Santander Customer Transaction Prediction

Date - 31/08/2020

Submitted By:

Ravi Sharma

Data Scientist
B.E. Mechanical 2015

Santander Customer Transaction Prediction

Project Report

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?

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.

Problem Classification:

This problem belongs to the binary classification supervised learning category.

Metrics for Evaluation:

This is a classification problem & to evaluate the model predictions the metrics to be used are Precision Score, Recall Score & AUC (ROC-AUC) Score.

- **Precision Score:**

  The precision is intuitively the ability of the classifier not to label as positive a sample that is negative or it is the ratio of correctly predicted positives by overall predicted positive.

  Calculated by: **tp / (tp + fp)**

  Where **tp** is the number of true positives and **fp** the number of false positives.

  The best value is 1 and the worst value is 0.

- **Recall Score:**

  The recall is intuitively the ability of the classifier to find all the positive samples or it is the ratio of correctly predicted positives by overall actual positive.

  Calculated by: **tp / (tp + fn)**

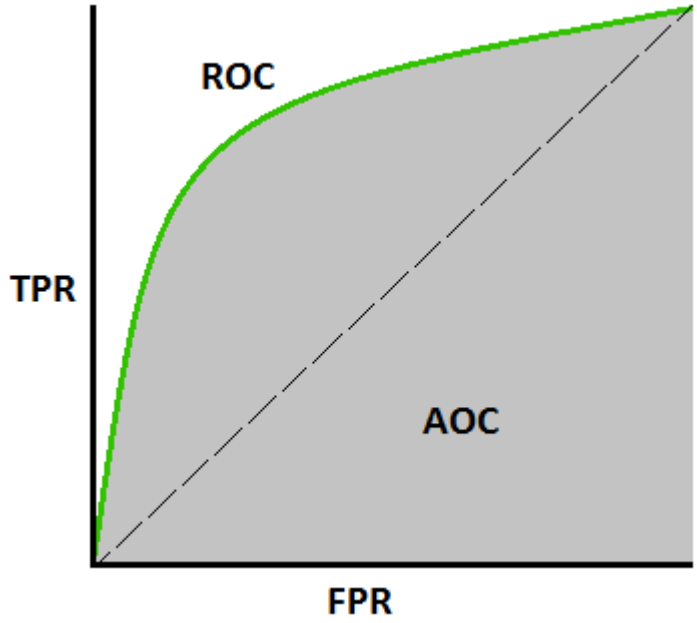
  Where **tp** is the number of true positives and **fn** the number of false negatives.

  The best value is 1 and the worst value is 0.

- **AUC (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 (True Positive rate) against the FPR (False Positive rate) where TPR is on y-axis and FPR is on the x-axis.



$$\text{TPR /Recall / Sensitivity} = \frac{TP}{TP + FN}$$

$$\text{FPR} = 1 - \text{Specificity} = \frac{FP}{TN + FP}$$

$$\text{Specificity} = \frac{TN}{TN + FP}$$

An excellent model has AUC near to the 1 which means it has good measure of separability.

A poor model has AUC near to the 0 which means it has worst measure of separability. In fact it means it is reciprocating the result. It is predicting 0s as 1s and 1s as 0s.

And when AUC is 0.5, it means model has no class separation capacity whatsoever.

Introduction to Data:

- We have been provided with an anonymised train & test dataset (.csv format).

```
Shape of Train dataset is: (200000, 202)
Shape of Test dataset is: (200000, 201)
```

- Train dataset contains 200000 records, 200 numerical variables [var_0, var_1, var_2,….var_199], a binary ‘target’ variable & a string ‘ID_code’ variable.

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7
0	train_0	0	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266
1	train_1	0	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338
2	train_2	0	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155
3	train_3	0	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250
4	train_4	0	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514

- Test dataset contains 200000 records, 200 numerical variables [var_0, var_1, var_2,….var_199 & a string ‘ID_code’ variable

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8
0	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337
1	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131
2	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233
3	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755
4	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890

Data Pre Processing:

Dropping the string column **‘ID_code’**, as this feature only represents the index of the observations in the given dataset & have no impact in prediction of the target variable.

```
Shape of train dataset after dropping the column is: (200000, 201)
```

Exploratory Data Analysis & Visualizations:

1. **Missing Value Analysis:**
   Any blank observation in data is known as missing value. These missing values create issues in performing model training as model needs numerical value. To deal with this issue, we have to perform either deletion of observed row containing missing value or imputation at the place of missing value.

