PROJECT

SANTANDER CUSTOMER TRANSACTION PREDICTION

SUBHADEEP CHOWDHURY

CONTENTS:-

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

1.1 PROBLEM STATEMENT :-

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

1.2 DATA:-

Our task is to build classification models, which will classify the customers who will make a specific transaction in the future.
Given below is a sample of the data set that we are using to predict.

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	...	var_190	var_191	var_192	var_193	var_194	var_195	var_196	var_
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	...	4.4354	3.9642	3.1364	1.6910	18.5227	-2.3978	7.8784	8.5
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	8.7
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	8.2
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	10.2
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

The data set consist of 200000 observation with 202 variables.

2. METHODOLOGY :-

2.1 PRE PROCESSING :-

Any predictive modelling requires that we look at the data before we start modelling. 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 visualising the data through graphs and plots. This is often called as Exploratory Data Analysis. To start this process we will first try and look at all the distributions of the variables.

2.1.1. Missing Value Analysis:-

```
bd_train=pd.read_csv("/Users/subhadeep/Downloads/train (1).csv")
bd_test=pd.read_csv("/Users/subhadeep/Downloads/test (1).csv")
```

```
bd_train.isnull().sum()
```

```
bd_test.isnull().sum()
```

First of all we have loaded the training and testing dataset under bd_train and bd_test data frame.

Then we have checked missing values in both train and test data.

There are no missing values in both the dataset.

```
bd_train_new=bd_train.drop(['target','ID_code'],axis=1)
```

```
print(bd_train_new.shape)
```

```
(200000, 200)
```

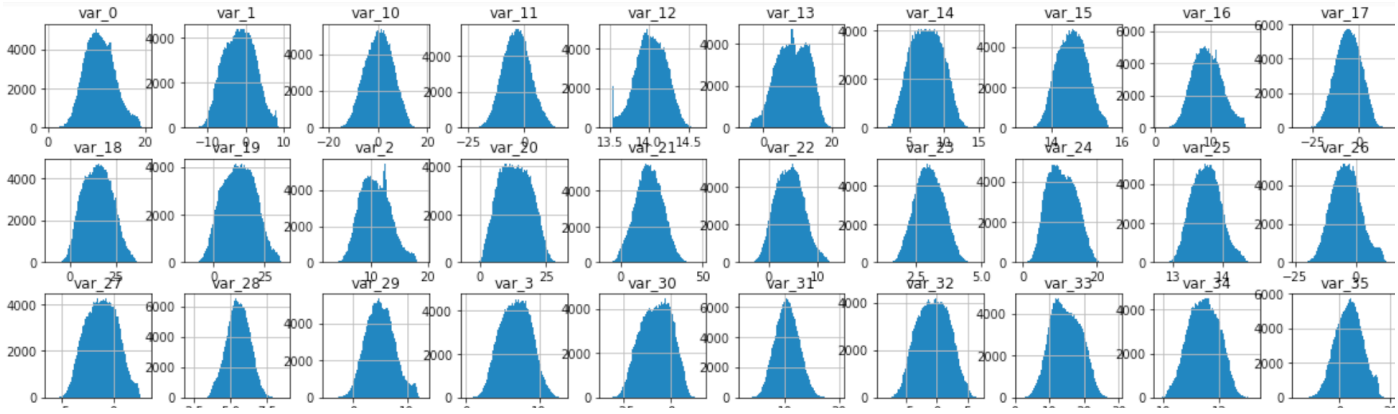
```
x=bd_train_new
y=bd_train['target']
```

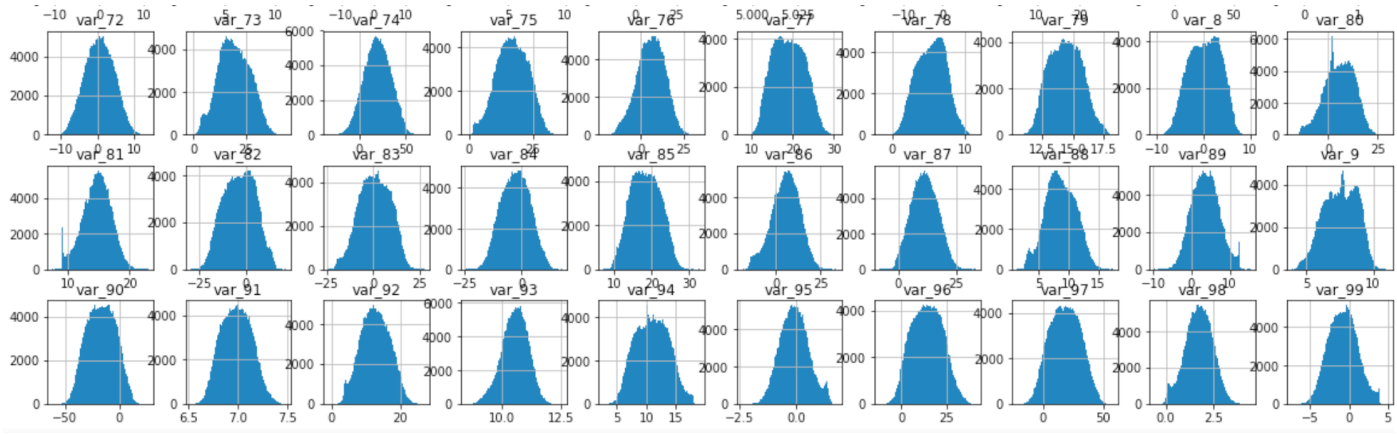
Next we are dropping 'target' and 'ID_code' variable from the training data and saved it in bd_train_new and later in x. We are saving our target variable as 'target' in y.

```
x_100=x.iloc[:,:100]
x_200=x.iloc[:,100:200]
```

```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
%matplotlib inline
x_100.hist(figsize=(20,20),bins=100)
plt.show()
```





To give an example , we have plotted for 2 different dataset. The first set consist of 1 to 100 variables and the new dataset consist from 101 to 200 variables.

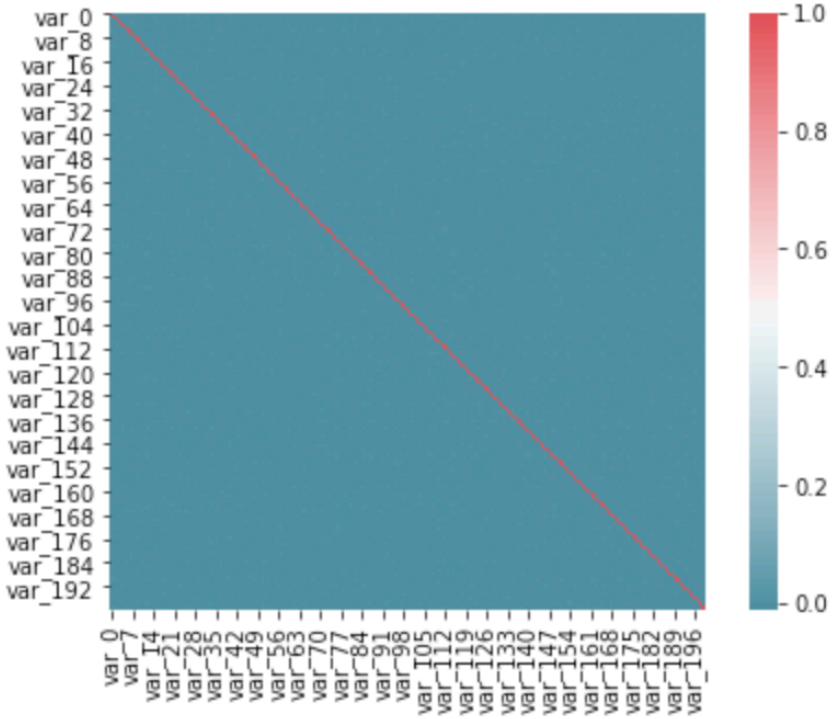
If we see from the above sample figures of the distribution of the datasets, we can see that some variable are normally distributed like var_81, var_72, var_10 and some variables are not normally distributed like var_13, var_80 etc.

The full distribution graphs of all the variables will be shown in the appendix.

2.1.2 Feature Selection :-

We have drawn a heatmap of correlation of all the training variables . We can see from the below correlation plot that there are no such correlation between the independent variables, i.e no multicollinearity.

```
<matplotlib.axes._subplots.AxesSubplot at 0x11badf358>
```

