SANTANDER CUSTOMER TRANSACTION PREDICTION

MICHAEL AJITH

21th JUNE 2019

Contents

Chapter 1

Introduction

1.1 Problem Statement:

Our client "Santander", is on a mission to help people and businesses prosper. They are always looking for ways to help their customers to understand their financial health and identify which products and services might help them to achieve their monetary goals. Our client wants to identify which customers will make a specific transaction in the future, irrespective of the amount of money transacted.

We need to develop a model with good accuracy to help our client to identify the customers which will continue a healthy business with our client in future.

1.2 Data:

We have classification problem with 200 continuous variables with one binary variable. It is a classification problem. In this problem we have train data with "202" variables including target variable and around "200000" observations and test data with "201" variables without target variable and around "200000" observations

Number of attributes:

- **ID_codes** – it just tell us about the number of transactions
- **Var_0 to Var_199** – they all are continuous and independent variables.

Dependent Variable:

Target- Our main Target variable, it is our binary variable with 1 (yes) and 0 (no)

Overview of top five observation of the train and test dataset

```
train.head()
```

	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	va
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	8
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	8
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	8
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	10
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	9

5 rows × 202 columns

```
test.head()
```

	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

Chapter 2

Methodology

2.1 Data Preprocessing: (Exploratory Data Analysis)

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues.

2.1.1 Understanding the Data:

Overview of our data and datatypes

```
train.shape, test.shape
```

```
((200000, 202), (200000, 201))
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Columns: 202 entries, ID_code to var_199
dtypes: float64(200), int64(1), object(1)
memory usage: 308.2+ MB
```

We have ID_code as object , target as int and var_0 to var_199 as float.

Analysis of Numerical Variables for train and test dataset:

Looking at summary for our numerical variables for train dataset

```
#Overview of numerical values in train  dataset
train.describe()
```

	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.00000
mean	0.100490	10.679914	-1.627622	10.715192	6.796529	11.078333	-5.065317	5.408949	16.545850	0.28416
std	0.300653	3.040051	4.050044	2.640894	2.043319	1.623150	7.863267	0.866607	3.418076	3.33263
min	0.000000	0.408400	-15.043400	2.117100	-0.040200	5.074800	-32.562600	2.347300	5.349700	-10.50550
25%	0.000000	8.453850	-4.740025	8.722475	5.254075	9.883175	-11.200350	4.767700	13.943800	-2.31780
50%	0.000000	10.524750	-1.608050	10.580000	6.825000	11.108250	-4.833150	5.385100	16.456800	0.39370
75%	0.000000	12.758200	1.358625	12.516700	8.324100	12.261125	0.924800	6.003000	19.102900	2.93790
max	1.000000	20.315000	10.376800	19.353000	13.188300	16.671400	17.251600	8.447700	27.691800	10.15130

8 rows × 201 columns

Looking at summary for our numerical variables for train dataset

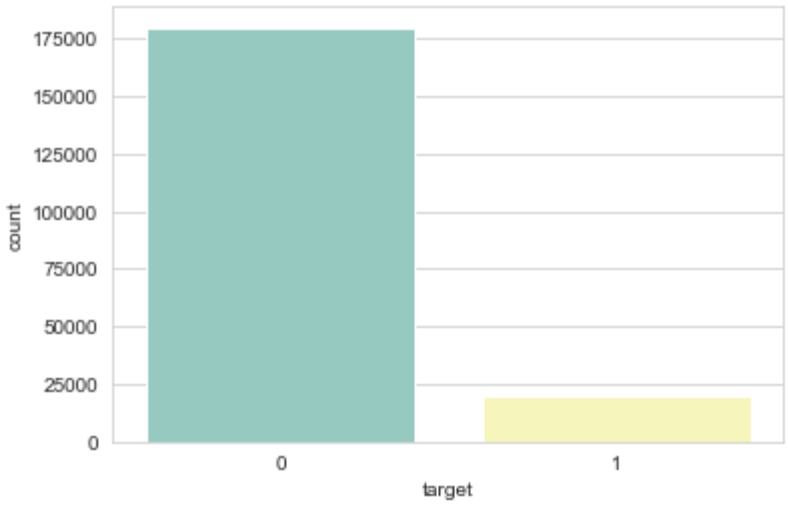
```
#Overview of numerical values in test  dataset
test.describe()
```

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.00000
mean	10.658737	-1.624244	10.707452	6.788214	11.076399	-5.050558	5.415164	16.529143	0.277135	7.56940
std	3.036716	4.040509	2.633888	2.052724	1.616456	7.869293	0.864686	3.424482	3.333375	1.23186
min	0.188700	-15.043400	2.355200	-0.022400	5.484400	-27.767000	2.216400	5.713700	-9.956000	4.24330
25%	8.442975	-4.700125	8.735600	5.230500	9.891075	-11.201400	4.772600	13.933900	-2.303900	6.62380
50%	10.513800	-1.590500	10.560700	6.822350	11.099750	-4.834100	5.391600	16.422700	0.372000	7.63200
75%	12.739600	1.343400	12.495025	8.327600	12.253400	0.942575	6.005800	19.094550	2.930025	8.58482
max	22.323400	9.385100	18.714100	13.142000	16.037100	17.253700	8.302500	28.292800	9.665500	11.00360

8 rows × 200 columns

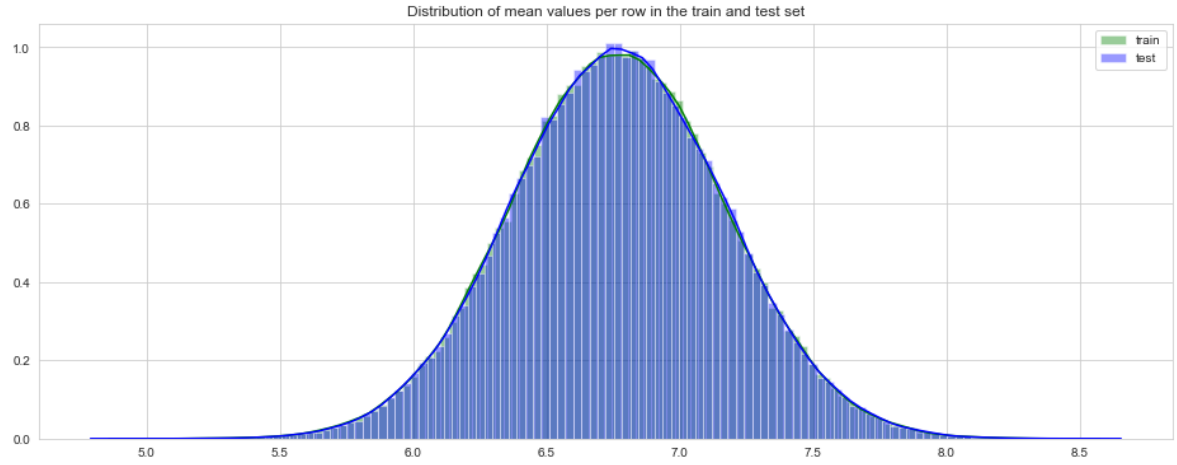
Distribution of the target variable of the problem

```
<matplotlib.axes._subplots.AxesSubplot at 0x72435ab7f0>
```



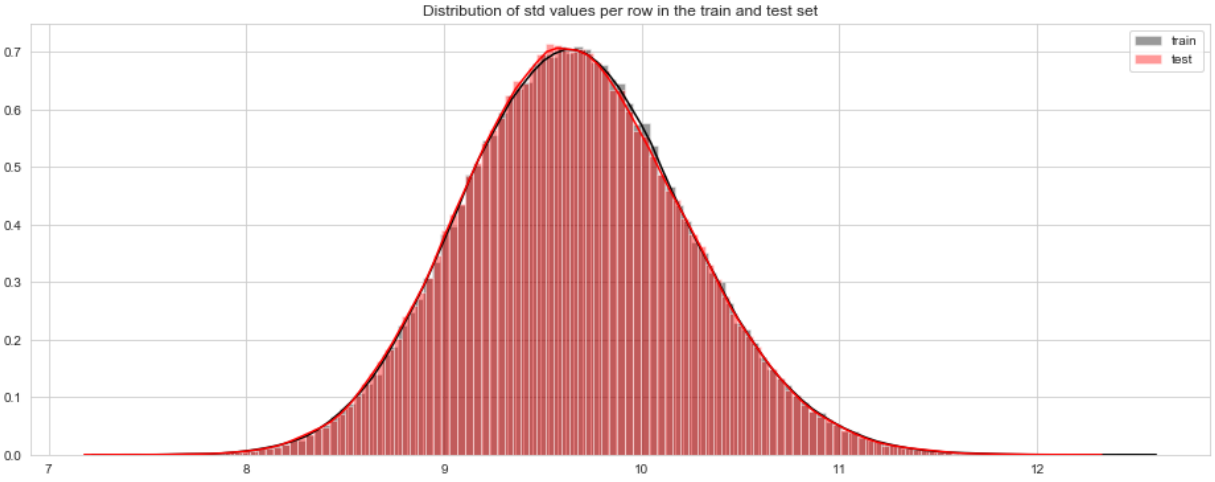
Distribution of Mean , Standard Deviation, Min and Max of our data