   Imputation of missing value can done through several ways i.e. by Mean, Median, Mode, Min, Max, K-NN, or other domain specific knowledge based value.

   Checking for missing values in the data.

1. Missing Value Check

```
In [11]: # Training Data
         train_data.isna().sum().sum()
Out[11]: 0

In [12]: #Testing Data
         test_data.isna().sum().sum()
Out[12]: 0
```

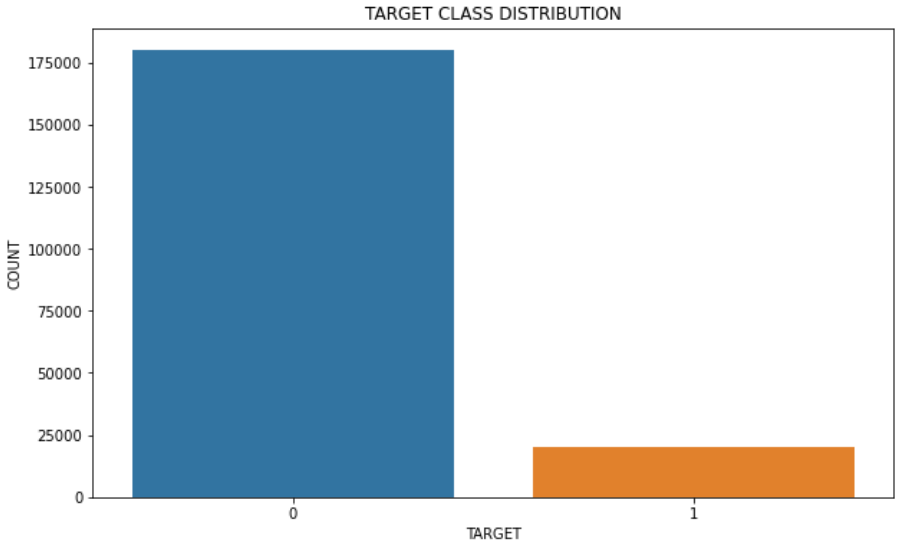
Observations from above:

```
No missing values in train or test data.
```

2. Target Class Balance Check:

The target class is a binary classification, i.e. it contains classes 0 & 1 only. The unique counts of these classes is:

Class	Counts	Percentage of Total
0	179902	89.95%
1	20098	10.05%



Observations from above:

The above distribution shows that the dataset is an imbalanced Dataset & have majority of class 0 i.e. 90% of data is for class 0.

3. Data Description:

	count	mean	std	min	25%	50%	75%	max
target	200000.0	0.100490	0.300653	0.0000	0.000000	0.00000	0.000000	1.0000
var_0	200000.0	10.679914	3.040051	0.4084	8.453850	10.52475	12.758200	20.3150
var_1	200000.0	-1.627622	4.050044	-15.0434	-4.740025	-1.60805	1.358625	10.3768
var_2	200000.0	10.715192	2.640894	2.1171	8.722475	10.58000	12.516700	19.3530
var_3	200000.0	6.796529	2.043319	-0.0402	5.254075	6.82500	8.324100	13.1883
...	...	...	...	...	...	...	...	...
var_195	200000.0	-0.142088	1.429372	-5.2610	-1.170700	-0.17270	0.829600	4.2729
var_196	200000.0	2.303335	5.454369	-14.2096	-1.946925	2.40890	6.556725	18.3215
var_197	200000.0	8.908158	0.921625	5.9606	8.252800	8.88820	9.593300	12.0004
var_198	200000.0	15.870720	3.010945	6.2993	13.829700	15.93405	18.064725	26.0791
var_199	200000.0	-3.326537	10.438015	-38.8528	-11.208475	-2.81955	4.836800	28.5007

Observations from above:

Mean of some variables are very high & difference between mean and max for some variables is quite high too. Which shows the presence of outliers in the dataset.