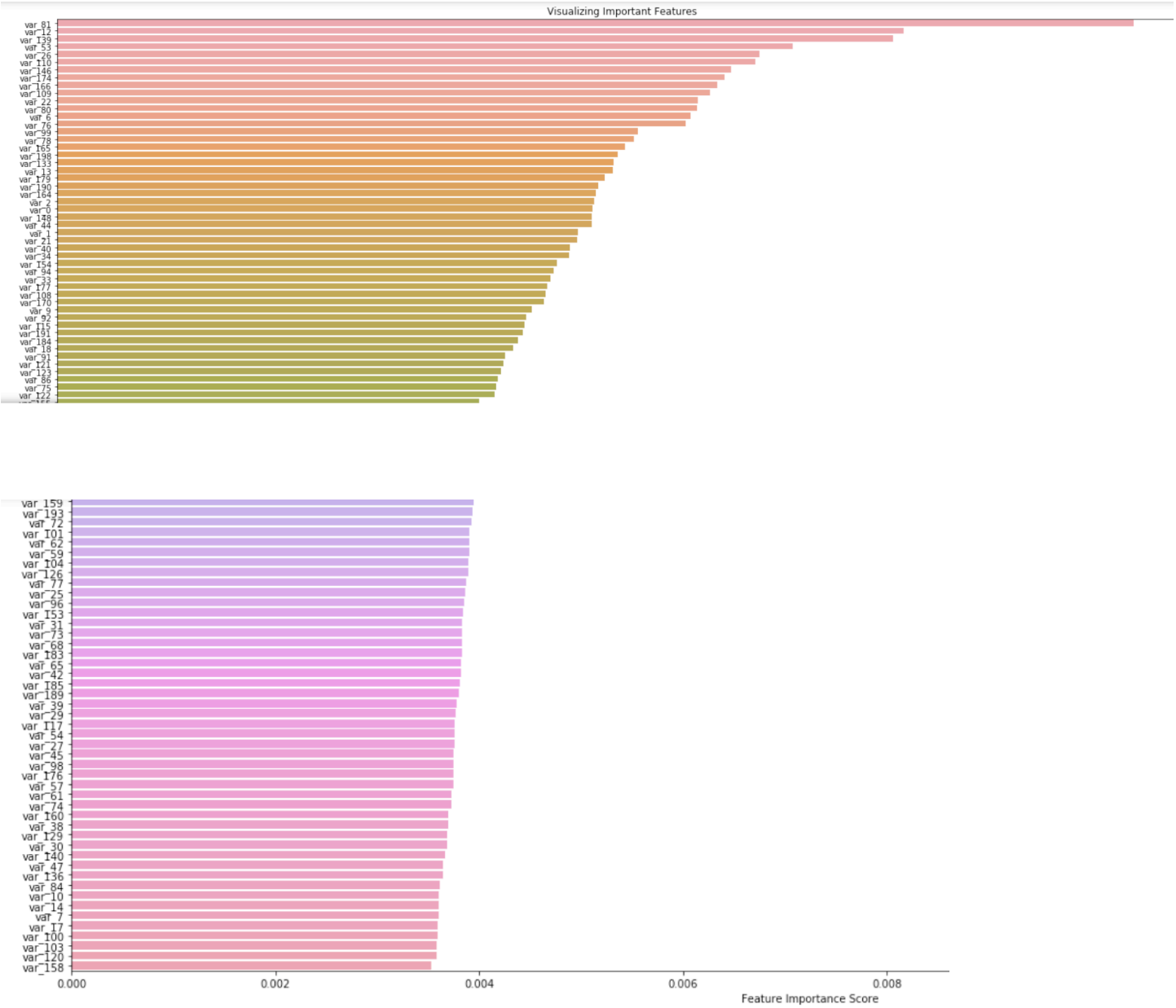


var_81	0.013805
var_12	0.010853
var_139	0.010718
var_53	0.009430
var_26	0.009005
var_110	0.008951
var_146	0.008638
var_174	0.008555
var_166	0.008465
var_109	0.008371
var_22	0.008210
var_80	0.008208
var_6	0.008126
var_76	0.008059
var_99	0.007449
var_78	0.007391
var_165	0.007277
var_198	0.007183
var_133	0.007129

var_57	0.003743
var_61	0.003724
var_74	0.003723
var_160	0.003692
var_38	0.003691
var_129	0.003684
var_30	0.003684
var_140	0.003658
var_47	0.003646
var_136	0.003639
var_84	0.003610
var_10	0.003605
var_14	0.003603
var_7	0.003597
var_17	0.003590
var_100	0.003585
var_103	0.003584
var_120	0.003582
var_158	0.003531

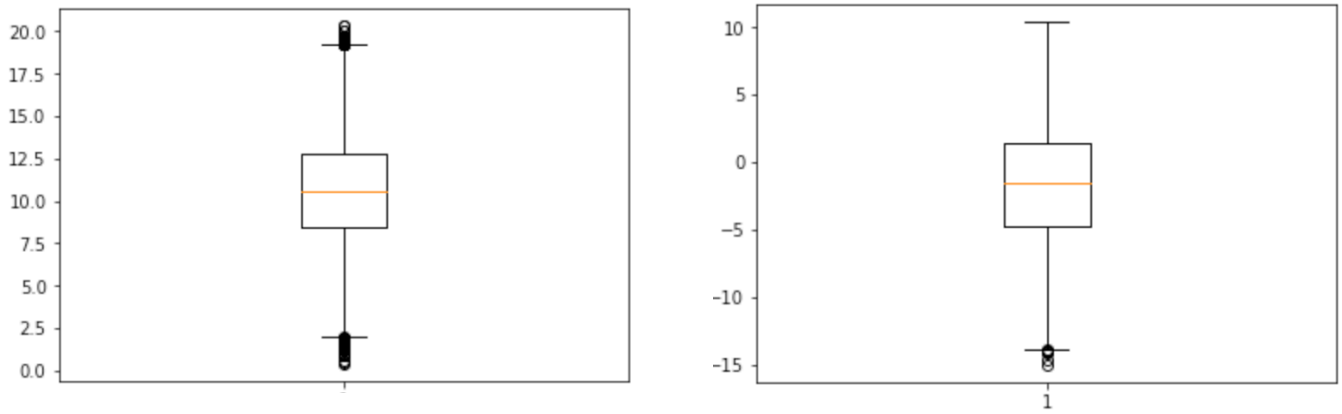
We have run RandomForestClassifier with n_estimator taking as 100. Finding the feature importance form the classifier we can see that var_81 is the most important variable followed by var_12, var_139. In the above figure left hand side of the table is the top 19 variables amongst the 200 variables. The right hand side of the box reflects the bottom 19 variables as the importance.

We can see the from the below plot of the feature importance that we can't drop any variables based on the important features. Every variable is important to some extent.



2.1.3 Outlier Analysis :-

We have seen the from the distribution of the variable that some the variables are skewed.The skew in these distributions can be most likely explained by the presence of outliers and extreme values in the data



The above 2 are graphical representation box plot analysis of var_0 and var_1.

So we have detected all the outliers from the all the variables and replaced with NA values. Later on we replaced the NA values with the mean value of the column.

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
mean	10.679914	-1.627622	10.715192	6.796529	11.078333	-5.065317	5.408949	16.545850	0.284162	7.567236
std	3.040051	4.050044	2.640894	2.043319	1.623150	7.863267	0.866607	3.418076	3.332634	1.235070
min	0.408400	-15.043400	2.117100	-0.040200	5.074800	-32.562600	2.347300	5.349700	-10.505500	3.970500
25%	8.453850	-4.740025	8.722475	5.254075	9.883175	-11.200350	4.767700	13.943800	-2.317800	6.618800
50%	10.524750	-1.608050	10.580000	6.825000	11.108250	-4.833150	5.385100	16.456800	0.393700	7.629600
75%	12.758200	1.358625	12.516700	8.324100	12.261125	0.924800	6.003000	19.102900	2.937900	8.584425
max	20.315000	10.376800	19.353000	13.188300	16.671400	17.251600	8.447700	27.691800	10.151300	11.150600

Table-1

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
mean	10.680009	-1.636018	10.711579	6.795464	11.078619	-5.070598	5.407901	16.545879	0.289666	7.567448
std	2.836903	3.789421	2.466563	1.912184	1.516982	7.366005	0.809304	3.200692	3.120572	1.156688
min	2.004400	-13.867400	3.035500	0.673200	6.318300	-29.188300	2.917000	6.465300	-9.991100	3.970500
25%	8.763800	-4.268400	8.987000	5.483800	10.080400	-10.251200	4.857800	14.303650	-1.921100	6.764800
50%	10.680009	-1.636018	10.711579	6.795464	11.078619	-5.070598	5.407901	16.545879	0.289666	7.567448
75%	12.434650	0.908325	12.278800	8.106100	12.086225	0.125850	5.913500	18.711100	2.596225	8.452325
max	19.213500	10.376800	18.206700	12.924900	15.807100	17.251600	7.854100	26.841000	10.151300	11.150600

Table-2

Table-1 describes the description of the variables before outlier detection while the Table-2 describes the description of the variables after the outlier detection and imputing with the mean value of the column after putting NA value in the outlier.

Lets take an example of var_0. From the box plot visualisation of var_0 we can clearly see that outliers are residing nearby the whiskers. So exactly the same thing can be seen in the table also. The IQR (Inter Quatrtile Range) for var_0 is 2.004 to 19.2135. The maximum and minimum data point in var_0 is 0.408 and 20.315. so we can see that the outliers are just around whiskers only.

If we see from the above chart we can clearly see that the outliers present in the data are not to the extreme.

2.1.4. Feature Scaling :-

As we have seen in the data distribution section that all the variables are not normally distributed. So we are using normalisation , just scale down the range of the data from 0 to 1.

	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_
0	0.402177	0.292074	0.584832	0.360750	0.541944	0.428616	0.445950	0.596853	0.251762	0.247420	0.582088	0.694649	0.484443	0.204591	0.581753	0.5052
1	0.504129	0.504508	0.505964	0.499707	0.501678	0.519331	0.504527	0.494735	0.510404	0.500961	0.502396	0.502651	0.493841	0.497481	0.487322	0.5069
2	0.383803	0.458737	0.596195	0.589273	0.449393	0.432917	0.815398	0.399996	0.251797	0.276041	0.479743	0.262776	0.650878	0.452441	0.486520	0.5375
3	0.526233	0.483233	0.389996	0.532375	0.660389	0.588981	0.592615	0.415186	0.205050	0.595326	0.562593	0.683532	0.328968	0.623783	0.406755	0.6160
4	0.455137	0.510803	0.648538	0.486814	0.627993	0.681244	0.612404	0.627517	0.807079	0.516413	0.192526	0.236556	0.330640	0.473670	0.506321	0.7061
5	0.550401	0.476370	0.630965	0.649151	0.489398	0.705195	0.327156	0.429934	0.674031	0.279731	0.515197	0.581014	0.135507	0.229767	0.536546	0.6788
6	0.569739	0.568557	0.416177	0.295339	0.507672	0.455826	0.664175	0.275416	0.278666	0.550508	0.328468	0.374510	0.623015	0.672905	0.510192	0.4090
7	0.671366	0.242503	0.714650	0.565252	0.246185	0.646412	0.561463	0.778422	0.747468	0.449381	0.535735	0.399137	0.742825	0.586981	0.441284	0.3717
8	0.819491	0.672738	0.718150	0.404801	0.261687	0.761227	0.310790	0.182575	0.337740	0.711996	0.519892	0.653279	0.532460	0.071001	0.758492	0.6319
9	0.610398	0.653422	0.386291	0.389954	0.767853	0.277830	0.637358	0.509219	0.502413	0.556775	0.517644	0.690811	0.438191	0.213147	0.665822	0.7931

So the above table shows the data after normalisation.

2.2 MODELING :-

2.2.1 Model Selection :-

We will be building different types of classification models, as the project is based on classification.

```
Count of target classes :
 0    179902
1     20098
Name: target, dtype: int64
percentage of count of target classes :
 0    89.951
1    10.049
Name: target, dtype: float64
```

Fist of all if we see the target variable we can see that the target variable is imbalanced.

We will be using different classification models like logistic regression, random forest, XGboost and stacking. Also other than just doing stratified samples we will also be doing SMOTE sampling for building model. We will be using different metrics like AUC, Recall, precision and F1 Score to choose the best model.

2.2.2 Logistic Regression :-

```
x1_train,x1_test,y1_train,y1_test=train_test_split(x_new,y,test_size=0.25,random_state=2)
```

```
print (x1_train.shape,y1_train.shape)
print (x1_test.shape,y1_test.shape)
```

```
(150000, 200) (150000,)
(50000, 200) (50000,)
```

First of all we are splitting the data into train and test data with 75% as train data and the rest 25% as test data, without stratified sampling.