Distribution of mean values per row in the train and test set



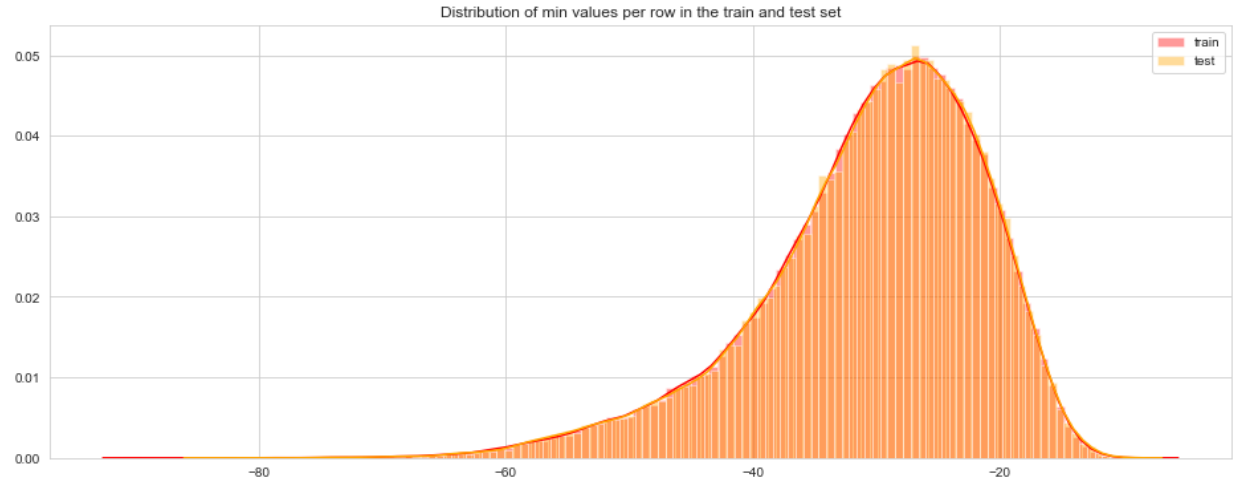
Distribution of mean values per column in the train and test set

Distribution of std values per row in the train and test set

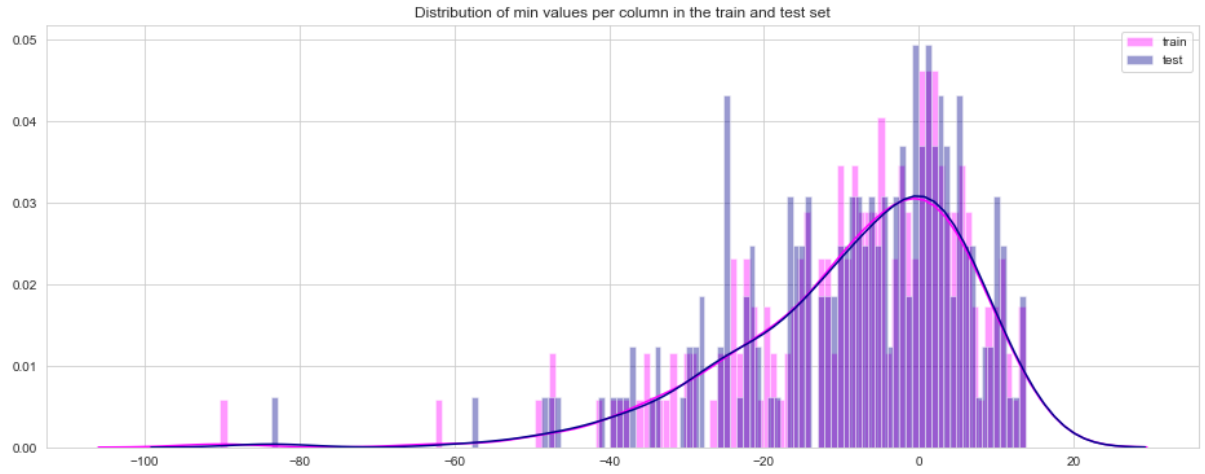


Distribution of std values per column in the train and test set

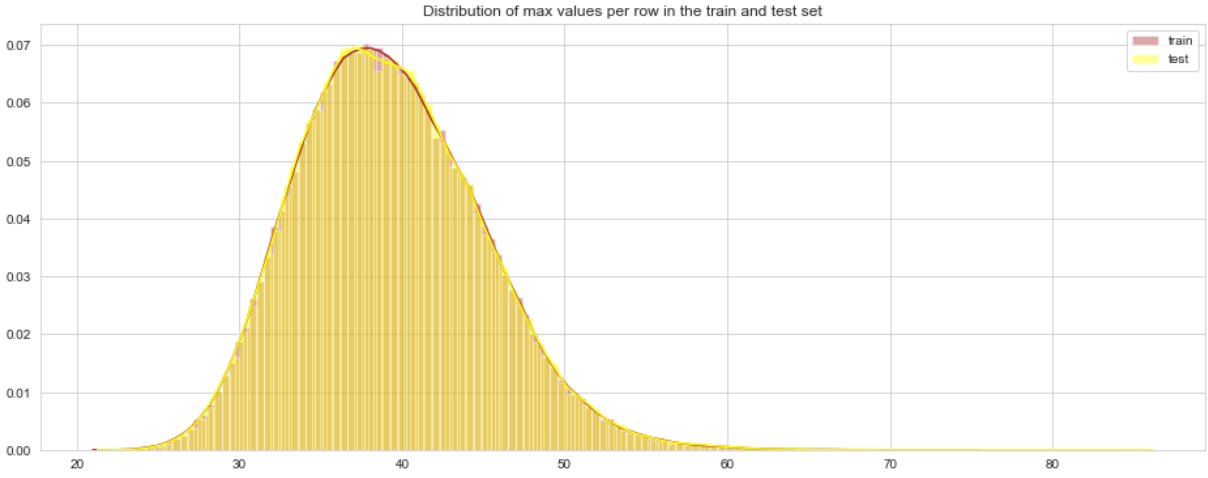
Distribution of min values per row in the train and test se



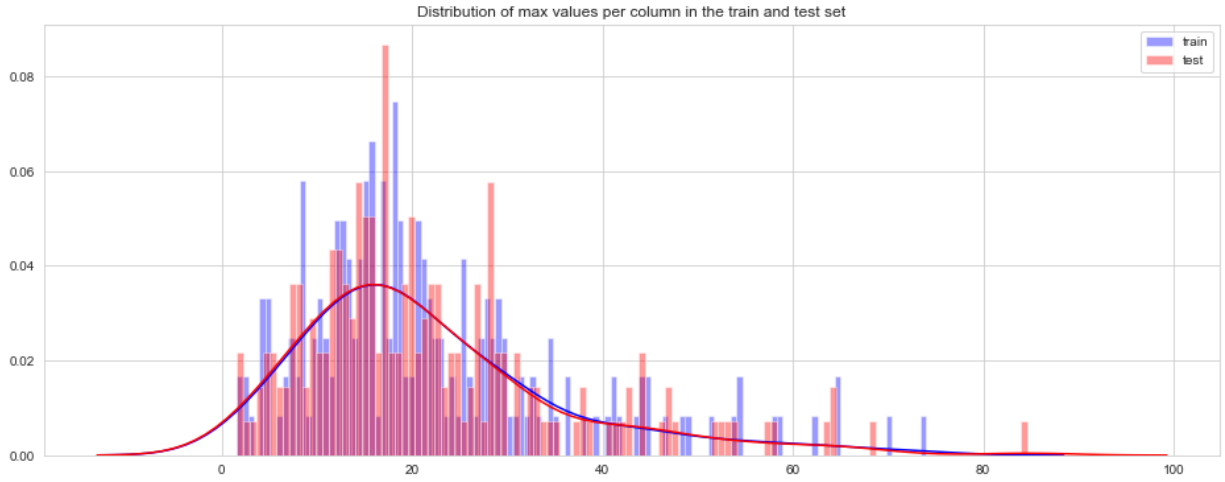
Distribution of min values per column in the train and test set



