prediction
cv_pred=cross_val_predict(smote,X_smote_v,y_smote_v,cv=5)
#Cross validation score
cv_score=cross_val_score(smote,X_smote_v,y_smote_v,cv=5)
print('cross_val_score :',np.average(cv_score))
#Confusion matrix
cm=confusion_matrix(y_smote_v,cv_pred)
#Plot the confusion matrix
plot_confusion_matrix(y_smote_v,cv_pred,normalize=False,figsize=(15,8))
#ROC_AUC score
roc_score=roc_auc_score(y_smote_v,cv_pred)
print('ROC score :',roc_score)
#ROC_AUC curve
plt.figure()
false_positive_rate,recall,thresholds=roc_curve(y_smote_v,cv_pred)
roc_auc=auc(false_positive_rate,recall)
plt.title('Reciver Operating Characteristics(ROC)')
plt.plot(false_positive_rate,recall,'b',label='ROC(area= %0.3f)' %roc_auc)
plt.legend()
plt.plot([0,1],[0,1], 'r--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.ylabel('Recall(True Positive Rate)')

```

```

plt.xlabel('False Positive Rate')
plt.show()
print('AUC:',roc_auc)
#Classification report
scores=classification_report(y_smote_v,cv_pred)
print(scores)
%%time
#Predicting the model
X_test=test.drop(['ID_code'],axis=1)
smote_pred=smote.predict(X_test)
print(smote_pred)
LightGBM
#Training the model
#training data
lgb_train=lgb.Dataset(X_train,label=y_train)
#validation data
lgb_valid=lgb.Dataset(X_valid,label=y_valid)
#Selecting best hyperparameters by tuning of different parameters
params={'boosting_type': 'gbdt',
'max_depth' : -1, #no limit for max_depth if <0
'objective': 'binary',
'boost_from_average':False,
'nthread': 8,
'metric': 'auc',
'num_leaves': 100,
'learning_rate': 0.03,
'max_bin': 950, #default 255
'subsample_for_bin': 200,
'subsample': 1,
'subsample_freq': 1,
'colsample_bytree': 0.8,
'reg_alpha': 1.2, #L1 regularization(>0)
'reg_lambda': 1.2,#L2 regularization(>0)
'min_split_gain': 0.5, #>0
'min_child_weight': 1,
'min_child_samples': 5,
'is_unbalance':True,
}
num_rounds=3000
lgbm= lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_valid],
verbose_eval=100,early_stopping_rounds = 1000)
X_test=test.drop(['ID_code'],axis=1)

#predict the model
#probability predictions
lgbm_predict_prob=lgbm.predict(X_test,random_state=42,num_iteration=lgbm.best_iteration)
62

```

```

#Convert to binary output 1 or 0
lgbm_predict=np.where(lgbm_predict_prob>=0.5,1,0)
print(lgbm_predict_prob)
print(lgbm_predict)

#plot the important features
lgb.plot_importance(lgbm,max_num_features=150,importance_type="split",figsize=(20,50))

Partial dependency of features
#Create the data we will plot 'var_53'
features=[v for v in X_valid.columns if v not in ['ID_code','target']]
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,feature='var_53')
#plot feature "var_53"
pdp.pdp_plot(pdp_data,'var_53')
plt.show()

#Create the data we will plot
pdp_data=pdp.pdp_isolate(rf_model,dataset=X_valid,model_features=features,feature='var_6')
#plot feature "var_6"
pdp.pdp_plot(pdp_data,'var_6')
plt.show()

#final submission
submission_df=pd.DataFrame({'ID_code':test['ID_code'].values})
submission_df['lgbm_predict_prob']=lgbm_predict_prob
submission_df['lgbm_predict']=lgbm_predict
submission_df.to_csv('submission.csv',index=False)
submission_df.head()

```


R code

Exploratory Data Analysis

```
#Load the libraries
```

```
library(tidyverse)
```

```
library(moments)
```

```
library(DataExplorer)
```

```
library(caret)
```

```
library(Matrix)
```

```
library(mlbench)
```

```
library(caTools)
```

```
library(randomForest)
```

```
library(glmnet)
```

```
library(mlr)
```

```
library(unbalanced)
```

```
library(vita)
```

```
library(boot)
```

```
library(pROC)
```

```
library(DMwR)
```

```
library(ROSE)
```

```
library(yardstick)
```

```
#loading the train data
```

```
train<-read.csv('../input/train.csv')
```

```
head(train)
```

```
#Dimension of train data
```

```
dim(train)
```

```
#Summary of the dataset
```

```
str(train)
```

```
#convert to factor
```

```
train$target<-as.factor(train$target)
```

Target classes count in train data