4. Outlier Analysis:

An outlier is an element of a data set that distinctly stands out from the rest of the data. In other words, outliers are those data points that lie outside the overall pattern of a distribution. These outliers affect the modelling algorithms, as they shift the weightage towards them.

These outliers can either be removed or Imputed with a values similar to missing value analysis.

Finding outliers columwise by deriving MAX, MIN bound limits using inner quartile range & then imputing with the mean value.

```
#Detect and replace outliers with mean
for i in train_data.drop(columns='target').columns:

    q75, q25 = np.percentile(train_data.loc[:,i], [75 ,25])
    iqr = q75 - q25
    minimum = q25 - (iqr*1.5)
    maximum = q75 + (iqr*1.5)

    train_data[i][train_data[i] < minimum]= np.nan
    train_data[i][train_data[i] > maximum]= np.nan
    print(i)
    print('Max out bound: ',maximum)
    print('Min out bound: ',minimum)
    print('Total No. of Outliers: ',train_data[i].isna().sum())
    train_data[i].fillna(train_data[i].mean(),inplace=True)
```

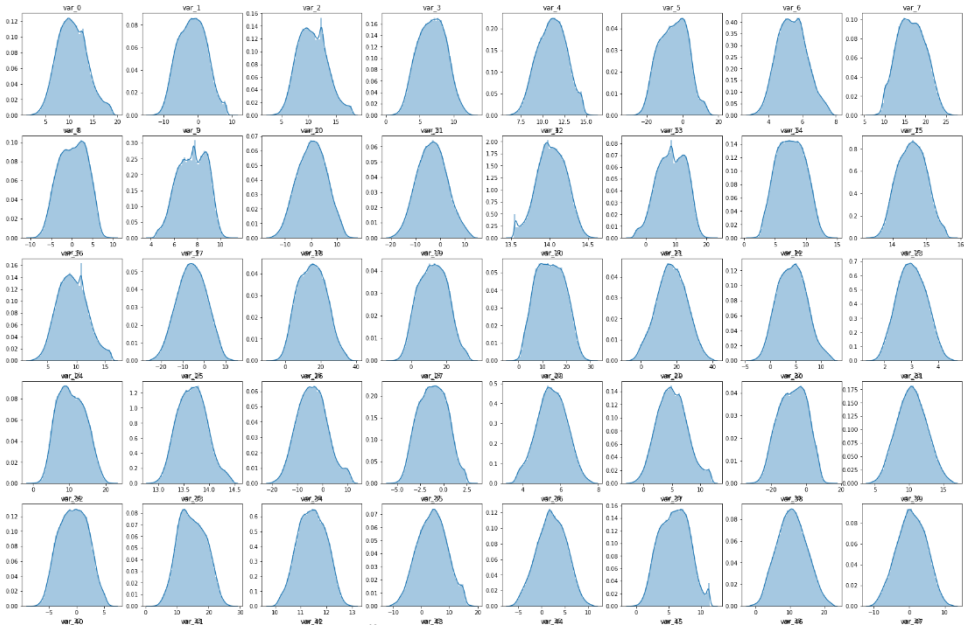
```
var_0
Max out bound:  19.214725
Min out bound:  1.9973250000000018
Total No. of Outliers:  104
var_1
Max out bound:  10.5066
Min out bound:  -13.888000000000002
Total No. of Outliers:  6
var_2
Max out bound:  18.208037500000003
Min out bound:  3.031137499999998
Total No. of Outliers:  49
var_3
Max out bound:  12.9291375
Min out bound:  0.6490375000000013
Total No. of Outliers:  22
var_4
Max out bound:  15.828050000000005
Min out bound:  6.316249999999997
Total No. of Outliers:  76
```

Observations from above:

Total outliers found in train data is 26536. These Outliers are then replaced by the column's mean value.