Distribution of max values per row in the train and test set



Distribution of max values per column in the train and test set



2.1.2 Outlier Analysis:

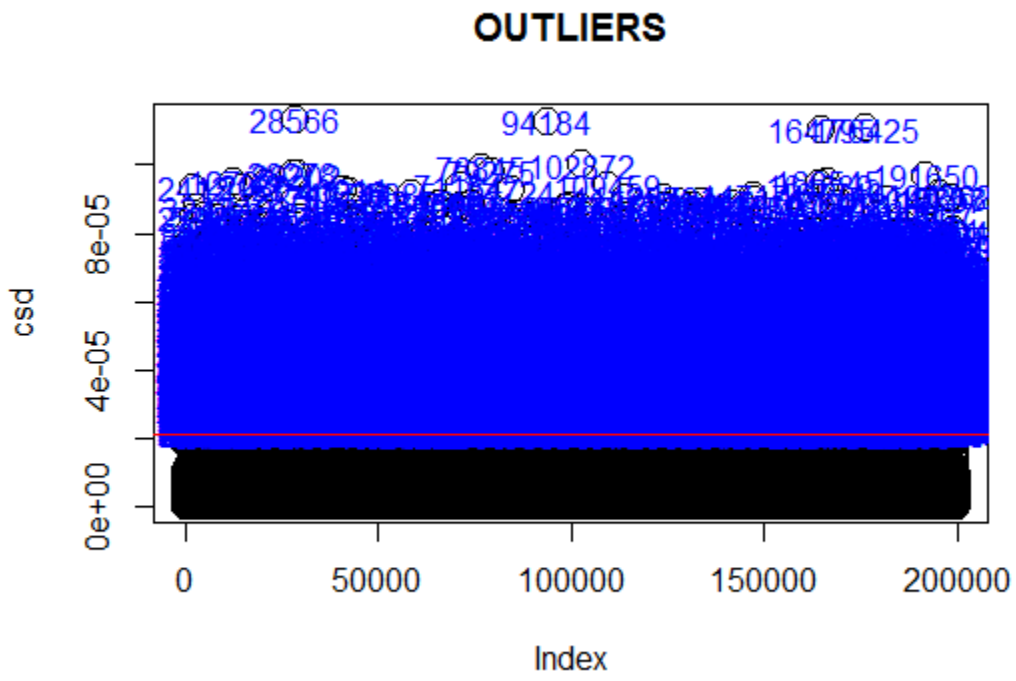
Outliers, are attributed to a rare chance and may not necessarily be fully explainable, Outliers in data can distort predictions and affect the accuracy, if you don't detect and handle them.

The contentious decision to consider or discard an outlier needs to be taken at the time of building the model. Outliers can drastically bias/change the fit estimates and predictions. It is left to the best judgement of the analyst to decide whether treating outliers is necessary and how to go about it.

Treating or altering the outlier/extreme values in genuine observations is not a standard operating procedure. If a data point (or points) is excluded from the data analysis, this should be clearly stated on any subsequent rep

Coming into the problem we have around 200 independent variables for which using the boxplot method is not ideal solutions as it will consume too much time and money. As, we have multivariate data, we can use cook's distance method for detecting the outliers.

In cook's distance method for each observation it will measures the changes in y-hat for all observations with and without the presence of observation, so to know how much it will impact the y. It will consider all values which is greater than 4 times the mean as an outlier or influential value.



Analysis:

All the blue color points are the outliers in our data. We have detected all the outlier. Now, we need to replace them by using the capping method.

Capping method is a very simple and fast method to replace the outliers. It will consider the value as an outlier which lie outside the 1.5*IQR.

It will use the IQR method to replace all the outlier in the data.

Lower limit of acceptable range = Q1 – 1.5*(Q3-Q1)

Upper limit of acceptable range = Q3 + 1.5*(Q3-Q1)

This analysis is done in R and in the project we have rejected the outliers because outliers may carry information and can increase accuracy in models

2.1.3 Missing value analysis:

Checking missing values in our dataset

```
> #checking the count of all the na's in data
> sum(is.na(cust))
[1] 0
>
```

```
#missing data in train dataset
missing_data(train)
```

	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	v
Total	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
Percent	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
Types	object	int64	float64	float64	float64	float64	float64	float64	float64	float64	...	float64	float64	float64	float64	float64	float64	float64	

3 rows × 202 columns

```
#missing data in test dataset
missing_data(test)
```

	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	v
Total	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
Percent	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	
Types	object	float64	float64	float64	float64	float64	float64	float64	float64	float64	...	float64	float64	float64	float64	float64	float64	float64	

3 rows × 201 columns

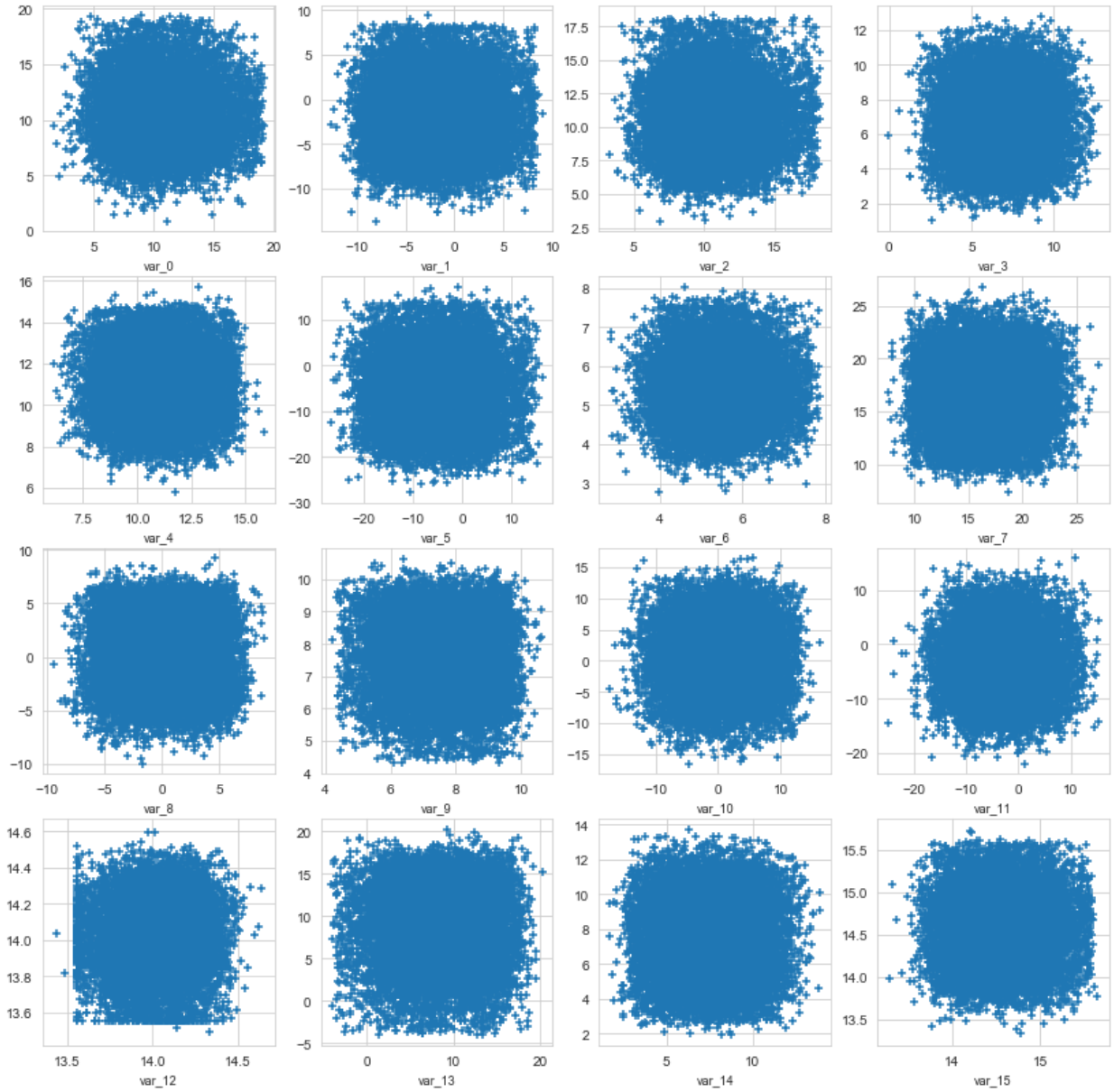
Analysis:

In our dataset we are lucky that we don't have any missing values. In case we have missing values then we should impute it using different method mean, median, KNN

2.1.4 Feature selection:

In Feature selection you have to check whether there is multicolinearity between our continuous independent variables.