```
require(gridExtra)
```

```
#Count of target classes
```

```
table(train$target)
```

```
#Percentatge counts of target classes
```

```
table(train$target)/length(train$target)*100
```

```
#Bar plot for count of target classes
```

```
plot1<-ggplot(train,aes(target))+theme_bw()+geom_bar(stat='count',fill='lightgreen')
```

```
#Violin with jitter plots for target classes
```

```
plot2<-ggplot(train,aes(x=target,y=1:nrow(train)))+theme_bw()+geom_violin(fill='lightblue')+  
64
```

```

facet_grid(train$target)+geom_jitter(width=0.02)+labs(y='Index')
grid.arrange(plot1,plot2, ncol=2)
Distribution of train attributes
#Distribution of train attributes from 3 to 102
for (var in names(train)[c(3:102)]){
  target<-train$target
  plot<-ggplot(train, aes(x=train[[var]],fill=target)) +
  geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()
  print(plot)
}
#Distribution of train attributes from 103 to 202
for (var in names(train)[c(103:202)]){
  target<-train$target
  plot<-ggplot(train, aes(x=train[[var]], fill=target)) +
  geom_density(kernel='gaussian') + ggtitle(var)+theme_classic()
  print(plot)
}
Distribution of test attributes
#loading test data
test<-read.csv('../input/test.csv')
head(test)
#Dimension of test dataset
dim(test)
#Distribution of test attributes from 2 to 101
plot_density(test[,c(2:101)], ggtheme = theme_classic(),
geom_density_args = list(color='blue'))
#Distribution of test attributes from 102 to 201
plot_density(test[,c(102:201)], ggtheme = theme_classic(),
geom_density_args = list(color='blue'))
Distribution of mean values in train and test dataset
#Applying the function to find mean values per row in train and test data.
train_mean<-apply(train[, -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=TRUE,color='blue')+theme_classic()+ 65

```

```

#Distribution of mean values per row in test data
geom_density(data=test[,-c(1)],aes(x=test_mean),kernel='gaussian',
show.legend=TRUE,color='green')+
labs(x='mean values per row',title="Distribution of mean values per row in
train and test dataset")
#Applying the function to find mean values per column in train and test data.
train_mean<-apply(train[,-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='green')+
labs(x='mean values per column',title="Distribution of mean values per row
in train and test dataset")
Distribution of standard deviation in train and test dataset
#Applying the function to find standard deviation values per row in train
and test data.
train_sd<-apply(train[,-c(1,2)],MARGIN=1,FUN=sd)
test_sd<-apply(test[,-c(1)],MARGIN=1,FUN=sd)
ggplot()+
#Distribution of sd values per row in train data
geom_density(data=train[,-c(1,2)],aes(x=train_sd),kernel='gaussian',
show.le gend=TRUE,color='red')+theme_classic()+
#Distribution of mean values per row in test data
geom_density(data=test[,-c(1)],aes(x=test_sd),kernel='gaussian',
show.legend=TRUE,color='blue')+
labs(x='sd values per row',title="Distribution of sd values per row in
train and test dataset")
#Applying the function to find sd values per column in train and test data.
train_sd<-apply(train[,-c(1,2)],MARGIN=2,FUN=sd)
test_sd<-apply(test[,-c(1)],MARGIN=2,FUN=sd)
ggplot()+
#Distribution of sd values per column in train data
geom_density(aes(x=train_sd),kernel='gaussian',show.legend=TRUE,color='red')+
theme_classic()+
#Distribution of sd values per column in test data
geom_density(aes(x=test_sd),kernel='gaussian',show.legend=TRUE,color='blue')+
labs(x='sd values per column',title="Distribution of std values per column
in train and test dataset")
Distribution of skewness values in train and test dataset
#Applying the function to find skewness values per row in train and test data. 66

```