5. Numerical Data Distribution:

Distribution of few variables:

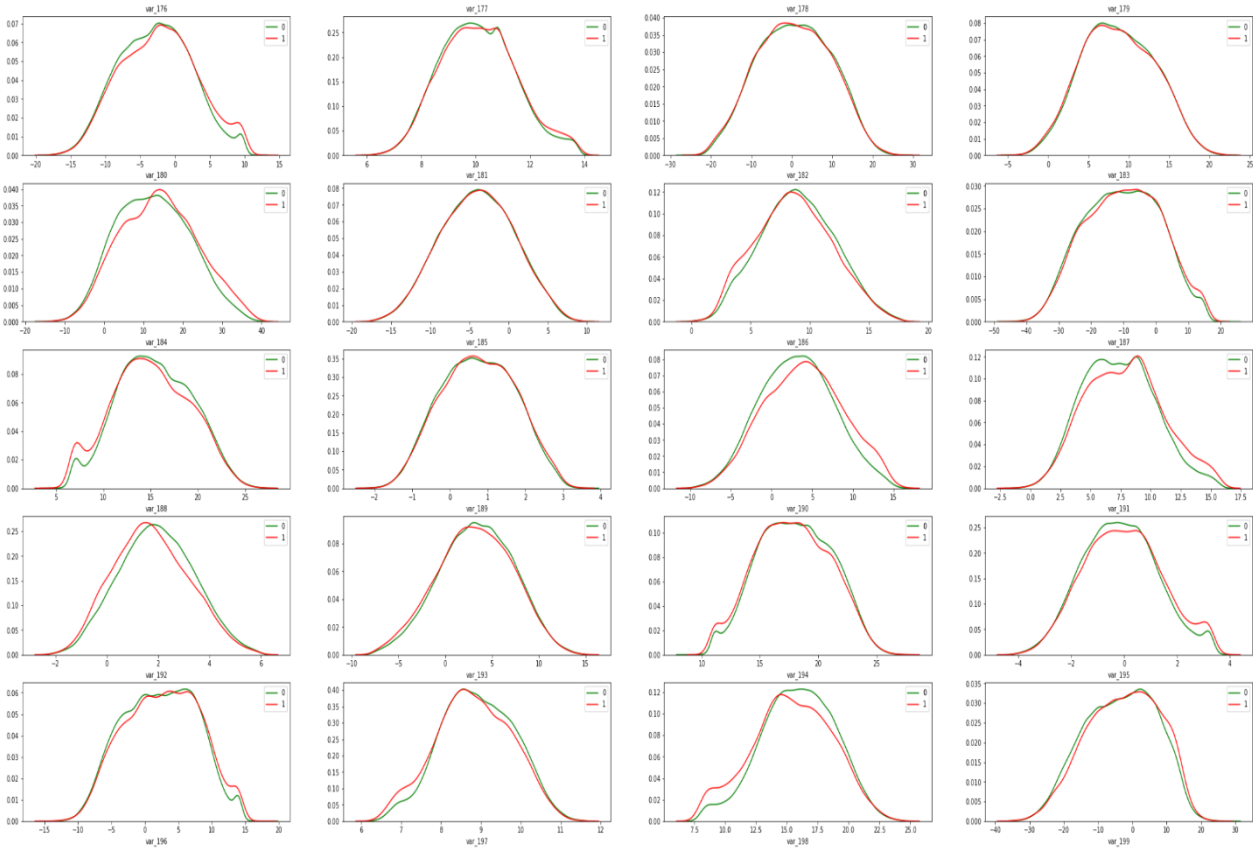


Observations from above:

All the features in Dataset, pretty much follows normalised distribution.