We build a logistic regression based on these data. After fitting the training data we have predicted on testing data.

roc_auc_score came as :-

0.7593689713025209

```
Accuracy: 0.76782
Precision: 0.2714114241333051
Recall: 0.748733930658356
```

	precision	recall	f1-score	support
0	0.96	0.77	0.86	44866
1	0.27	0.75	0.40	5134
accuracy			0.77	50000
macro avg	0.62	0.76	0.63	50000
weighted avg	0.89	0.77	0.81	50000

From the above result we can see that auc is 0.75 where as precision is 0.27 which is on the lower side. F1-score (which is harming mean of precision and recall) is 0.4

Next we will try with different model/technique to improve the performance of the model.

2.2.3 Logistic Regression with stratified sampling :-

```
x_train,x_test,y_train,y_test=train_test_split(x_new,y,test_size=0.25,random_state=2,stratify=y)
```

Here we have split the data with sampling technique as stratified.

On top of it build a logistic regression model. After fitting to train dat and predicting on test data the below result has been found

roc_auc_score :- 0.7607464603910479

```
Accuracy: 0.76822
Precision: 0.26744597945448106
Recall: 0.7513933121019108
```

	precision	recall	f1-score	support
0	0.97	0.77	0.86	44976
1	0.27	0.75	0.39	5024
accuracy			0.77	50000
macro avg	0.62	0.76	0.63	50000
weighted avg	0.90	0.77	0.81	50000

We can see that we the model is able to improve the roc_auc_score by 0.1 So we will go to the next model/technique to see whether we can further improve the model.

2.2.4 Logistic Regression with SMOTE :-

```
Before OverSampling, counts of label '1': 20098
Before OverSampling, counts of label '0': 179902

After OverSampling, the shape of train_X: (269852, 200)
After OverSampling, the shape of train_y: (269852,)

After OverSampling, counts of label '1': 134926
After OverSampling, counts of label '0': 134926
```

As earlier mentioned that the target class variable is imbalanced , so we are using SMOTE oversampling technique to balanced the target class.

As we can see in the above pic, that after using SMOTE we are able to balanced the target class. Next we have build a logistic regression on SMOTE set and found the below mentioned result.

Roc_auc_score :- 0.7483657950907151

```
Accuracy: 0.69712
Precision: 0.22325530518486109
Recall: 0.8125
```

	precision	recall	f1-score	support
0	0.97	0.68	0.80	44976
1	0.22	0.81	0.35	5024
accuracy			0.70	50000
macro avg	0.60	0.75	0.58	50000
weighted avg	0.90	0.70	0.76	50000

Here we can see that the roc_auc_score along with f1-score has come down.

We will be not accepting this model. We will go for next model.

2.2.5 RandomForest :-

We have used a cross validation hyper parameter tuning n random forest. Its a Bagging Algorithm.

The technique of cross validation (CV) is best explained by example using the most common method, K-Fold CV. When we approach a machine learning problem, we make sure to split our data into a training and a testing set. In K-Fold CV, we further split our training set into K number of subsets, called folds. We then iteratively fit the model K times, each time training the data on K-1 of the folds and evaluating on the Kth fold (called the validation data).
For hyperparameter tuning, we perform many iterations of the entire K-Fold CV process, each time using different model settings. We then compare all of the models, select the best one, train it on the full training set, and then evaluate on the testing set.
Using Scikit-Learn's RandomizeeSearchCV method, we can define a grid of hyperparameter ranges, and randomly sample from the grid, performing Fold CV with each combination of values.
We will try adjusting the following set of hyperparameters:

- n_estimators = number of trees in the foreset
- max_features = max number of features considered for splitting a node
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- criterion = rules fro sampling (Gini or Entropy)

To use RandomizedSearchCV, we first need to create a parameter grid to sample from during fitting:

```
param_dist = {"n_estimators":[10,50,100,200],
              "max_depth": [3,5,10, None],
              "max_features": [5,10,20],
              "min_samples_split": [2,5,10],
              "min_samples_leaf": [1,5,10],
              "criterion": ["gini", "entropy"]}
```

On each iteration, the algorithm will choose a difference combination of the features. Altogether, there are 4*4*3*3*3*2= 864
However, the benefit of a random search is that we are not trying every combination, but selecting at random to sample a wide range of values.

The most important arguments in RandomizedSearchCV are n_iter, which controls the number of different combinations to try, and cv which is the number of folds to use for cross validation (we use 20 and 3 respectively).
After running RandomizedSearchCV we found the best combination of parameter for randomforestclassifier.
roc_auc_score : 0.5052338246357009
Which is very low compare to other model.
We are not accepting this model and will build a different model for further testing.

2.2.6 XGboost :

XGBoost (eXtreme Gradient Boosting) is an advanced implementation of gradient boosting algorithm.
First of all we tried building xgboost model without any hyperparameter tuning.
Roc_auc_score :- 0.74.
So now we are trying building this model with hyperparameter tuning.
Hyperparameter tuning:

1. max_depth [default=6] : Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample. Typical value 3-10.
2. Subsample :Denotes the fraction of observations to be randomly samples for each tree.
   Lower values make the algorithm more conservative and prevents overfitting but too small
   values might lead to under-fitting.typical value 0.5 to 1
3. Colsample_bytree : Denotes the fraction of columns to be randomly samples for each tree.typical value 0.5 -1.
4. Learning_rate : This determines the impact of each tree on the final outcome. Lower values are generally preferred as they make the model robust to the specific characteristics of tree and thus allowing it to generalize well.
5. n_estimators : The number of sequential trees to be modeled

On top of that we are using n_iter =20 and cv =10 (stratifiedKFold)

```
# hyperparameter tuning set for param_dist
param_dist1 = {"learning_rate":[0.05,0.1,0.2],
               "n_estimators":[100,150,250],
               "max_depth": [3,4,5],
               "subsample":[0.8,1,0.5],
               "colsample_bytree":[0.6,0.7,0.8]
              }
n_iter_search=20
cv_strat=StratifiedKFold(n_splits=10,shuffle=True,random_state=2)
```

After choosing the best combination of parameters we are using scale_pos_weight=9 in xgbclassifier. After training and fitting the model the result are as follows.

roc_auc_score : 0.7821823089795095

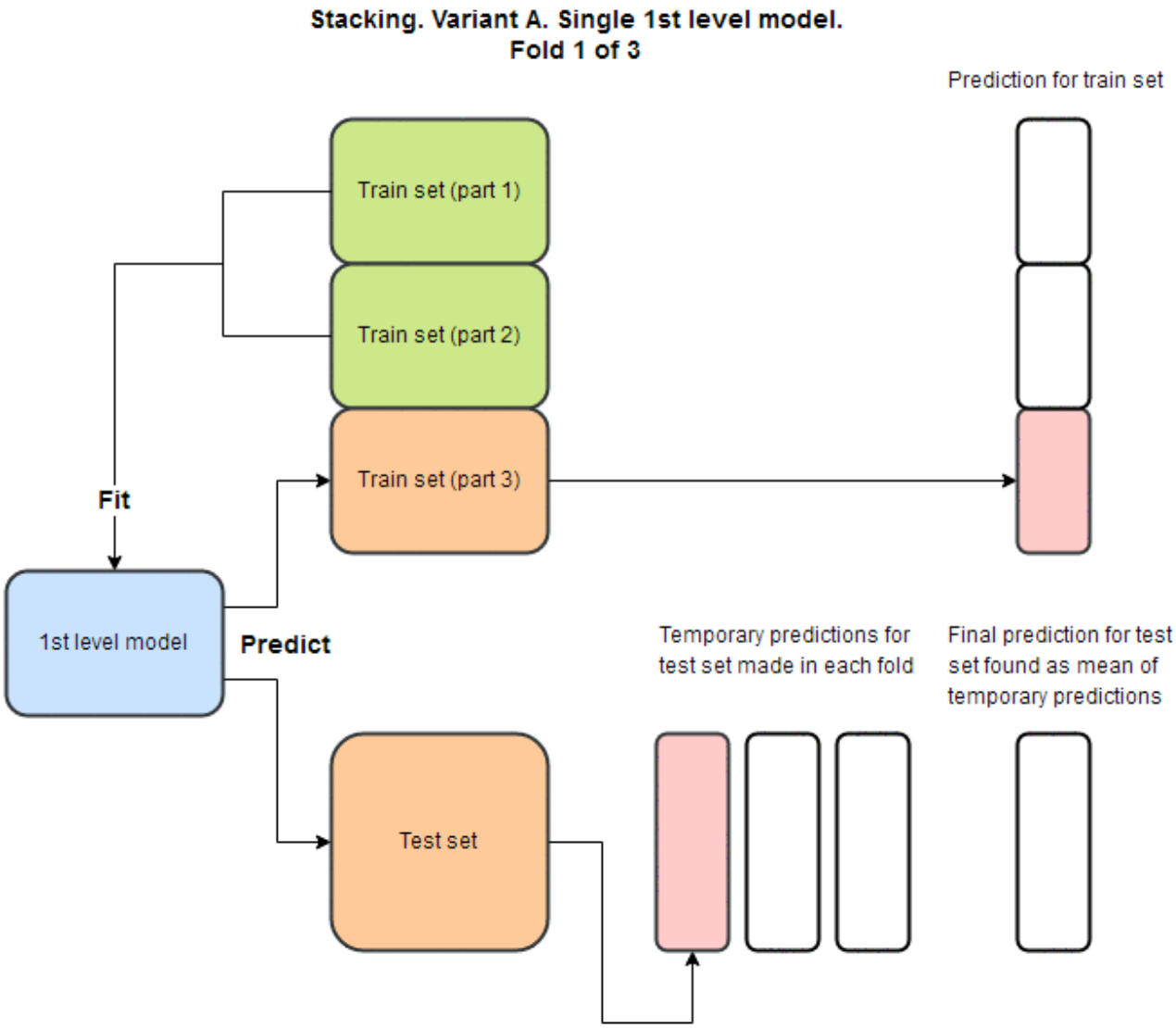
```
Accuracy: 0.8141
Precision: 0.3179299172990025
Recall: 0.7422372611464968
```

	precision	recall	f1-score	support
0	0.97	0.82	0.89	44976
1	0.32	0.74	0.45	5024
accuracy			0.81	50000
macro avg	0.64	0.78	0.67	50000
weighted avg	0.90	0.81	0.84	50000

Here we can see that the improvement in result by xgboost.
roc_auc_score along with f1 score and precision has increased from all the previous models.
Next will build another model to see whether we can improve the performance further or not.