Overview of scatter plot between of our continuous variables



Analysis:

It is too hard to run the entire 200 continuous independent variables . so we have selected the best 25 features to build the models in the python but in r we have take all the 200 continuous independent

variables for model building except in random forest taken only two variables(var_170, var_99) as per Boruta method suggestions

2.1.6 Feature Scaling:

We are using the normalization to scale the data because our data is not normally distributed.

We scale our data because it helps to normalize the data within a particular range. Sometimes, it also helps in speeding up the calculations in an algorithm.

Done feature scaling(normalization) on two important variables only in the project

After normalization process our data will look this:

```
> head(cust1)
  cust.target cust.var_99 cust.var_170
1           0   0.3129724    0.3118487
2           0   0.6453427    0.6416500
3           0   0.5870114    0.2373175
4           0   0.7468318    0.2353519
5           0   0.6435300    0.5107674
6           0   0.5963583    0.2676588
>
```

2.2 Modeling

We will now build our model, before proceeding terms used in our codes:

- train_X_new: containing all independent variables used for training model
- train_Y: containing target variable used for training model
- test_X_new: containing independent variable used for testing model

2.2.1 GridsearchCV and Accuracy metrics

Before building models on our dataset, we would like to explore two things:

- GridSearchCV
- Accuracy metrics

GridsearchCV : (Hyperparameter tuning):

Hyperparameter are the parameters which we pass as argument to our building function, like kernel, criterion, n_estimators etc. So to get best values of these gridserchcv is used. In this technique, we make list of these different parameters and then gridsearchcv build model for every combination of these parameters and then check crossvalidation score and based on score it gives the best combination of hyperparameters.

And then we can build our model with the values of hyperparameter given by GridSearchCV.

This is called performance tuning and we would use this to tune our model.

Accuracy metrics

Evaluating your machine learning algorithm is an essential part of any project. Your model may give you satisfying results when evaluated using a metric say accuracy_score but may give poor results when evaluated against other metrics such as logarithmic_loss or any other such metric. Most of the times we use classification accuracy to measure the performance of our model, however it is not enough to truly judge our model

We have different metrics for classification

- Classification Accuracy
- AUC, or Area Under Curve
- Recall
- Precision

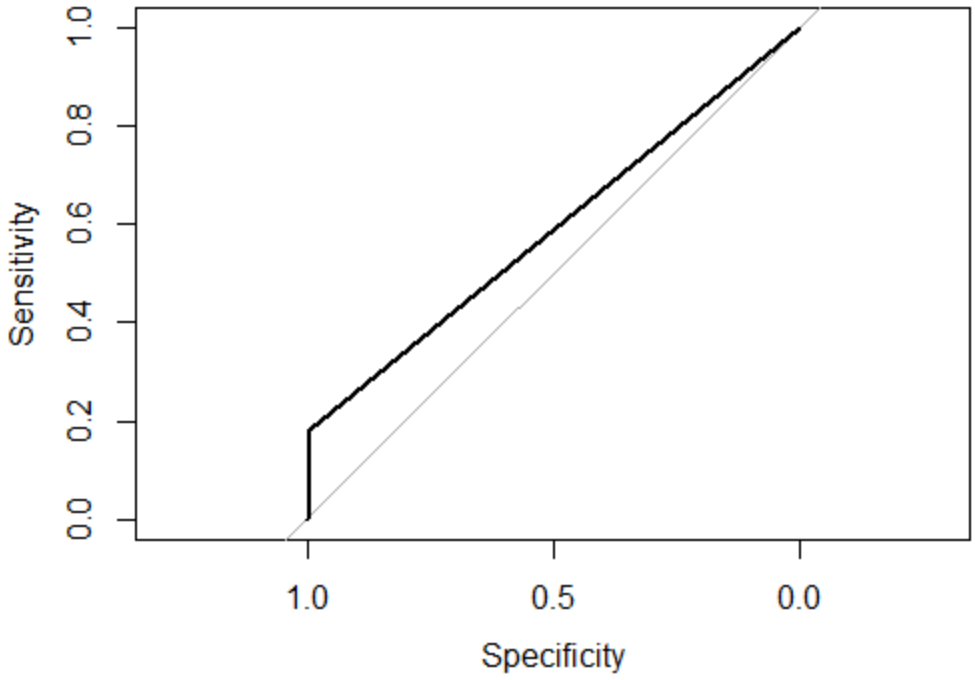
We can use any of them for comparing our model, but we will use **AUC**. Best possible score is 1.0 (perfect prediction) and less is worse.

2.2.2 Building models

Models and performance of models:

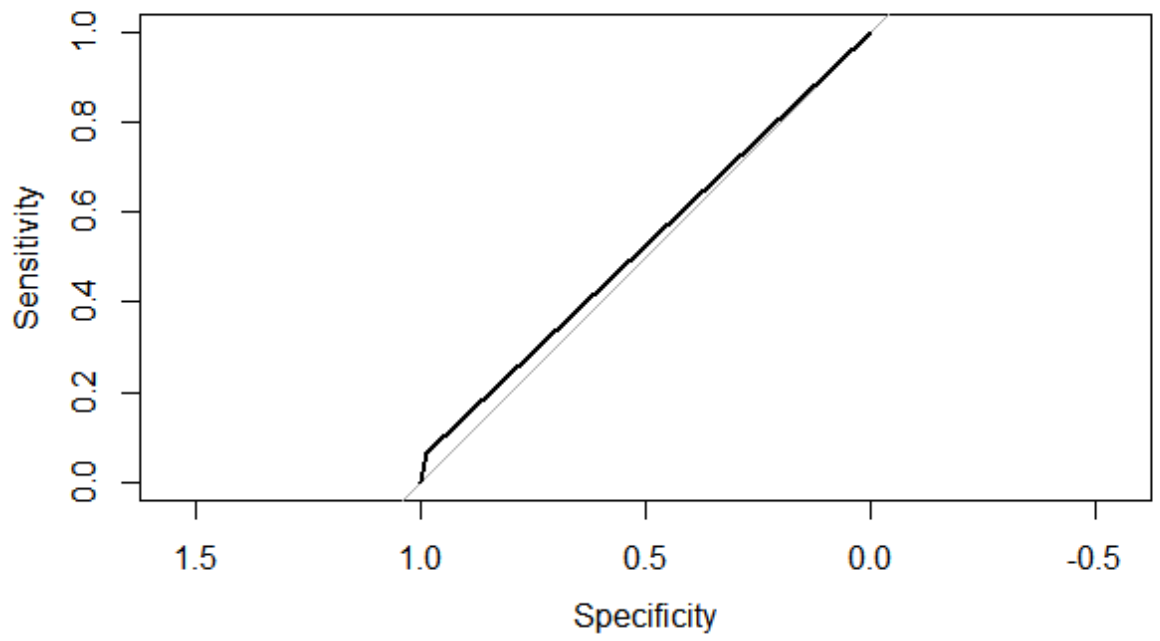
We will now build one by one all models and will check performance of our model and then at the will decide final model which we should use for our project.