```

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='green')+theme_classic()+
#Distribution of skewness values per column in test data
geom_density(aes(x=test_skew),kernel='gaussian',show.legend=TRUE,
color='blue') +labs(x='skewness values per row',title="Distribution of
skewness values per row in train and test dataset")
#Applying the function to find skewness values per column in train and
test data.
train_skew<-apply(train[,-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='green')+theme_classic()+
#Distribution of skewness values per column in test data
geom_density(aes(x=test_skew),kernel='gaussian',show.legend=TRUE,
color='blue')+labs(x='skewness values per column',title="Distribution of
skewness values per column in train and test dataset")
Distribution of kurtosis values in train and test dataset
#Applying the function to find kurtosis values per row in train and test data.
train_kurtosis<-apply(train[,-c(1,2)],MARGIN=1,FUN=kurtosis)
test_kurtosis<-apply(test[,-c(1)],MARGIN=1,FUN=kurtosis)
ggplot()+
#Distribution of sd values per column in train data
geom_density(aes(x=train_kurtosis),kernel='gaussian',show.legend=TRUE,
color='blue')+theme_classic()+
#Distribution of sd values per column in test data
geom_density(aes(x=test_kurtosis),kernel='gaussian',show.legend=TRUE,
color='red')+labs(x='kurtosis values per 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 sd values per column in train data
geom_density(aes(x=train_kurtosis),kernel='gaussian',show.legend=TRUE,
color='blue')+theme_classic()+
#Distribution of sd values per column in test data
geom_density(aes(x=test_kurtosis),kernel='gaussian',show.legend=TRUE,
color='red')+
labs(x='kurtosis values per column',title="Distribution of kurtosis values
per column in train and test dataset") 67

```

Missing value analysis and Correlations

```
#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
#Correlations in train data
train$target<-as.numeric(train$target)
train_correlations<-cor(train[,c(2:202)])
train_correlations
#Correlations in test data
test_correlations<-cor(test[,c(2:201)])
test_correlations
```

Feature Engineering

```
#Split the training data
train_index<-sample(1:nrow(train),0.75*nrow(train))
train_data<-train[train_index,]
valid_data<-train[-train_index,]
dim(train_data)
dim(valid_data)
#Training the Random forest classifier
set.seed(2732)
train_data$target<-as.factor(train_data$target)
mtry<-floor(sqrt(200))
tuneGrid<-expand.grid(.mtry=mtry)
rf<-randomForest(target~.,train_data[,c(1)],mtry=mtry,ntree=10,
importance=TRUE)
#Variable importance
VarImp<-importance(rf,type=2)
VarImp
#Partial dependence plot
#We will plot "var_81"
partialPlot(rf,valid_data[,c(1,2)],valid_data$var_81,xlab='var_81')
#We will plot "var_12"
partialPlot(rf,valid_data[,c(1,2)],valid_data$var_12,xlab='var_12') 68
```

Handling of imbalanced data

```
#Split the data using CreateDataPartition
set.seed(689)
train.index<-createDataPartition(train$target,p=0.8,list=FALSE)
train.data<-train[train.index,]
valid.data<-train[-train.index,]
dim(train.data)
dim(valid.data)
#training dataset
set.seed(682)
X_t<-as.matrix(train.data[-c(1,2)])
y_t<-as.matrix(train.data$target)
#validation dataset
X_v<-as.matrix(valid.data[-c(1,2)])
y_v<-as.matrix(valid.data$target)
#test data
test<-as.matrix(test[, -c(1)])
#Logistic regression model
set.seed(667)
lr_model <- glmnet(X_t, y_t, family = "binomial")
summary(lr_model)
#Cross validation prediction
set.seed(8909)
cv_lr <- cv.glmnet(X_t,y_t,family = "binomial", type.measure = "class")
cv_lr
#Minimum lambda
cv_lr$lambda.min
#plot the auc score vs log(lambda)
plot(cv_lr)
#Model performance on validation dataset
set.seed(5363)
cv_predict.lr<-predict(cv_lr,X_v,s = "lambda.min", type = "class")
cv_predict.lr
#Confusion matrix
set.seed(689)
#actual target variable
target<-valid.data$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor
cv_predict.lr<-as.factor(cv_predict.lr)
confusionMatrix(data=cv_predict.lr,reference=target) 69
```