2.2.7 STACKING :

The main idea behind the structure of a stacked generalisation is to use one or more first level models, make predictions using these models and then use these predictions as features to fit one or more second level models on top. To avoid overfitting, cross-validation is usually used to predict the OOF (out-of-fold) part of the training set. There are two different variants available in this package but I'm going to describe 'Variant A' in this paragraph. To get the final predictions in this variant, we take the mean or mode of all of our predictions. The whole process can be visualized using this GIF from vecstacks' documentation:



It is time now to define a few first level models for our stacked generalization. We are going to use three models: Logistic Regression, a Random Forest Classifier and an XGBoost Classifier.

```
models = [
    LogisticRegression(class_weight='balanced'),

    RandomForestClassifier(n_estimators=200,min_samples_split=10,min_samples_leaf=10,max_features='sqrt',max_depth=None
                           criterion='gini',bootstrap=True,random_state=2,n_jobs=-1,class_weight="balanced"),

    XGBClassifier(learning_rate=0.2,n_estimators=250,min_samples_split=5,min_samples_leaf=1,max_features='sqrt',
                                  max_depth=3,colsample_bytree=0.6,subsample=0.5,objective='binary:logistic',
                  scale_pos_weight=9)
]
```

The above base models are taken from the earlier models . The hyperparameter tuning is also been done earlier.
Now developing the training and testing data of the stacked model.

```
_train, s_test = stacking(models,x_train,y_train,x_test,regression=False,mode='oof_pred_bag',needs_proba=False,
        save_dir=None,metric=roc_auc_score,n_folds=4,stratified=True,random_state=2,verbose=2)
```

The stacking function takes several inputs:

- models: the first level models we defined earlier
- x_train, y_train, x_test: our data
- regression: Boolean indicating whether we want to use the function for regression. In our case set to False since this is a classification
- mode: using the earlier describe out-of-fold during cross-validation
- needs_proba: Boolean indicating whether you need the probabilities of class labels
- save_dir: save the result to directory Boolean
- metric: what evaluation metric to use (we imported the roc_auc_score)
- n_folds: how many folds to use for cross-validation
- stratified: whether to use stratified cross-validation
- random_state: setting a random state for reproducibility
- verbose: 2 here refers to printing all info

After having the data for train and test for from stacking we are using Logistic Regression ,model as our 2nd level for fitting the second level model(s) of our choice on our predictions to make our final predictions.

We have made the model based on s_train data and fitted on y_train and then tested on s_test data.

The roc_auc_score : 0.784952859501005

So based on roc_auc_score we can say that this stacking model has definitely increased our model performance.

```
Accuracy: 0.84644
Precision: 0.36414823914823913
Recall: 0.7080015923566879
```

	precision	recall	f1-score	support
0	0.96	0.86	0.91	44976
1	0.36	0.71	0.48	5024
accuracy			0.85	50000
macro avg	0.66	0.78	0.70	50000
weighted avg	0.90	0.85	0.87	50000

Also we are able to increase the f1 score and precision rate also from xgbclassifier.

So now we have prepared another stacking model based on original data without outlier analysis and normalisation. We have checked this model solely because of the structure of the data. If we see that the outliers , actually are so much far from the whiskers , and no variables are of completely different scale.

So we build a stack model based on the original data.

```
s4_train, s4_test = stacking(models,bd_train1,y,test_data,regression=False,mode='oof_pred_bag',needs_proba=False,
        save_dir=None,metric=roc_auc_score,n_folds=4,stratified=True,random_state=2,verbose=2)
```

```
raw_model=LogisticRegression(class_weight='balanced')
```

```
raw_model.fit(s4_train,y)
```

```
raw_predict=raw_model.predict(s4_test)
```

```
submission_new=pd.DataFrame(list(zip(bd_test['ID_code'],list(raw_predict))),
                      columns=['ID_code','target'])
```

The roc_auc_score :- 0.7998538003000044

```
Accuracy: 0.86116
Precision: 0.3955792682926829
Recall: 0.7231289808917197
```

	precision	recall	f1-score	support
0	0.97	0.88	0.92	44976
1	0.40	0.72	0.51	5024
accuracy			0.86	50000
macro avg	0.68	0.80	0.72	50000
weighted avg	0.91	0.86	0.88	50000

If we compare the roc_auc_score and f1 score along with precision and recall we can see that this current stack model is giving better results in every aspect.
F1 score also increased from 0.48 to 0.51.
Precision increased from 0.36 to 0.4.

So we are selecting the above stack model which is based on original data without any preprocessing.

So now we are reciting the on original test data.

We are creating 2 different data of train and test from the entire train data and test data. Once the model has been built we are creating a dataframe

```
submission_new.target.value_counts()
```

consist of ‘ID_code’ and ‘target’.

```
0     163858
1      36142
```

This is the original output what the model has predicted . Based on the test data, the model has predicted that 36142 customer will make a specific transaction in future.

MODEL BUILDING IN R :-

We have also built algorithm based on the same problem in R. We have imported the training and testing file into R.

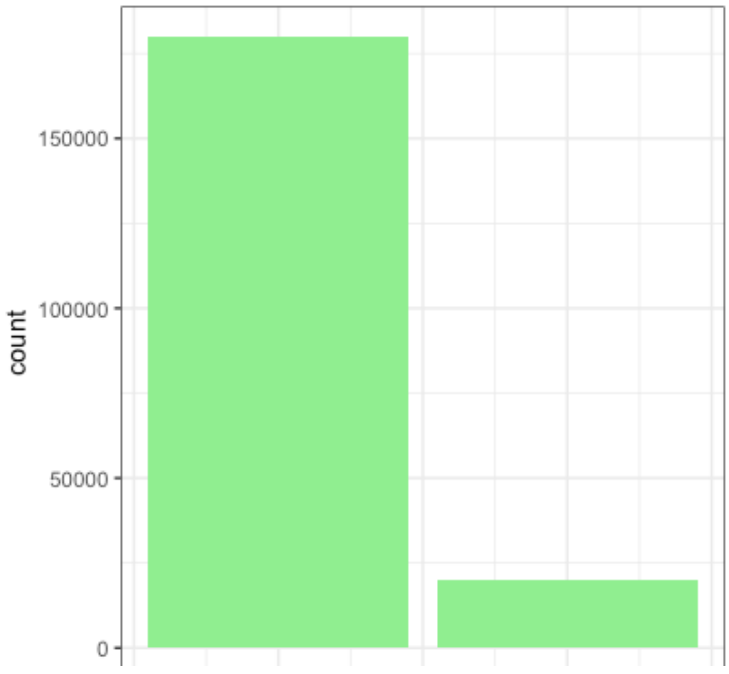
```
## Dropping 'target' and ' ID_code' from train and 'ID_code' from test data
bd_train_new= subset(bd_train, select = -c(target,ID_code))
bd_test_new=subset(bd_test,select= -c(ID_code))
```

```
table(bd_train$target)

   0        1
179902  20098
```

We can clearly see that the target class is imbalanced. Below is the graphical presentation of the table of target class variable.

```
library(ggplot2)
ggplot(bd_train,aes(target))+theme_bw()+geom_bar(stat='count',fill='lightgreen')
```

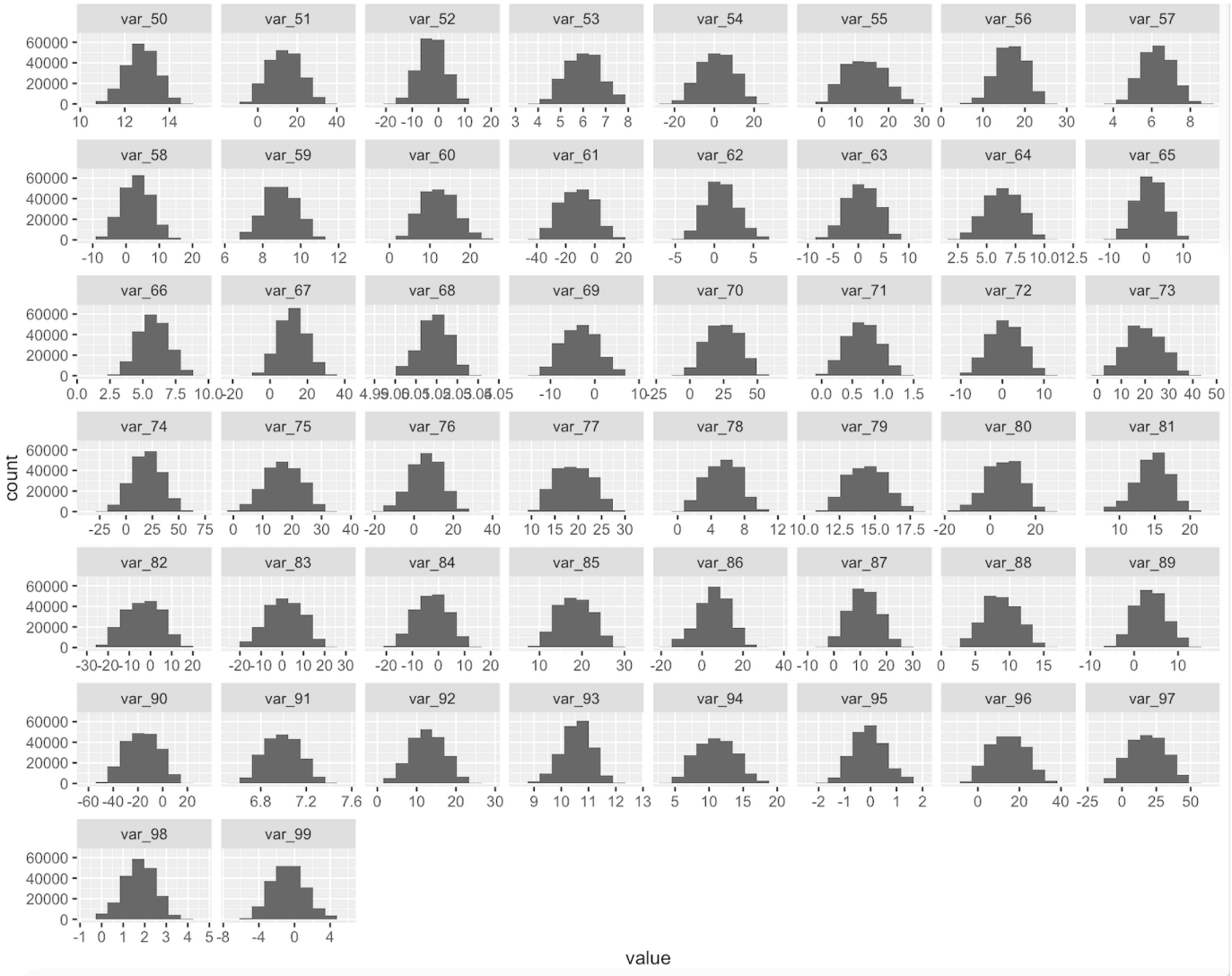


# # Let us look into distribution of the Training Data and Test Data
library(tidyverse)
hist_data=bd_train_new %>% gather() %>% head()