Logistic Regression:



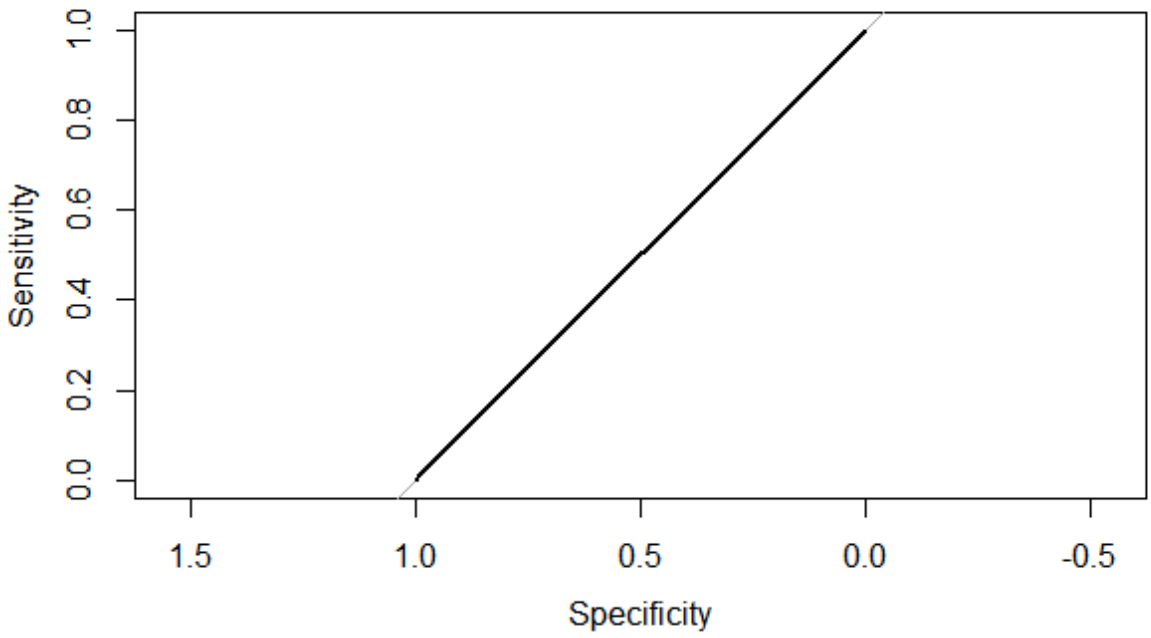
```
Console D:/MBA REF/edwisor/project/sanders/
> #Recall,Precision and Accuracy
> recall<-53668/(53668+4919)
> recall
[1] 0.9160394
> precision<- 53668/(53668+321)
> precision
[1] 0.9940543
> accuracy<- (53668+1092)/60000
> accuracy
[1] 0.9126667
> #Recall for logistic regression is 91.6%
> #Precision  for logistic regression is 99.4%
> #Accuracy for logistic regression is 91.2%
> # model performance
> plot.roc(testing$target, pred1)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> auc(roc(testing$target, pred1)) # provides AUC value which is ~0.5879
Setting levels: control = 0, case = 1
Setting direction: controls < cases
Area under the curve: 0.5879
>
```

Decision Tree Classifier:



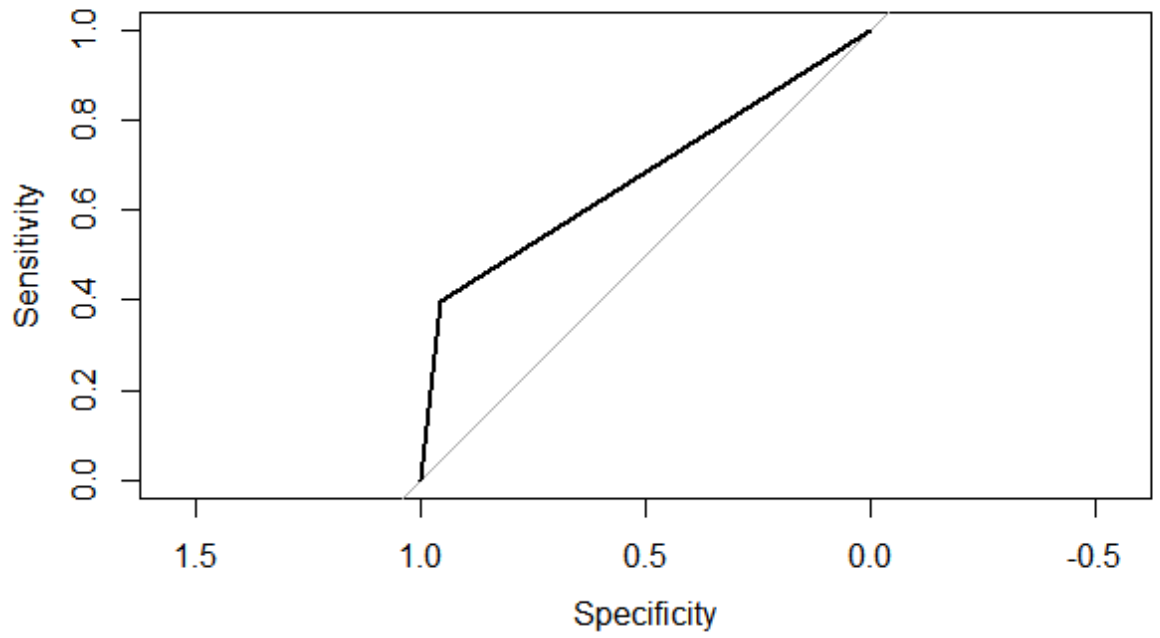
```
D:/MBA REF/edwisor/project/sanders/
> #Recall,Precision and Accuracy
> recall<-53386/(53386+5624)
> recall
[1] 0.9046941
> precision<- 53386/(53386+603)
> precision
[1] 0.9888311
> accuracy<- (53386+387)/60000
> accuracy
[1] 0.8962167
> # model performance
> pred2<-as.numeric(pred2)
> plot.roc(testing$target, pred2)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> auc(roc(testing$target, pred2)) # provides AUC value which is ~0.5266
Setting levels: control = 0, case = 1
Setting direction: controls < cases
Area under the curve: 0.5266
>
```

Random Forest Classifier:



```
Console D:/MBA REF/edwisor/project/sanders/
> #Recall,Precision and Accuracy
> recall<-35683/(35683+3998)
> recall
[1] 0.8992465
> precision<- 35683/(35683+275)
> precision
[1] 0.9923522
> accuracy<- (35683+44)/40000
> accuracy
[1] 0.893175
> # model performance
> pred3 <- as.numeric(pred3)
> plot.roc(test_data$cust.target, pred3)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> auc(roc(test_data$cust.target, pred3))# provides AUC value which is ~0.5023
Setting levels: control = 0, case = 1
Setting direction: controls < cases
Area under the curve: 0.5015
>
```

XGBClassifier:



```
D:/MBA REF/edwisor/project/sanders/
        'Positive' Class : 0

> #Recall,Precision and Accuracy
> recall<-51701/(51701+3610)
> recall
[1] 0.9347327
> precision<- 51701/(51701+2288)
> precision
[1] 0.957621
> accuracy<- (51701+2401)/60000
> accuracy
[1] 0.9017
> plot.roc(testing$target, pred4)
Setting levels: control = 0, case = 1
Setting direction: controls < cases
> auc(roc(testing$target, pred4)) # provides AUC value which is ~0.6708
Setting levels: control = 0, case = 1
Setting direction: controls < cases
Area under the curve: 0.6785
>
```

AUC Result for Random Forest, Decision Tree, Logistic Regression and XGBoost models in python

	Model	Score
0	XGB	78.68
1	Logistic Regression	75.31
2	Random_Forest	69.36
3	DT	56.06

Analysis:

Models has been builded in both R and python and have used Random Forest, Decision Tree, Logistic Regression and XGBoost models in both R and python and calculated accuracy based on AUC value. Out of four models Logistic Regression and XGBoost has better accuracy so lets tune the parameter of those two models to increase accuracy.

2.2.3 Hyperparameter tuning:

We have to tune our two best models (Logistic regression and and XGBClassifier). So we will tune our model for dataset . With the help of hyperparameter tuning we would find optimum values for parameter used in function and would increase our accuracy.