6. Numerical Data Distribution per Target Class

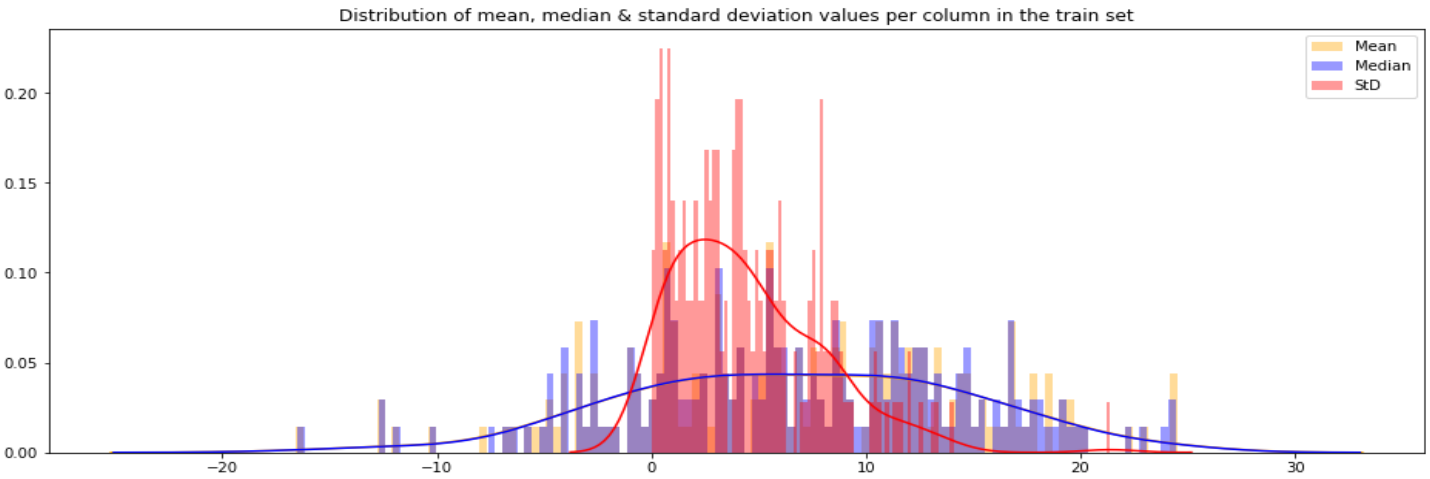
Distribution of few variables:



Observations from above:

Numerical columns almost follows same distribution for both classes.

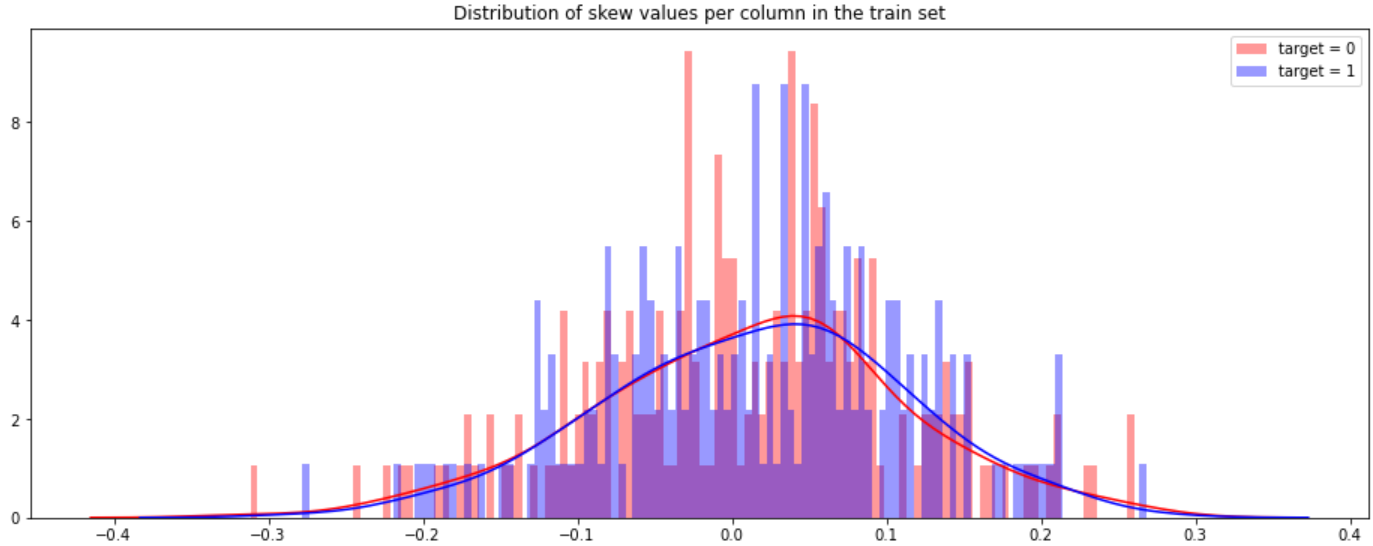
7. Distribution of Mean, Median & Standard Deviation