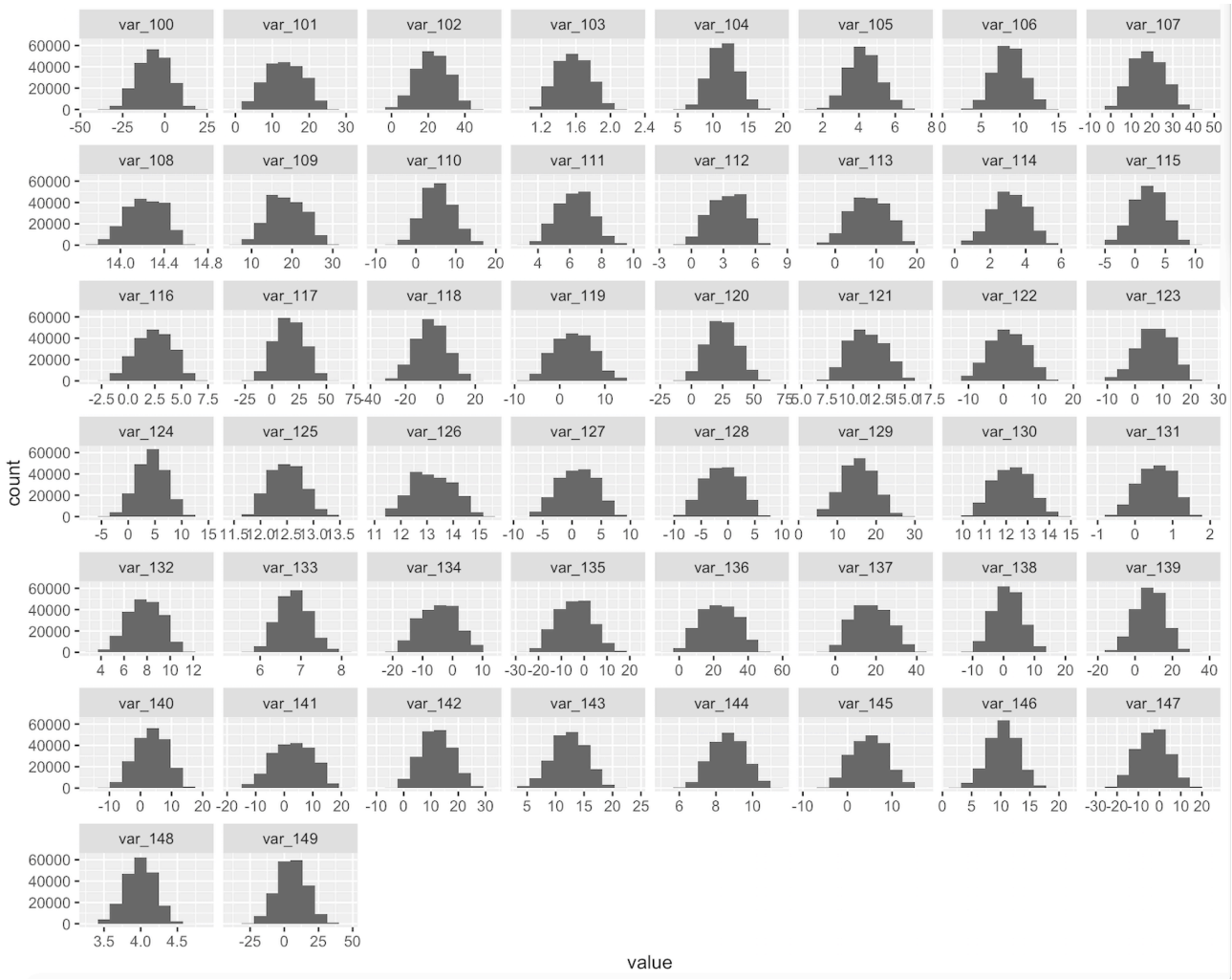
We have imported the library tidyverse. We have used 'gather' command to make key value pair .

ggplot(gather(bd_train_new[,c(1:50)]), aes(value)) +
geom_histogram(bins = 10) +
facet_wrap(~key, scales = 'free_x')

```
## ggplot(gather(bd_train_new[,c(51:100)]), aes(value)) +
##  geom_histogram(bins = 10) +
##  facet_wrap(~key, scales = 'free_x')
```



```
## ggplot(gather(bd_train_new[,c(101:150)]), aes(value)) +
##  geom_histogram(bins = 10) +
## facet_wrap(~key, scales = 'free_x')
```



```
## ggplot(gather(bd_train_new[,c(151:200)]), aes(value)) +
##  geom_histogram(bins = 10) +
## facet_wrap(~key, scales = 'free_x')
```

MISSSING VALUE ANAYSIS :-

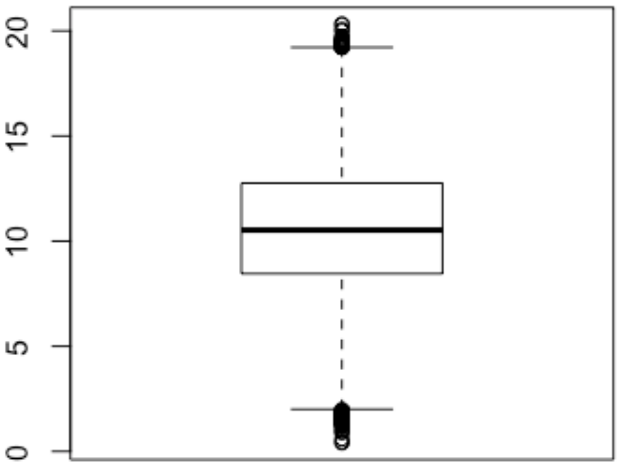
```
## Misising Value Analysis fro train and test data
missing_val= data.frame(apply(bd_train_new, 2,function(x){sum(is.na(x))}))
missing_test=data.frame(apply(bd_test_new,2,function(x){sum(is.na(x))}))
```

We have checked the missing in train and test data. But there was no missing value present in the data.

OUTLIER ANALYSIS:-

```
boxplot(bd_train_new$var_0)
boxplot(bd_train_new$var_3)
```

So we have done the box plot analysis in var_0.




```
b=boxplot(bd_train_new$var_0)
b

$stats
        [,1]
[1,]  2.00440
[2,]  8.45380
[3,] 10.52475
[4,] 12.75820
[5,] 19.21350

$n
[1] 2e+05

$conf
        [,1]
[1,] 10.50954
[2,] 10.53996
```

```
$out
  [1] 19.2893  1.3350  1.6044  1.9220  0.5979  1.7243  1.8611  1.9488 19.3259
 [10] 19.2875 19.2857 19.3166  1.6088  1.8504  1.8386 19.2756 19.4583 19.2787
 [19] 19.3793  1.4449 19.4528  1.3215  1.1563  1.4920 19.2175 19.2539 19.4477
 [28] 19.2767 19.2203 19.3872  0.8935  1.3350  1.9484  0.9302 19.3049  1.0147
 [37] 19.2210  1.6659 20.3150  1.5724 19.6021 19.2306 19.4997 19.3315 19.3184
 [46] 19.2247  1.5858 19.3565 19.3479 19.3296 19.2292  1.7568  1.6938 19.3315
 [55] 20.3150 19.3042 19.3992  1.6044 19.2815 19.5251 19.2445 19.5599  1.6824
 [64]  1.9484 19.7379 19.4725 19.2717 19.3283  1.2832 19.4640  1.1862 19.2237
 [73] 19.3294  0.4528 19.3259  1.2968  1.8093  1.9822  1.5822 19.2429  1.9365
 [82] 19.6265 19.2619 20.0740 19.4313 19.3536  1.2944  1.4059 19.3582 19.2682
 [91] 19.2157  0.4084 19.4310  1.4566  1.7385  1.5307 19.9984  1.6005 19.2514
[100] 19.7011  1.9094  1.8753 19.3166  1.7892

$group
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [40] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [79] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

$names
[1] "1"
```

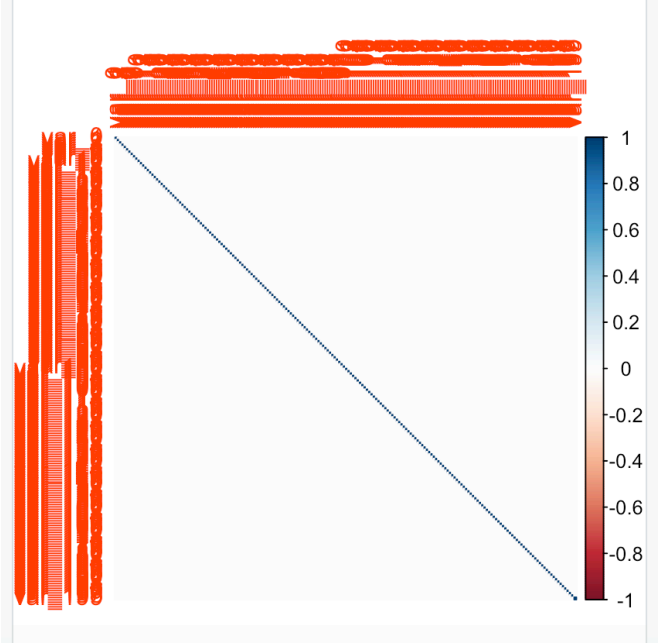
From $out we can see the observation which are falling into outlier category. Here we Can see that the outliers is just hovering around whiskers only.

```
## cnames= colnames(bd_train_new %>% select(1:200))
## for(i in cnames){
  val = bd_train_new[,i][bd_train_new[,i] %in% boxplot.stats(bd_train_new[,i])$out]
  print(length(val))
  bd_train_new[,i][bd_train_new[,i] %in% val] = NA
}

## for (i in cnames){
  bd_train_new[,i][is.na(bd_train_new[,i])]=
    mean(bd_train_new[,i], na.rm = T)
}
```

So in the above code what we have done is that, we have identified the outliers and replaced with NA values and later we replaced all the NA values with mean values of each column.