Logistic Regression Hyperparameter tuning:

Tuning Logistic Regression model parameters

Logistic Regression Hyperparameter tuning

```
#Grid Search to Optimize the Parameters
grid={"C":np.logspace(-2,3,7), "penalty":["l1","l2"]}# l1 lasso l2 ridge
logreg1=LogisticRegression()
logreg_hp=GridSearchCV(logreg1,grid,cv=3,verbose=0)
logreg_hp.fit(train_X_new,train_Y)

print("tuned hpyerparameters :(best parameters) ",logreg_hp.best_params_)
print("accuracy :",logreg_hp.best_score_)
```

```
  FutureWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: Future
ed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: Future
ed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)

C:\Users\admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: Future
ed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: Future
ed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: Future
ed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
tuned hpyerparameters :(best parameters)  {'C': 3.1622776601683795, 'penalty': 'l2'}
accuracy : 0.902065
```

```
#Modeling with the Best Parameters from GridSearch
logreg2=LogisticRegression(C=1000,penalty="l2")#Grid Search Result  C:1000 l2 Accuracy :0.902095
logreg2.fit(train_X_new,train_Y)
logreg2_score = round(cross_val(logreg2,train_X_new, train_Y) * 100, 2)
print("Accuracy for logistic regression after doing a GridSearchCV: ", logreg2_score)
```

```
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning:
to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning:
to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning:
to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning:
to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning:
to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
C:\Users\admin\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning:
to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

```
Accuracy for logistic regression after doing a GridSearchCV:  75.36
```

Analysis:

From above result (on tuned parameter), we have increased our model AUC score 75.31% to 75.36%.

XGBClassifier Hyperparameter tuning:

Tuning XGBClassifier model parameters

XGBoost Hyperparameter tuning

```
#Grid Search to Optimize the Parameters
params = {
        'min_child_weight': [1, 5, 10],
        'gamma': [0.5, 1, 1.5, 2, 5],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5]
        }
xgbm_hp= GridSearchCV(estimator =  xgbm, param_grid = params, scoring='roc_auc',n_jobs=1,iid=False, cv=3)
xgbm_hp.fit(train_X_new,train_Y)
xgbm_hp.best_params_, xgbm_hp.best_score_
```

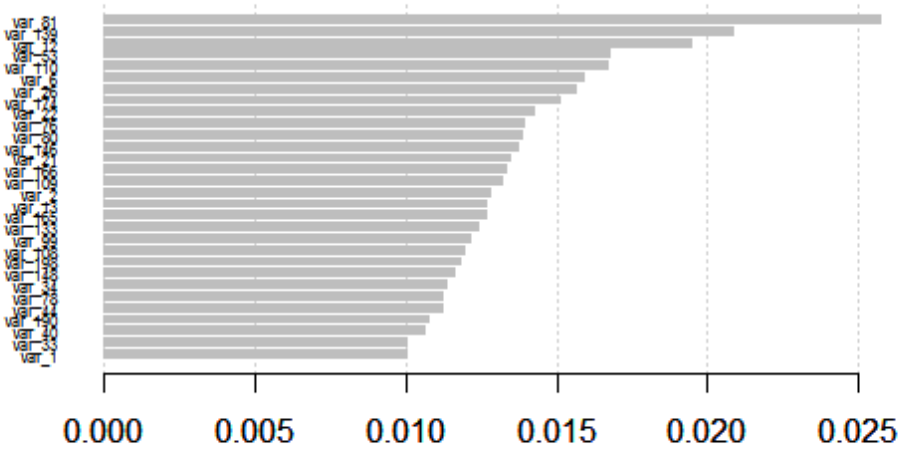
```
#Training XGB Model with Best Parameters from GridSearch
xgbm2 = xgb.XGBClassifier(silent=1,
                      scale_pos_weight=1,
                     learning_rate=0.01,
                      colsample_bytree = 0.4,
                      subsample = 0.8,
                      objective='binary:logistic',
                      n_estimators=1000,
                      reg_alpha = 0.3,
                      max_depth=4,
                      gamma=10)
xgbm2.fit(train_X_new,train_Y)
xgb2_score = round(cross_val(xgbm2,train_X_new, train_Y) * 100, 2)
print("Accuracy for XGBoost  after doing a GridSearchCV: ", xgb2_score)
```

```
Accuracy for XGBoost  after doing a GridSearchCV:  78.68
```

Analysis:

From above result (on tuned parameter), there is no increase in accuracy it still stays 78.68%

Important variables according to xgboost classifier



Chapter 3

Conclusion

3.1 Final Model and Training Dataset

Final Dataset:

- Took whole dataset in R except random forest(var_170, var_99) and top 25 feature independent variables for training in python.

Final Model:

- Use XGBClassifier model using training set.
- Do hyperparameter tuning for training set.
- Build XGBClassifier model with tuned parameters(it hasn't increased accuracy for this features but may increase if we increase the features as 50, 100 or 200.

3.2 End Notes

- All the analysis results and plot diagrams are based on R and python. In R, result would not be exact same but would be almost same.
- Outlier too may have information so it should be treated well by gaining domain knowledge and experimenting with model building and checking performance. So we considered outliers as an information to this model

While doing hyperparameter tuning, after getting result of parameter, do again with finer values of parameter and so on. At the end we will get optimum values.

Complete Python code:

```
# coding: utf-8

# # <center> **SANTANDER CUSTOMER TRANSACTION PREDICTION** </center>



# Importing Packages

import numpy as np

import pandas as pd

import os

import seaborn as sns

import matplotlib.pyplot as plt

#Importing packages for cross_validation

from sklearn.model_selection import cross_val_score


#for modeling

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear_model import LogisticRegression

import xgboost as xgb

from collections import Counter


#for feature selection

from sklearn.feature_selection import SelectKBest, f_classif
```

```
## Grid search cross validation
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
#set working directory-
os.chdir("D:/MBA REF/edwisor/project/sanders")

#check current working directory-
os.getcwd()
train = pd.read_csv('train.csv')
test=pd.read_csv('test.csv')


# #                          Exploratory Data Analysis

# ##  Understanding the data
train.shape, test.shape
train.info()
train.head()
test.head()
test.info()


# ## Outlier analysis
#The number of values in train and test set is the same.
```

```
#Let's plot the scatter plot for train and test set for few of the features.
def plot_feature_scatter(df1, df2, features):
    i = 0
    sns.set_style('whitegrid')
    plt.figure()
    fig, ax = plt.subplots(4,4,figsize=(14,14))

    for feature in features:
        i += 1
        plt.subplot(4,4,i)
        plt.scatter(df1[feature], df2[feature], marker='+')
        plt.xlabel(feature, fontsize=9)
    plt.show();
#Overview  of 5% the data. On x axis we show train values and on the y axis we show the test values
features = ['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',
          ]
plot_feature_scatter(train[::20],test[::20], features)
#the distribution of target value in train dataset
sns.countplot(train['target'], palette='Set3')



# ## Distribution of Mean and Standard Deviation
#Distribution of mean values per row in the train and test set
plt.figure(figsize=(16,6))
```

```
features = train.columns.values[2:202]
plt.title("Distribution of mean values per row in the train and test set")
sns.distplot(train[features].mean(axis=1),color="green", kde=True,bins=120, label='train')
sns.distplot(test[features].mean(axis=1),color="blue", kde=True,bins=120, label='test')
plt.legend()
plt.show()
#Distribution of mean values per column in the train and test set
plt.figure(figsize=(16,6))
plt.title("Distribution of mean values per column in the train and test set")
sns.distplot(train[features].mean(axis=0),color="magenta",kde=True,bins=120, label='train')
sns.distplot(test[features].mean(axis=0),color="darkblue", kde=True,bins=120, label='test')
plt.legend()
plt.show()
#Distribution of std values per row in the train and test set
plt.figure(figsize=(16,6))
plt.title("Distribution of std values per row in the train and test set")
sns.distplot(train[features].std(axis=1),color="black", kde=True,bins=120, label='train')
sns.distplot(test[features].std(axis=1),color="red", kde=True,bins=120, label='test')
plt.legend()
plt.show()
#Distribution of std values per column in the train and test set
plt.figure(figsize=(16,6))
plt.title("Distribution of std values per column in the train and test set")
sns.distplot(train[features].std(axis=0),color="blue",kde=True,bins=120, label='train')
sns.distplot(test[features].std(axis=0),color="green", kde=True,bins=120, label='test')
```

```
plt.legend()

plt.show()



# ## Distribution of Min and Max

#Distribution of min values per row in the train and test set

plt.figure(figsize=(16,6))

features = train.columns.values[2:202]

plt.title("Distribution of min values per row in the train and test set")

sns.distplot(train[features].min(axis=1),color="red", kde=True,bins=120, label='train')

sns.distplot(test[features].min(axis=1),color="orange", kde=True,bins=120, label='test')

plt.legend()

plt.show()

#Distribution of min values per column in the train and test set

plt.figure(figsize=(16,6))

features = train.columns.values[2:202]

plt.title("Distribution of min values per column in the train and test set")

sns.distplot(train[features].min(axis=0),color="magenta", kde=True,bins=120, label='train')

sns.distplot(test[features].min(axis=0),color="darkblue", kde=True,bins=120, label='test')

plt.legend()

plt.show()