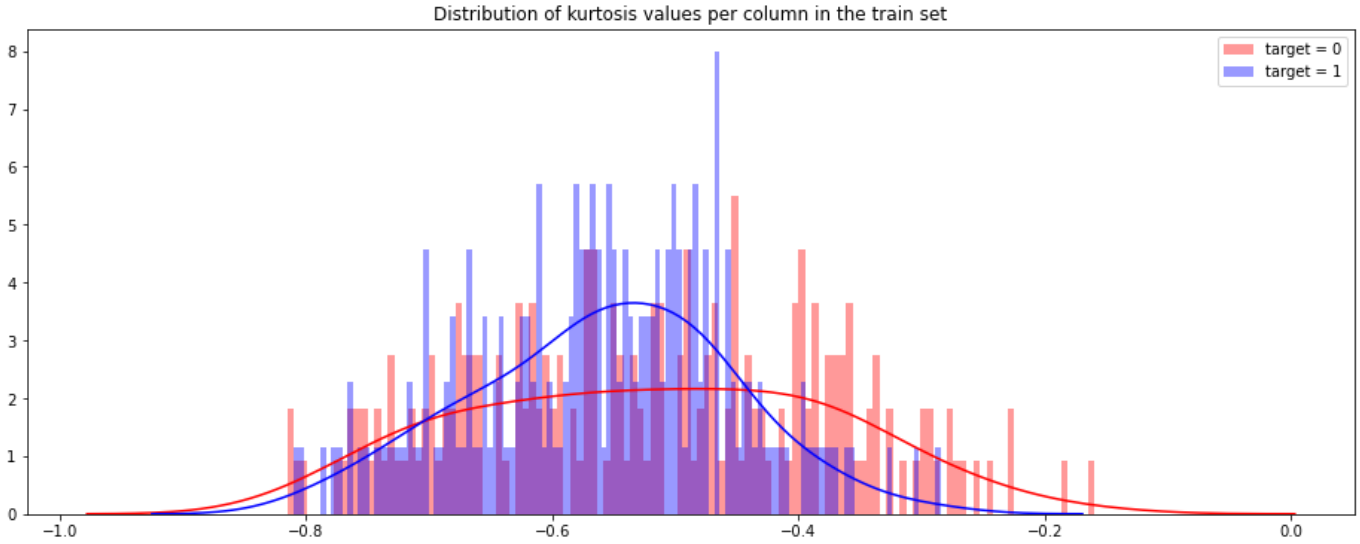
Observations from above:

- Mean values are distributed over a large range.
- Moreover mean and median have similar distribution.
- Standard deviation is relatively large.