```

#ROC_AUC score and curve
set.seed(892)
cv_predict.lr<-as.numeric(cv_predict.lr)
roc(data=valid.data[,-c(1,2)],response=target,predictor=cv_predict.lr,
auc=TRUE,plot=TRUE)
#predict the model
set.seed(763)
lr_pred<-predict(lr_model,test,type='class')
lr_pred
#Random Oversampling Examples(ROSE)
set.seed(699)
#train.data$target<-as.factor(train.data$target)
train.rose <- ROSE(target~., data =train.data[,-c(1)],seed=32)$data
table(train.rose$target)
valid.rose <- ROSE(target~., data =valid.data[,-c(1)],seed=32)$data
table(valid.rose$target)
#Logistic regression model
set.seed(462)
lr_rose <-glmnet(as.matrix(train.rose),as.matrix(train.rose$target), family = "binomial")
summary(lr_rose)
#Cross validation prediction
set.seed(473)
cv_rose = cv.glmnet(as.matrix(valid.rose),as.matrix(valid.rose$target),
family = "binomial", type.measure = "class")
cv_rose
#Minimum lambda
cv_rose$lambda.min
#plot the auc score vs log(lambda)
plot(cv_rose)
#Model performance on validation dataset
set.seed(442)
cv_predict.rose<-predict(cv_rose,as.matrix(valid.rose),s = "lambda.min",
type = "class")
cv_predict.rose
#Confusion matrix
set.seed(478)
#actual target variable
target<-valid.rose$target
#convert to factor
target<-as.factor(target)
#predicted target variable
#convert to factor 70

```

```

cv_predict.rose<-as.factor(cv_predict.rose)
confusionMatrix(data=cv_predict.rose,reference=target)
#ROC_AUC score and curve
set.seed(843)
cv_predict.rose<-as.numeric(cv_predict.rose)
roc(data=valid.rose[,-c(1,2)],response=target,predictor=cv_predict.rose,
auc=TRUE,plot=TRUE)
#predict the model
set.seed(6543)
rose_pred<-predict(lr_rose,test,type='class')
rose_pred

```

XGBoost

```

library(data.table)
library(caret)
library(xgboost)
library(pROC)
#setting working directory
setwd("F:/EdwisorVanusha/Project/Santander")
#loading the train data
train=read.csv("train.csv")
test=read.csv("test.csv")
#Let's check the dimension of train and test sets. Also check what are the variables that are
there in train but not in test. Also let's have a look at the head of the data sets
dim(train) ; dim(test) ; setdiff(colnames(train) , colnames(test)) ; head(train) ; head(test)
#It seems like the variables have no names as such and the only variable that is missing in the
test set is the target column which we need to predict. Let's check the sample submission file.
We will check if the ids in sample sub and the test file are in same order or not as well.
#training and testing data
trainX=as.matrix(train[,-c(1,2)])
trainY=as.matrix(train$target)
testX=as.matrix(test[,-c(1)])
# preparing XGB matrix
dtrain <- xgb.DMatrix(data = as.matrix(trainX), label = as.matrix(trainY))
# parameters
params <- list(booster = "gbtree",
               objective = "binary:logistic",
               eta=0.02,
               #gamma=80,
               max_depth=2,
               min_child_weight=1,
               subsample=0.5,
               colsample_bytree=0.1,
               scale_pos_weight = round(sum(!trainY) / sum(trainY), 2))
# CV
set.seed(123)
xgbcv <- xgb.cv(params = params,

```



```

        data = dtrain,
        nrounds = 30000,
        nfold = 5,
        showsd = F,
        stratified = T,
        print_every_n = 100,
        early_stopping_rounds = 500,
        maximize = T,
        metrics = "auc")
cat(paste("Best iteration:", xgbcv$best_iteration))
# train final model
set.seed(123)
xgb_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = xgbcv$best_iteration,
  print_every_n = 100,
  maximize = T,
  eval_metric = "auc")
#view variable importance plot
imp_mat <- xgb.importance(feature_names = colnames(trainX), model = xgb_model)
xgb.plot.importance(importance_matrix = imp_mat[1:30])
#prediction
pred_sub <- predict(xgb_model, newdata=as.matrix(testX), type="response")
#Submission
submission <- read.csv("submission.csv")
submission$target <- pred_sub
write.csv(submission, file="submission_XgBoost.csv", row.names=F)

```

References

1. <https://www.kaggle.com/>
2. <http://rprogramming.net/>
3. <https://medium.com/>
4. <https://stackoverflow.com/>
5. <https://www.rdocumentation.org/>
6. <https://www.analyticsvidhya.com/blog/>