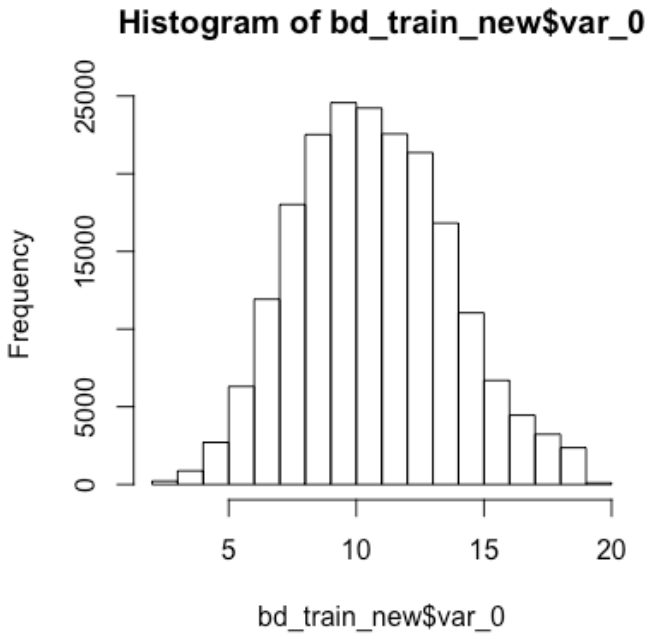
```
## library(corrplot)
## cor_plot=cor(bd_train_new)
## head(round(cor_plot,2))
## corrplot(cor_plot,method = "color")
```



So from above correlation plot we can see that there are no correlation between the Variables.

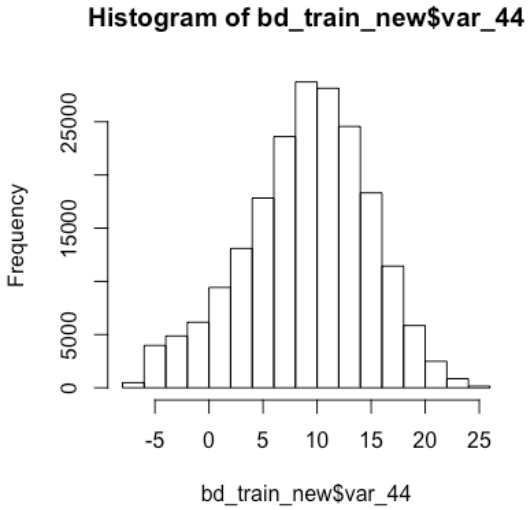
NORMALISATION :-

hist(bd_train_new$var_0)



hist(bd_train_new$var_33)

hist(bd_train_new$var_44)



From the above histogram chart we can see that the variables are not normally distributed.

for (i in cnames) {

```
  print(i)
  bd_train_new[,i]=(bd_train_new[,i]-min(bd_train_new[,i]))/
    (max(bd_train_new[,i]-min(bd_train_new[,i])))
}
```

We have normalised the data.

MODEL DEVELOPMENT :-

We have added the target variable in the training data gain.

```
## bd_train_new$target=bd_train$target

## stratified sampling
set.seed(1234)
train.index=createDataPartition(bd_train_new$target, p=.8, list= FALSE)
train1=bd_train_new[train.index,]
test1=bd_train_new[-train.index,]
```

We have sampled the data and with the help of createDatapartition we have divided the data into train a test set.

```
# model.logit=glm(target~.,data = train1,family = 'binomial')
```

Then we will build a logistic regression model.

```
# summary(model.logit)

Call:
glm(formula = target ~ ., family = "binomial", data = train1)

Deviance Residuals:
   Min      1Q  Median      3Q     Max
-2.6350  -0.4029  -0.2349  -0.1255   3.7704

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.678376   0.394121  -9.333  < 2e-16 ***
var_0        0.942163   0.053855  17.494  < 2e-16 ***
var_1        0.956149   0.057524  16.622  < 2e-16 ***
var_2        1.029290   0.054495  18.888  < 2e-16 ***
var_3        0.162592   0.057962   2.805 0.005029 **
var_4        0.183851   0.056326   3.264 0.001098 **
var_5        0.614725   0.056836  10.816  < 2e-16 ***
var_6        1.278950   0.054505  23.465  < 2e-16 ***
var_7        0.015395   0.057522   0.268 0.788985
var_8        0.351795   0.058312   6.033 1.61e-09 ***
var_9       -0.806497   0.055699 -14.480  < 2e-16 ***
var_10      -0.020634   0.056240  -0.367 0.713702
var_11       0.457628   0.054315   8.425  < 2e-16 ***
var_12      -1.227174   0.053798 -22.811  < 2e-16 ***
```

var_13	-1.052883	0.056233	-18.723	< 2e-16	***
var_14	-0.056300	0.057527	-0.979	0.327746	
var_15	0.286940	0.057141	5.022	5.12e-07	***
var_16	0.137565	0.054318	2.533	0.011323	*
var_17	-0.026828	0.055931	-0.480	0.631466	
var_18	0.819042	0.057694	14.196	< 2e-16	***
var_19	0.215049	0.056308	3.819	0.000134	***
var_20	-0.399158	0.056871	-7.019	2.24e-12	***
var_21	-1.128331	0.054577	-20.674	< 2e-16	***
var_22	1.123528	0.055328	20.307	< 2e-16	***
var_23	-0.524070	0.056640	-9.253	< 2e-16	***
var_24	0.577917	0.058377	9.900	< 2e-16	***
var_25	0.237611	0.055168	4.307	1.65e-05	***
var_26	1.111751	0.052929	21.005	< 2e-16	***
var_27	-0.066813	0.056856	-1.175	0.239949	
var_28	-0.466758	0.054438	-8.574	< 2e-16	***
var_29	0.136844	0.053661	2.550	0.010768	*
var_30	-0.058503	0.057990	-1.009	0.313051	
var_31	-0.506323	0.054871	-9.228	< 2e-16	***
var_32	0.598176	0.056474	10.592	< 2e-16	***
var_33	-0.920417	0.059440	-15.485	< 2e-16	***
var_34	-1.064425	0.057738	-18.435	< 2e-16	***
var_35	0.691081	0.054823	12.606	< 2e-16	***
var_36	-0.753417	0.055404	-13.599	< 2e-16	***
var_37	0.159479	0.054611	2.920	0.003497	**
var_38	0.007949	0.054836	0.145	0.884746	
var_39	-0.102629	0.055583	-1.846	0.064834	.
var_40	0.989465	0.054973	17.999	< 2e-16	***
var_41	-0.049083	0.056434	-0.870	0.384445	
var_42	-0.186549	0.057572	-3.240	0.001194	**
var_43	-0.511933	0.057230	-8.945	< 2e-16	***
var_44	-0.849919	0.050887	-16.702	< 2e-16	***
var_45	-0.421155	0.057768	-7.290	3.09e-13	***
var_46	0.074949	0.055149	1.359	0.174141	
var_47	0.188618	0.058463	3.226	0.001254	**
var_48	0.594849	0.056677	10.495	< 2e-16	***
var_49	0.538733	0.057832	9.315	< 2e-16	***
var_50	-0.307294	0.053879	-5.703	1.17e-08	***
var_51	0.424803	0.056723	7.489	6.94e-14	***
var_52	0.532655	0.054696	9.738	< 2e-16	***
var_53	1.217266	0.054123	22.491	< 2e-16	***
var_54	-0.347657	0.057303	-6.067	1.30e-09	***
var_55	0.357785	0.055619	6.433	1.25e-10	***
var_56	-0.676905	0.056349	-12.013	< 2e-16	***
var_57	-0.329100	0.056342	-5.841	5.18e-09	***
var_58	-0.452438	0.054445	-8.310	< 2e-16	***
var_59	-0.187437	0.055740	-3.363	0.000772	***
var_60	0.181948	0.056274	3.233	0.001224	**
var_61	0.167489	0.058697	2.853	0.004325	**
var_62	0.273287	0.053522	5.106	3.29e-07	***

var_63	-0.297814	0.057063	-5.219	1.80e-07	***
var_64	-0.235300	0.056822	-4.141	3.46e-05	***
var_65	0.161618	0.056085	2.882	0.003956	**
var_66	0.365064	0.055516	6.576	4.84e-11	***
var_67	0.799851	0.054438	14.693	< 2e-16	***
var_68	-0.240432	0.054118	-4.443	8.88e-06	***
var_69	0.123736	0.056222	2.201	0.027747	*
var_70	0.550980	0.058735	9.381	< 2e-16	***
var_71	0.616269	0.056560	10.896	< 2e-16	***
var_72	-0.263782	0.055984	-4.712	2.46e-06	***
var_73	-0.105443	0.056314	-1.872	0.061149	.
var_74	0.339754	0.055324	6.141	8.19e-10	***
var_75	-0.738302	0.056186	-13.140	< 2e-16	***
var_76	-1.175410	0.054219	-21.679	< 2e-16	***
var_77	-0.342743	0.059668	-5.744	9.24e-09	***
var_78	0.986351	0.058618	16.827	< 2e-16	***
var_79	0.096917	0.057275	1.692	0.090622	.
var_80	-1.056609	0.054552	-19.369	< 2e-16	***
var_81	-1.400920	0.051133	-27.398	< 2e-16	***
var_82	0.442537	0.058065	7.621	2.51e-14	***
var_83	-0.425162	0.056451	-7.531	5.02e-14	***
var_84	0.159038	0.056921	2.794	0.005206	**
var_85	-0.401146	0.056953	-7.043	1.88e-12	***
var_86	-0.691833	0.052878	-13.084	< 2e-16	***
var_87	-0.709990	0.056090	-12.658	< 2e-16	***
var_88	-0.316585	0.055838	-5.670	1.43e-08	***
var_89	0.741967	0.053924	13.760	< 2e-16	***
var_90	0.587104	0.058114	10.103	< 2e-16	***
var_91	0.775600	0.058244	13.316	< 2e-16	***
var_92	-0.896516	0.056478	-15.874	< 2e-16	***
var_93	-0.606328	0.053656	-11.300	< 2e-16	***
var_94	0.929650	0.055912	16.627	< 2e-16	***
var_95	0.698232	0.052624	13.268	< 2e-16	***
var_96	0.058944	0.058053	1.015	0.309933	
var_97	0.294669	0.057942	5.086	3.66e-07	***
var_98	-0.044043	0.053666	-0.821	0.411818	
var_99	1.080923	0.053707	20.126	< 2e-16	***
var_100	0.033936	0.056405	0.602	0.547409	
var_101	-0.182785	0.055243	-3.309	0.000937	***
var_102	-0.300137	0.056394	-5.322	1.03e-07	***
var_103	-0.015170	0.057355	-0.265	0.791393	
var_104	-0.523347	0.055484	-9.432	< 2e-16	***
var_105	0.437188	0.053786	8.128	4.35e-16	***
var_106	0.609189	0.054193	11.241	< 2e-16	***
var_107	-0.814141	0.056388	-14.438	< 2e-16	***
var_108	-0.833655	0.056521	-14.749	< 2e-16	***
var_109	-0.948998	0.057520	-16.498	< 2e-16	***
var_110	1.121930	0.053328	21.038	< 2e-16	***
var_111	0.483451	0.055583	8.698	< 2e-16	***
var_112	0.464196	0.059259	7.833	4.75e-15	***