8. Distribution of Skew



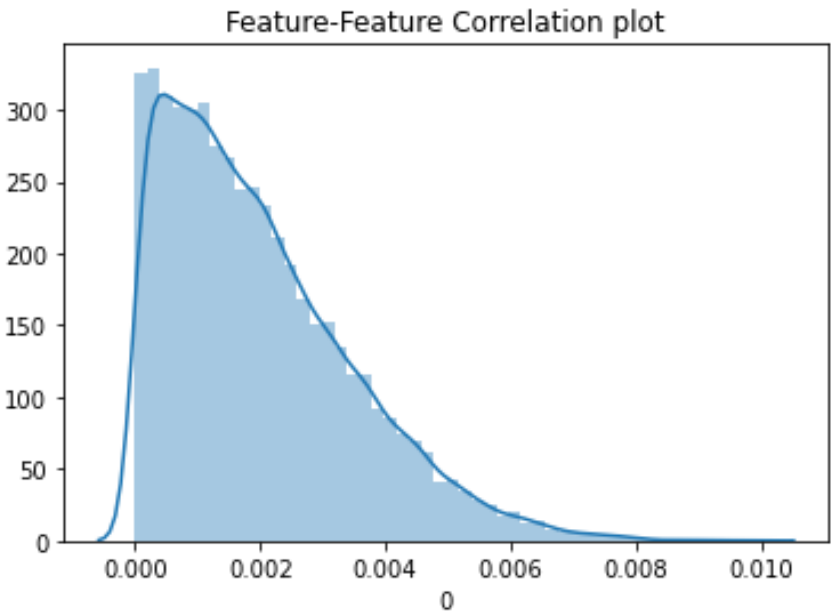
9. Distribution of Kurtosis



10. Features Correlation with each other

	level_0	level_1	0
0	var_61	var_125	3.420113e-08
1	var_125	var_61	3.420113e-08
2	var_114	var_172	5.401101e-08
3	var_172	var_114	5.401101e-08
4	var_46	var_109	1.139129e-07
...	...	...	...
39795	var_53	var_148	9.590880e-03
39796	var_139	var_26	9.778775e-03
39797	var_26	var_139	9.778775e-03
39798	var_81	var_165	9.927379e-03
39799	var_165	var_81	9.927379e-03

39800 rows × 3 columns



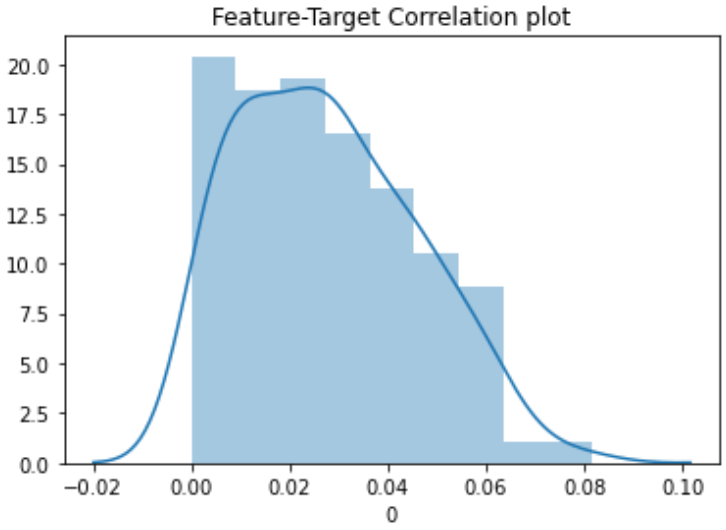
Observations from above:

All Features have inter feature correlation value less than 0.02, Thus features are not correlated to each other.

11. Feature – Target Correlation

	level_0	level_1	0
201	target	var_81	0.081642
204	target	var_139	0.074199
205	target	var_12	0.069573
207	target	var_6	0.066868
210	target	var_53	0.063411
...	...	...	...
29735	target	var_38	0.000847
30391	target	var_17	0.000794
32750	target	var_27	0.000598
33042	target	var_30	0.000575
40163	target	var_185	0.000017

200 rows × 3 columns



Top 10 most correlated variable:

	level_0	level_1	0
201	target	var_81	0.081642
204	target	var_139	0.074199
205	target	var_12	0.069573
207	target	var_6	0.066868
210	target	var_53	0.063411
212	target	var_110	0.063176
214	target	var_26	0.062412
215	target	var_174	0.061616
217	target	var_76	0.061187
220	target	var_146	0.060801

Top 10 least correlated variable:

	level_0	level_1	0
40163	target	var_185	0.000017
33042	target	var_30	0.000575
32750	target	var_27	0.000598
30391	target	var_17	0.000794
29735	target	var_38	0.000847
24466	target	var_41	0.001298
23411	target	var_126	0.001393
22589	target	var_103	0.001473
15489	target	var_10	0.002195
15111	target	var_100	0.002239

Observations from above:

- 10 Most Correlated Feature with Target: [var_81, var_139, var_12, var_6, var_53, var_110, var_26, var_174, var_76, var_146].
- 10 Least Correlated Feature with Target: [var_185, var_30, var_27, var_17, var_38, var_41, var_126, var_103, var_10, var_100]

12. Feature Selection:

It is a process of selecting only the required and decisive features which mainly defines the target variable.