#Distribution of max values per row in the train and test set

plt.figure(figsize=(16,6))

features = train.columns.values[2:202]

plt.title("Distribution of max values per row in the train and test set")
```

```
sns.distplot(train[features].max(axis=1),color="brown", kde=True,bins=120, label='train')
sns.distplot(test[features].max(axis=1),color="yellow", kde=True,bins=120, label='test')
plt.legend()
plt.show()
#Distribution of max values per column in the train and test set
plt.figure(figsize=(16,6))
features = train.columns.values[2:202]
plt.title("Distribution of max values per column in the train and test set")
sns.distplot(train[features].max(axis=0),color="blue", kde=True,bins=120, label='train')
sns.distplot(test[features].max(axis=0),color="red", kde=True,bins=120, label='test')
plt.legend()
plt.show()



# ## Missing value analysis
#Function to find missing values in train and test data
def missing_data(data):
    total = data.isnull().sum()
    percent = (data.isnull().sum()/data.isnull().count()*100)
    tt = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
    types = []
    for col in data.columns:
        dtype = str(data[col].dtype)
        types.append(dtype)
    tt['Types'] = types
```

```
    return(np.transpose(tt))
#missing data in train dataset
missing_data(train)
#missing data in test dataset
missing_data(test)
#Overview of numerical values in train  dataset
train.describe()
#Overview of numerical values in test  dataset
test.describe()



# ## Model Building
#Defining Target Variable and Test Variables for Modeling
train_X= train.drop(columns=['target','ID_code'],axis=1)
train_Y=train['target']
test_X= test.drop(columns=['ID_code'],axis=1)



# ## Feature Selection for the models
#Selecting the Features to be Considered for Modeling
selector = SelectKBest(f_classif, k=25)
X_new=selector.fit_transform(train_X, train_Y)
mask = selector.get_support(indices=True)

# Names of the selected columns
```

```
colname = train_X.columns[mask]


#Redefining Train and Test Data Using the Selected Features
train_X_new = train_X[colname]
test_X_new = test_X[colname]



# ## Random Forest,  Decision Tree,  Logistic Regression  and  XGBoost models
#Defining Random Forest, Decision Tree, Logistic Regression and XGBoost models
RFmodel=RandomForestClassifier(random_state=1)
DTmodel=DecisionTreeClassifier(random_state=1)
logreg = LogisticRegression()
xgbm = xgb.XGBClassifier(silent=True, scale_pos_weight=1,learning_rate=0.01,
                         colsample_bytree = 0.4,
                         subsample = 0.8,
                         objective='binary:logistic',
                         n_estimators=1000,
                         reg_alpha = 0.3,
                         max_depth=4,
                         gamma=10)
#Training the Models
RFmodel.fit(train_X_new,train_Y)
DTmodel.fit(train_X_new,train_Y)
logreg.fit(train_X_new,train_Y)
xgbm.fit(train_X_new,train_Y)
```

```
#defining a function that can be called to calculate cross validation score for each model
def cross_val(X,x,y):
    scores = cross_val_score(X, x, y, cv=5, scoring = "roc_auc")
    return scores.mean()
#Calculating Cross Validation Score based on ROC_AUC by calling the cross_val function defined before
RF_score = round(cross_val(RFmodel,train_X_new, train_Y) * 100, 2)
DT_score =round(cross_val(DTmodel,train_X_new, train_Y) * 100, 2)
logreg_score = round(cross_val(logreg,train_X_new, train_Y) * 100, 2)
xgb_score = round(cross_val(xgbm,train_X_new, train_Y) * 100, 2)



# ## AUC Result for Random Forest, Decision Tree, Logistic Regression and XGBoost models
#Tabulating the CV Scores based on ROC_AUC for Different Models
results = pd.DataFrame({'Model': ['XGB',  'Logistic Regression', 'Random_Forest','DT'
            ],'Score': [xgb_score , logreg_score,RF_score,DT_score]})
results = results.sort_values(['Score'], ascending=[False])
results.head()



# #         Hyperparameter Tuning


# ## Logistic Regression         Hyperparameter tuning
#Grid Search to Optimize the Parameters
grid={"C":np.logspace(-2,3,7), "penalty":["l1","l2"]}# l1 lasso l2 ridge
```

```
logreg1=LogisticRegression()
logreg_hp=GridSearchCV(logreg1,grid,cv=3,verbose=0)
logreg_hp.fit(train_X_new,train_Y)

print("tuned hpyerparameters :(best parameters) ",logreg_hp.best_params_)
print("accuracy :",logreg_hp.best_score_)
#Modeling with the Best Parameters from GridSearch
logreg2=LogisticRegression(C=1000,penalty="l2")#Grid Search Result  C:1000 l2 Accuracy :0.902095
logreg2.fit(train_X_new,train_Y)
logreg2_score = round(cross_val(logreg2,train_X_new, train_Y) * 100, 2)
print("Accuracy for logistic regression after doing a GridSearchCV: ", logreg2_score)



# ## XGBoost        Hyperparameter tuning
#Grid Search to Optimize the Parameters
params = {
      'min_child_weight': [1, 5, 10],
      'gamma': [0.5, 1, 1.5, 2, 5],
      'subsample': [0.6, 0.8, 1.0],
      'colsample_bytree': [0.6, 0.8, 1.0],
      'max_depth': [3, 4, 5]
      }
xgbm_hp= GridSearchCV(estimator =  xgbm, param_grid = params, scoring='roc_auc',n_jobs=1,iid=False,
cv=3)
xgbm_hp.fit(train_X_new,train_Y)
xgbm_hp.best_params_, xgbm_hp.best_score_
```

```
#Training XGB Model with Best Parameters from GridSearch
xgbm2 = xgb.XGBClassifier(silent=1,
                  scale_pos_weight=1,
                 learning_rate=0.01,
                  colsample_bytree = 0.4,
                  subsample = 0.8,
                  objective='binary:logistic',
                  n_estimators=1000,
                  reg_alpha = 0.3,
                  max_depth=4,
                  gamma=10)
xgbm2.fit(train_X_new,train_Y)
xgb2_score = round(cross_val(xgbm2,train_X_new, train_Y) * 100, 2)
print("Accuracy for XGBoost  after doing a GridSearchCV: ", xgb2_score)
```

Complete R code:

```
#remove all the objects stored

rm(list=ls())


#set current working directory

setwd("D:/MBA REF/edwisor/project/sanders")


#Current working directory

getwd()


# importing all required library

Packages <- c("ggplot2","ggpubr","randomForest","caret", "class", "e1071",

          "rpart", "DMwR","usdm","dplyr","caTools",

          "C50","RODBC","outliers","car","Boruta","Metrics","ggthemes",

          "DataCombine","inTrees","pROC","xgboost")


lapply(Packages, library, character.only = TRUE)


# Reading/Loading the csv file

cust<-read.csv("train.csv", header =  TRUE , stringsAsFactors = FALSE)



# ***************************  Exploratory Data Analysis ***************************
```

```
# ******************** Understanding the data  **********************************

# checking datatypes of all columns

class(cust)

dim(cust)

head(cust)

names(cust)

str(cust)

summary(cust)

#checking the target variable of the dataset

hist(cust$target , col = "red")



#removing one variable  "ID_Code" the target variable from the training set

cust <- cust[, -1, drop = FALSE]

# ********************** Outlier Analysis *************************

#We are using the cooks Distance

#first we will build the model and on that bases we will build the model

mod <- lm(target ~., data = cust)

csd<-cooks.distance(mod)
```

```
plot(csd, pch = 1, cex = 2, main = "influential points")
abline(h = 4*mean(csd, na.rm = T), col = "red")
text(x=1:length(csd)+1, y=csd, labels = ifelse(csd >4*mean(csd, na.rm = T), names(csd), ""), col = "blue")
influential <- as.numeric(names(csd)[(csd >4*mean(csd, na.rm = T))])
head(cust[influential,])

#we doing outliers test using car package
car:: outlierTest(mod)
#it show that 94184 in this row it has the most extreme values

#imputation of the outliers we are using the capping function
x<- as.data.frame(cust)
caps <- data.frame(apply(cust,2, function(x){
  quantiles <- quantile(x, c(0.25, 0.75))
  x[x < quantiles[1]] <- quantiles[1]
  x[x > quantiles [2]] <- quantiles[2]
}))

caps

# **********  Missing value analysis **********

#checking the count of all the na's in data
sum(is.na(cust))
```

```
# ************************  Feature Selection  ***************************

#feature selection of the data using the Boruta method

#Boruta method

set.seed(123)

btrain <- Boruta(target~., data = cust, doTrace = 2)

print(btrain)



#Boruta performed 99 iterations in 7.526 hrs.