var_113	-0.330818	0.057848	-5.719	1.07e-08	***
var_114	-0.467916	0.055080	-8.495	< 2e-16	***
var_115	-0.897632	0.054460	-16.482	< 2e-16	***
var_116	-0.508113	0.057882	-8.778	< 2e-16	***
var_117	0.013163	0.056318	0.234	0.815201	
var_118	0.739693	0.055278	13.381	< 2e-16	***
var_119	0.556107	0.057119	9.736	< 2e-16	***
var_120	-0.161527	0.055703	-2.900	0.003734	**
var_121	-0.832314	0.057946	-14.364	< 2e-16	***
var_122	-0.814784	0.056793	-14.347	< 2e-16	***
var_123	-0.743217	0.055075	-13.495	< 2e-16	***
var_124	0.102486	0.054876	1.868	0.061820	.
var_125	0.559694	0.057273	9.772	< 2e-16	***
var_126	0.018083	0.052878	0.342	0.732370	
var_127	-0.807639	0.057748	-13.986	< 2e-16	***
var_128	0.580229	0.058169	9.975	< 2e-16	***
var_129	-0.116941	0.055561	-2.105	0.035314	*
var_130	0.605443	0.057810	10.473	< 2e-16	***
var_131	-0.581995	0.057875	-10.056	< 2e-16	***
var_132	-0.474079	0.057724	-8.213	< 2e-16	***
var_133	0.945294	0.054283	17.414	< 2e-16	***
var_134	0.345933	0.056894	6.080	1.20e-09	***
var_135	0.485273	0.054312	8.935	< 2e-16	***
var_136	-0.094876	0.058757	-1.615	0.106371	
var_137	0.569568	0.058890	9.672	< 2e-16	***
var_138	0.376528	0.054779	6.874	6.26e-12	***
var_139	-1.307053	0.053996	-24.206	< 2e-16	***
var_140	0.367514	0.057016	6.446	1.15e-10	***
var_141	-0.592177	0.057441	-10.309	< 2e-16	***
var_142	-0.371748	0.055336	-6.718	1.84e-11	***
var_143	-0.278596	0.054656	-5.097	3.45e-07	***
var_144	0.390388	0.056138	6.954	3.55e-12	***
var_145	0.590433	0.056332	10.481	< 2e-16	***
var_146	-1.141506	0.053285	-21.423	< 2e-16	***
var_147	0.731414	0.055979	13.066	< 2e-16	***
var_148	-0.954099	0.053573	-17.809	< 2e-16	***
var_149	-0.839510	0.054569	-15.384	< 2e-16	***
var_150	-0.523622	0.056893	-9.204	< 2e-16	***
var_151	0.525347	0.055615	9.446	< 2e-16	***
var_152	-0.216751	0.054849	-3.952	7.76e-05	***
var_153	-0.097744	0.056442	-1.732	0.083313	.
var_154	-0.909906	0.056629	-16.068	< 2e-16	***
var_155	0.661899	0.054469	12.152	< 2e-16	***
var_156	-0.417920	0.058250	-7.175	7.25e-13	***
var_157	0.556454	0.054309	10.246	< 2e-16	***
var_158	-0.083546	0.056361	-1.482	0.138249	
var_159	0.302811	0.054849	5.521	3.37e-08	***
var_160	-0.059025	0.057777	-1.022	0.306969	
var_161	0.043953	0.056695	0.775	0.438187	
var_162	0.587443	0.055510	10.583	< 2e-16	***

```
var_163     0.576869   0.055151  10.460  < 2e-16 ***
var_164     0.806078   0.056083  14.373  < 2e-16 ***
var_165    -1.033990   0.055634 -18.586  < 2e-16 ***
var_166    -1.097345   0.055357 -19.823  < 2e-16 ***
var_167     0.565415   0.054444  10.385  < 2e-16 ***
var_168     0.223786   0.053523   4.181 2.90e-05 ***
var_169    -0.877389   0.055537 -15.798  < 2e-16 ***
var_170     0.889218   0.055413  16.047  < 2e-16 ***
var_171     0.276615   0.055099   5.020 5.16e-07 ***
var_172    -0.763213   0.054442 -14.019  < 2e-16 ***
var_173     0.859296   0.056051  15.331  < 2e-16 ***
var_174    -1.166066   0.056314 -20.706  < 2e-16 ***
var_175     0.477034   0.055684   8.567  < 2e-16 ***
var_176     0.129705   0.055343   2.344 0.019095 *
var_177    -0.759511   0.055477 -13.691  < 2e-16 ***
var_178    -0.346514   0.056888  -6.091 1.12e-09 ***
var_179     0.836030   0.052923  15.797  < 2e-16 ***
var_180     0.615209   0.055653  11.054  < 2e-16 ***
var_181     0.262909   0.054187   4.852 1.22e-06 ***
var_182    -0.182773   0.057718  -3.167 0.001542 **
var_183    -0.045622   0.057545  -0.793 0.427889
var_184     0.977935   0.056732  17.238  < 2e-16 ***
var_185    -0.041670   0.055654  -0.749 0.454022
var_186    -0.530069   0.054740  -9.683  < 2e-16 ***
var_187     0.259532   0.057357   4.525 6.04e-06 ***
var_188    -0.677937   0.055283 -12.263  < 2e-16 ***
var_189     0.139335   0.057792   2.411 0.015911 *
var_190     1.070327   0.054764  19.544  < 2e-16 ***
var_191     0.855414   0.054831  15.601  < 2e-16 ***
var_192    -0.832867   0.054927 -15.163  < 2e-16 ***
var_193    -0.288680   0.054882  -5.260 1.44e-07 ***
var_194    -0.440375   0.057889  -7.607 2.80e-14 ***
var_195     0.545319   0.053699  10.155  < 2e-16 ***
var_196     0.446712   0.057506   7.768 7.96e-15 ***
var_197    -0.705500   0.055904 -12.620  < 2e-16 ***
var_198    -0.954127   0.053603 -17.800  < 2e-16 ***
 [ reached getOption("max.print") -- omitted 1 row ]
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 104505  on 159999  degrees of freedom
Residual deviance:  74373  on 159799  degrees of freedom
AIC: 74775
```

We have got maximum and minimum error as 3.77 and -2.635. so there
Are not so much variation in the error. Variables like var_0, var_1,var_6 can
explain the variance of target variable much.category like those who have p value>

0.5 like var_7,var_17,var_27 can't explain much about target variable.

null deviance explains how well the target variable can be explained by the model with only coefficients.Residual deviance explains while including all the variables.
null deviance and residual deviance is 104505 and 74373 respectively.
Lower value of residual deviance points out that the model has become better
When it has included 200 variables.The degrees of freedom for null deviance equals N−1, where N is the number of observations in data sample.Here N=160000, therefore N-1=160000-1=159999
The degrees of freedom for residual deviance equals N−k−1, where k is the number of variables and N is the number of observations in data sample.
Here N=160000,k=200 ,therefore N-k-1=160000-200-1=159799

After predicting the model we have done confusion matrix.

```
#                Reference
# Prediction     0      1
#     0        35530   2872
#     1          481   1117
```

True positive (TP)= 1117
True Negative (TN)= 35530
False positive (FP)= 481
False Negative (FN)= 2872

Accuracy= (TP+TN)/(TP+TN+FP+FN)= 36644/39997 =0.9162
Precision= TP/(TP+FP)=1117/(1117+481) =0.699
Recall= TP/(TP+FN)= 1117/(1117+2872) =0.28
FNR= FN/(FN+TP)=2872/3949 = 0.72

auc(test1$target,logit_prediction)

Auc score is 0.633

We will be finding the importance of all the variables and can find out the most important variable.

#varImp(model.logit)

```
          Overall
var_0   17.4943328
var_1   16.6216962
var_2   18.8879261
var_3    2.8051471
var_4    3.2640779
var_5   10.8157945
var_6   23.4647072
var_7    0.2676293
var_8    6.0329347
var_9   14.4795197
```

var_10 0.3668890
var_11 8.4254615
var_12 22.8107416
var_13 18.7234477
var_14 0.9786633
var_15 5.0215840
var_16 2.5325645
var_17 0.4796651
var_18 14.1964034
var_19 3.8191268
var_20 7.0186956
var_21 20.6739355
var_22 20.3065317
var_23 9.2525666
var_24 9.8996693
var_25 4.3070830
var_26 21.0046731
var_27 1.1751149
var_28 8.5741978
var_29 2.5501505
var_30 1.0088403
var_31 9.2275928
var_32 10.5921106
var_33 15.4847495
var_34 18.4353351
var_35 12.6055951
var_36 13.5985708
var_37 2.9202813
var_38 0.1449558
var_39 1.8464009
var_40 17.9991325
var_41 0.8697347
var_42 3.2402786
var_43 8.9451116
var_44 16.7021649
var_45 7.2904771
var_46 1.3590186
var_47 3.2262791
var_48 10.4953465
var_49 9.3154938
var_50 5.7034601
var_51 7.4890630
var_52 9.7384805
var_53 22.4909394
var_54 6.0670045
var_55 6.4327720
var_56 12.0126735
var_57 5.8411484
var_58 8.3099952
var_59 3.3626935