#2 attributes confirmed important: var_170, var_99;

#196 attributes confirmed unimportant: var_0, var_1, var_10, var_100, var_101 and 191 more;

#2 tentative attributes left: var_114, var_92;


#Ploting graph

plot(btrain, xlab = "", xaxt = "n")

lz<-lapply(1:ncol(btrain$ImpHistory),function(i)

  btrain$ImpHistory[is.finite(btrain$ImpHistory[,i]),i])

names(lz) <- colnames(btrain$ImpHistory)

Labels <- sort(sapply(lz,median))

axis(side = 1,las=2,labels = names(Labels),

     at = 1:ncol(btrain$ImpHistory), cex.axis = 0.7)


#tentative variable test

final.boruta <- TentativeRoughFix(btrain)
```

```
print(final.boruta)

getSelectedAttributes(final.boruta, withTentative = F)

boruta.df <- attStats(final.boruta)


# **************************  Feature Scaling  ****************************

#Feature scaling only for  Random forest

#selecting variables

cust1 <- data.frame(cust$target, cust$var_99, cust$var_170)

colnames(cust1)

View(cust1)


#normalization method

cnames = c("cust.var_99", "cust.var_170")

for (i in cnames) {

  cust1 [,i] = (cust1[,i] - min(cust1[,i]))/ (max(cust1[,i] - min(cust1[,i])))


}

head(cust1)


# ****************************  Building models  ******************************

rm(list= ls()[!(ls() %in% c('cust','cust1'))])

set.seed(321)


intrain <- createDataPartition(y = cust$target, p = 0.7, list = FALSE)

# ?createDataPartition part of caret package
```

```
training<- cust[intrain,]
hist(training$target)
length(training$target)
length(which(training$target==0))
length(which(training$target==1))

testing <- cust[-intrain,]
hist(testing$target)
length(testing$target)
length(which(testing$target==0))
length(which(testing$target==1))
#View(testing)
# model preparation. Exclude ID column from dataset
trainingExID <- training[]
head(trainingExID)
View(trainingExID)

# **************************  Logistic Regression  ****************************

# model preparation using Logistic regression
model1 <- glm(trainingExID$target~., data = trainingExID, family = binomial(link = 'logit'))
summary(model1)
model1
# prediction on test data
pred1 <- predict(model1, testing[,-1])
```

```
head(pred1) # gives prediction probabilities

pred1 <- as.numeric(pred1 > 0.5)

summary(pred1)

hist(pred1)



#error matrix


conmatrix <- table(testing$target, pred1)

confusionMatrix(conmatrix)


#Recall,Precision and Accuracy

recall<-53668/(53668+4919)

recall

precision<- 53668/(53668+321)

precision

accuracy<- (53668+1092)/60000

accuracy

#Recall for logistic regression is 91.6%

#Precision  for logistic regression is 99.4%

#Accuracy for logistic regression is 91.2%

# model performance

plot.roc(testing$target, pred1)

auc(roc(testing$target, pred1)) # provides AUC value which is ~0.5879
```

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

# model preparation using Decision Tree

model2 <-C5.0(as.factor(trainingExID$target)~., data = trainingExID ,trails=100,rules = TRUE)

write(capture.output(summary(model2)), "summary_cust.text")

pred2 <- predict(model2, testing[,-1])

#error matrix

conmatrix <- table(testing$target, pred2)

confusionMatrix(conmatrix)

#Recall,Precision and Accuracy

recall<-53386/(53386+5624)

recall

precision<- 53386/(53386+603)

precision

accuracy<- (53386+387)/60000

accuracy

#Recall for Decision Tree is 90.4%

#Precision  for Decision Tree is 98.8%

#Accuracy for Decision Tree is 89.62%
```

```
# model performance
pred2<-as.numeric(pred2)
plot.roc(testing$target, pred2)
auc(roc(testing$target, pred2)) # provides AUC value which is ~0.5266

# ********************************  Random Forest  ************************
# model preparation using Random Forest
cust1<- as.data.frame(cust1)

train_index = sample(1:nrow(cust1), 0.8*nrow(cust1))
train_data = cust1[train_index,]
test_data = cust1[-train_index,]

model3 <- randomForest(as.factor(cust.target)~., train_data, importance = TRUE)
treelist <- RF2List(model3)
rules <- extractRules(treelist, train_data[,-1])
rules[5,]

readrule <- presentRules(rules, colnames(train_data))
head(readrule)

rulematrix <- getRuleMetric(rules, train_data[,-1], train_data$cust.target)
head(rulematrix)

#error matrix
```

```
pred3 <- predict(model3, test_data[,-1])

confmt <- table(test_data$cust.target, pred3)

confusionMatrix(confmt)

summary(model3)

#Recall,Precision and Accuracy
recall<-35683/(35683+3998)
recall

precision<- 35683/(35683+275)
precision
accuracy<- (35683+44)/40000
accuracy
#Recall for Random Forest is 89.9%
#Precision  for Random Forest is 99.2%
#Accuracy for Random Forest is 89.3%

# model performance
pred3 <- as.numeric(pred3)
plot.roc(test_data$cust.target, pred3)
auc(roc(test_data$cust.target, pred3))# provides AUC value which is ~0.5015
```

```
# ********************************  XGBoost ******************************


# model preparation using XGBoost


model4 <- xgboost(data = as.matrix(training[,-1:-2]), label = as.matrix(training$target), max_depth = 2,
eta = 1, nthread = 6, nrounds = 3000, maximize = T, print_every_n = 100, objective = "binary:logistic")

summary(model4)

model4


# prediction on test data

pred4 <- predict(model4,as.matrix(testing[,-1:-2]))

head(pred4) # gives prediction probabilities

pred4 <- as.numeric(pred4 > 0.5)

summary(pred4)

hist(pred4)

length(which(pred4 == 1))

length(which(pred4 == 0))

hist(pred4)


#error matrix


conmatrix <- table(testing$target, pred4)

confusionMatrix(conmatrix)

#Recall,Precision and Accuracy
```

```
recall<-51701/(51701+3610)

recall

precision<- 51701/(51701+2288)

precision

accuracy<- (51701+2401)/60000

accuracy

#Recall for  XGBoost is 93.4%

#Precision  for XGBoost is 95.7%

#Accuracy for  XGBoost is 90.9%


# model performance


plot.roc(testing$target, pred4)

auc(roc(testing$target, pred4)) # provides AUC value which is ~0.6785


#************************ Hyperparameter tuning ************************************


#***************************** Tuning XGBoost  ****************************


# preparing XGB matrix

dtrain <- xgb.DMatrix(data = as.matrix(training[,-1:-2]), label = as.matrix(training$target))

# 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(!training$target) / sum(training$target), 2))
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))

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(training[,-1:-2]), model = xgb_model)
xgb.plot.importance(importance_matrix = imp_mat[1:30])



# prediction on test data
pred5 <- predict(xgb_model, as.matrix(testing[,-1:-2]))
head(pred5) # gives prediction probabilities
pred5 <- as.numeric(pred5 > 0.5)
summary(pred4)
length(which(pred5 == 1))
length(which(pred5 == 0))
hist(pred5)

#error matrix

conmatrix <- table(testing$target, pred5)
confusionMatrix(conmatrix)
```

```
#Recall,Precision and Accuracy

recall<-45847/(45847+1355)

recall


precision<- 45847/(45847+8142)

precision

accuracy<- (45847+4656)/60000

accuracy

#Recall for Tunned XGBoost is 97.1%

#Precision  for Tunned XGBoost is 84.9%

#Accuracy for Tunned XGBoost is 84.17%


# model performance

plot.roc(testing$target, pred5)

auc(roc(testing$target, pred5)) # provides AUC value which is ~0.8119
```

References:

- https://learning.edwisor.com/
- https://www.udemy.com/machinelearning/
- https://medium.com/greyatom/why-how-and-when-to-scale-your-features-4b30ab09db5e
- https://towardsdatascience.com/data-pre-processing-techniques-you-should-know-8954662716d6
- https://towardsdatascience.com/machine-learning-general-process-8f1b510bd8af
- https://github.com/topics/machine-learning

THANK YOU