var_60 3.2332337
var_61 2.8534526
var_62 5.1060487
var_63 5.2190280
var_64 4.1410180
var_65 2.8816494
var_66 6.5758326
var_67 14.6928698
var_68 4.4427068
var_69 2.2008406
var_70 9.3808498
var_71 10.8958396
var_72 4.7117628
var_73 1.8724151
var_74 6.1411265
var_75 13.1403556
var_76 21.6790869
var_77 5.7441541
var_78 16.8268646
var_79 1.6921264
var_80 19.3687068
var_81 27.3976719
var_82 7.6214296
var_83 7.5314843
var_84 2.7939916
var_85 7.0434078
var_86 13.0836048
var_87 12.6581637
var_88 5.6697447
var_89 13.7595841
var_90 10.1025756
var_91 13.3163012
var_92 15.8735895
var_93 11.3003488
var_94 16.6270004
var_95 13.2683857
var_96 1.0153616
var_97 5.0856206
var_98 0.8206978
var_99 20.1262377
var_100 0.6016466
var_101 3.3087285
var_102 5.3221356
var_103 0.2645019
var_104 9.4324059
var_105 8.1282862
var_106 11.2410221
var_107 14.4381291
var_108 14.7493843
var_109 16.4984818

var_110 21.0383710
var_111 8.6978428
var_112 7.8333664
var_113 5.7187020
var_114 8.4952168
var_115 16.4823176
var_116 8.7783878
var_117 0.2337222
var_118 13.3812341
var_119 9.7358792
var_120 2.8997689
var_121 14.3635503
var_122 14.3465963
var_123 13.4945755
var_124 1.8675861
var_125 9.7724131
var_126 0.3419749
var_127 13.9855160
var_128 9.9748340
var_129 2.1047374
var_130 10.4729241
var_131 10.0560008
var_132 8.2129136
var_133 17.4143210
var_134 6.0802817
var_135 8.9348592
var_136 1.6147212
var_137 9.6717192
var_138 6.8735531
var_139 24.2063499
var_140 6.4458454
var_141 10.3092333
var_142 6.7179774
var_143 5.0972290
var_144 6.9540895
var_145 10.4813201
var_146 21.4225716
var_147 13.0658851
var_148 17.8093960
var_149 15.3843177
var_150 9.2036832
var_151 9.4461239
var_152 3.9517731
var_153 1.7317775
var_154 16.0678954
var_155 12.1519514
var_156 7.1746357
var_157 10.2460596
var_158 1.4823429
var_159 5.5207890

var_160 1.0216032
var_161 0.7752583
var_162 10.5825726
var_163 10.4599100
var_164 14.3729344
var_165 18.5857310
var_166 19.8228806
var_167 10.3853062
var_168 4.1810968
var_169 15.7981505
var_170 16.0470482
var_171 5.0202836
var_172 14.0188302
var_173 15.3306790
var_174 20.7064929
var_175 8.5667786
var_176 2.3436760
var_177 13.6906072
var_178 6.0911461
var_179 15.7969581
var_180 11.0543592
var_181 4.8518899
var_182 3.1666741
var_183 0.7928085
var_184 17.2376770
var_185 0.7487258
var_186 9.6834486
var_187 4.5248944
var_188 12.2630764
var_189 2.4109610
var_190 19.5442657
var_191 15.6010012
var_192 15.1630800
var_193 5.2600071
var_194 7.6071714
var_195 10.1550245
var_196 7.7681537
var_197 12.6197947
var_198 17.7997623
var_199 8.0825687

From the above chart we can see that var_81 is the most important variable with score of 27.3976719.

RANDOM FORSET:-

RF_model = randomForest(target ~ ., data=train1, importance = TRUE, ntree = 50)

We build a random forest model with no of tress 50. After predicting we ca see auc Score is
auc score came as 0.5006

The auc score is too low than random forest model.

Build a confusion matrix on top that and four the below result.

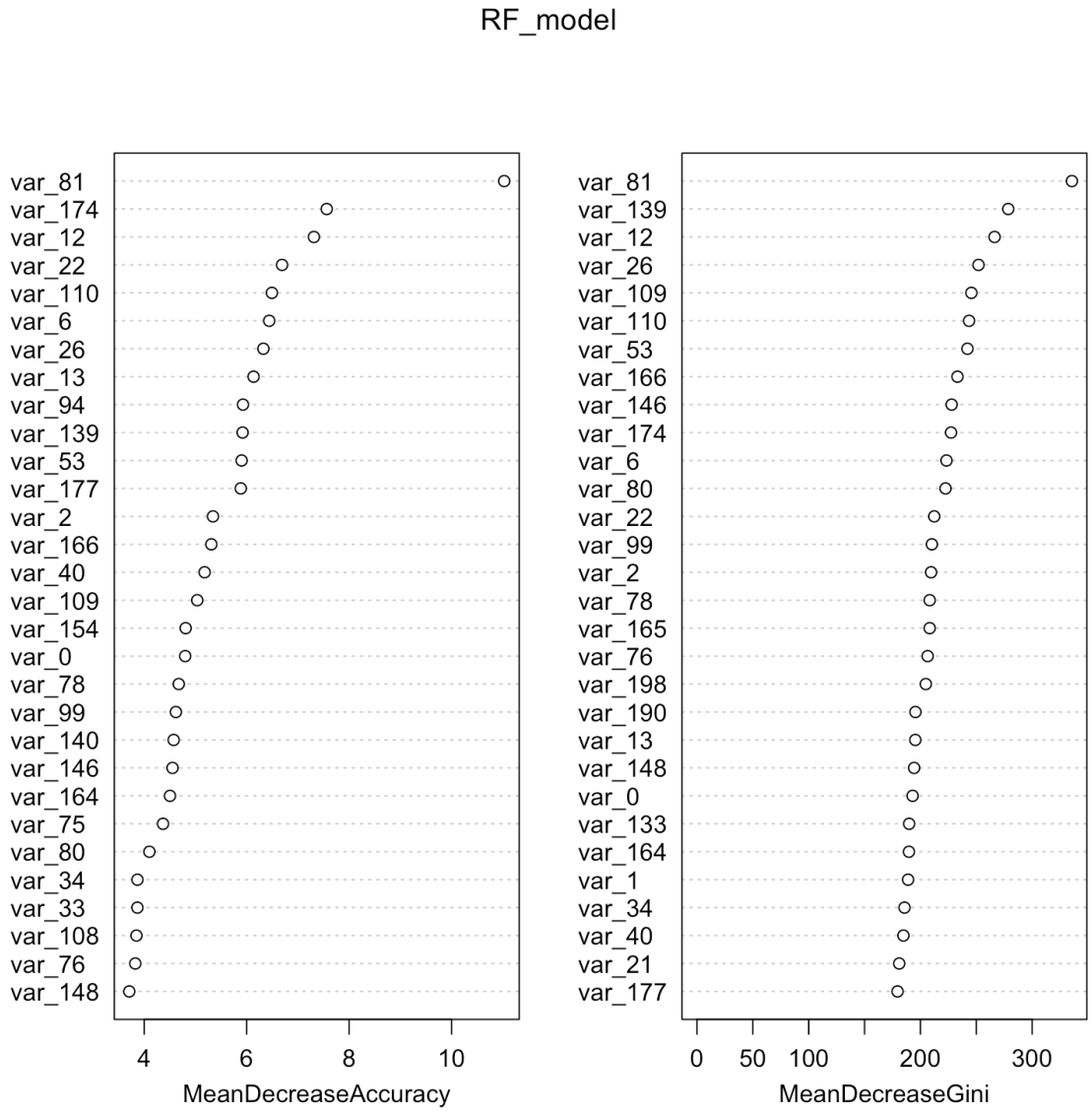
```
#              Reference
#Prediction      0      1
#          0   36011  3984
#          1     0     5
```

True positive (TP)= 5
True Negative (TN)= 36011
False positive (FP)= 0
False Negative (FN)= 3984

Accuracy= (TP+TN)/(TP+TN+FP+FN)= 36644/39997 =0.9004
Precision= TP/(TP+FP)=1117/(1117+481) = 1
Recall= TP/(TP+FN)= 1117/(1117+2872) =0.0012
FNR= FN/(FN+TP)=2872/3949 = 0.99

If we compare randomforest model with logistic regression model FNR is too high for randomforest model.

#varImpPlot(RF_model)



From the above variable importance plot we can see that var_81 is the most important variable.

3. Conclusion :-

3.1 Model Evaluation:-

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

We have done several model building in R and in python. Based on different parameters we have chosen roc_auc_score and f1 score as the model evolution metrics. F1 score is the harmonic mean of precision and recall.

3.2 Model Selection :-

On based of above metrics, we will choose the stacking model which is raw_model (based on original data with any outlier analysis and normalisation) as roc_auc_score of this model is 0.79 whereas F1 score is 0.51. The precision and recall of this model is 0.4 and 0.72.

3.3 Improvement of Model :-

We can do some other techniques and can used other different algorithms with different hyperparameter tuning for better model. Some way of improvements are as follows.

1. Trying building other models like LGB (Light Gradient boosting method) with hyperparameter tuning.
2. KNN (distance based algorithm) can also be tried out. As all the detests consists of numeric variable, so it might give good result with different parameter tuning like no of k values etc.
3. Base models like KNN, LGBM, XGBoost can be included in stacking model.
4. Based on important plot features we can drop some variables and create a new dataset and can build a model on top that.

4. SUMMARIZATION :-

In this project the Santander is asking to predict the customer who will make a transaction in the future.

We have evaluated this model based on roc_auc_score.

AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s. The ROC curve is plotted with TPR against the FPR where TPR is on y-axis and FPR is on the x-axis.
Our final model has roc_auc_score is 0.79. We should be able to predict right customer as much as possible so that the company can get better service to the right customers. Along with that we should not miss out those customers who have done transaction but our model has failed to identify them and the company shall loose those customers and start making transaction with the competitors . In that case FNR should be low as much as possible.

5. INSTRUCTION TO RUN PYTHON AND R CODE :-

PYTHON CODE:-

The 'Santander Customer Transaction Prediction-kaggle.ipynb' are attached along with the project. One can go to cell and click on the command 'run all' . The code will run automatically. Just in case while doing randomizedsearchcv if the result of the best parameter comes different from the existing one then just we have to change the parameters in the model based on tuning and run run the code.

```
bd_train=pd.read_csv("/Users/subhadeep/Downloads/train (1).csv")
bd_test=pd.read_csv("/Users/subhadeep/Downloads/test (1).csv")
```

While loading the file into Jupyter Notebook from the location of the data folder (train and test data), we have to pass the location.

R CODE:-

```
# bd_train=read.csv("/Users/subhadeep/Downloads/train (1).csv")
# bd_test=read.csv("/Users/subhadeep/Downloads/test (1).csv")
```

After giving the right path of the training and testing data for downloading into R we can select all the codes will execute automatically.