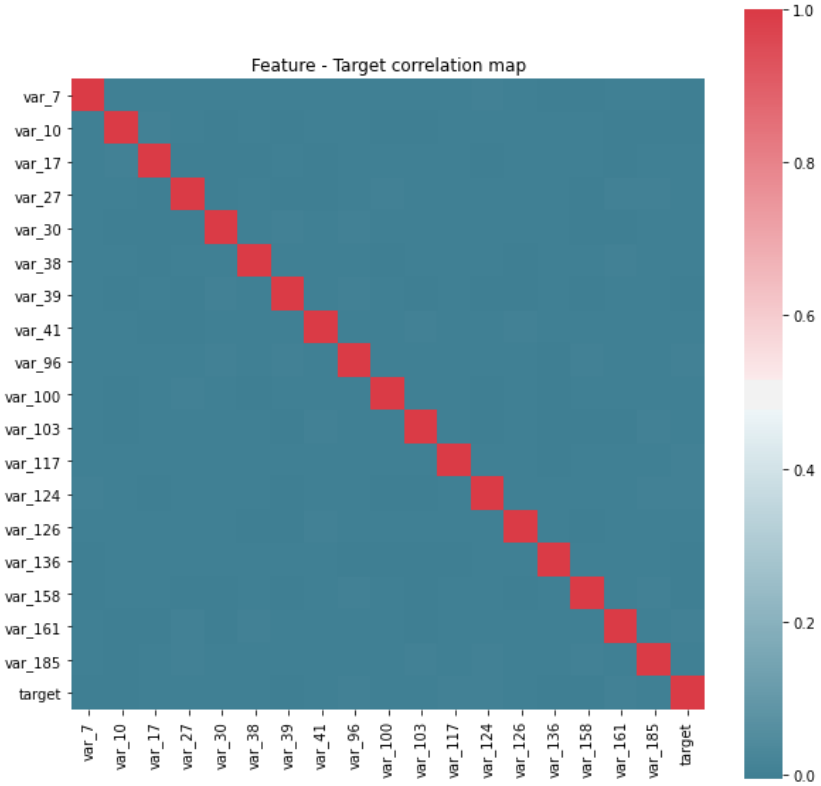
Performing Pearson Correlation & Significance Hypothesis Testing to determine the feature which have a relationship with the target variable:

```
Assuming Significance Level, SL is 0.05,

Null Hypothesis: Feature and Target doesn't have Linear Relationship:
if calc_SL > assumed_SL.

Alternate Hypothesis: Feature and target have Linear Relationship:
if calc_SL <= assumed_SL.
```

Feature variable ['var_7', 'var_10', 'var_17', 'var_27', 'var_30', 'var_38', 'var_39', 'var_41', 'var_96', 'var_98', 'var_100', 'var_103', 'var_117', 'var_126', 'var_136', 'var_158', 'var_161', 'var_185'] fail to reject NULL hypothesis.



Dropping feature variables ['var_7', 'var_10', 'var_17', 'var_27', 'var_30', 'var_38', 'var_39', 'var_41', 'var_96', 'var_98', 'var_100', 'var_103', 'var_117', 'var_126', 'var_136', 'var_158', 'var_161', 'var_185'] from train data, because they have negligible impact on target data.

Shape of data after dropping columns: `(200000, 183)`

13. Feature Scaling:

As the features almost follows normalized distribution, performing Standardization on the dataset to scale down the features values in an appropriate range which is centred to zero to lower down the variance of features to same scale. This is done to equalize the magnitude of features values so that no feature will dominate the calculations during modelling like Logistic Regression etc.

We are using **StandardScaler** from scikit learn module to standardized the data.

Data after Standardization

Train Data after Standardization:

	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_8	var_9
0	0	-0.578232	-1.274019	0.452533	-0.834318	0.235040	-0.536552	-0.335609	-1.561692	-1.473796
1	0	0.270735	-0.622323	1.191961	-0.689377	0.791456	1.539999	0.245257	0.858986	0.419300
2	0	-0.682477	-0.276201	0.517883	0.536644	-0.306996	-0.511153	1.774526	-1.561482	-1.307408
3	0	0.125608	-0.129539	-0.667924	0.195299	0.928723	0.410651	0.502082	-1.844037	0.548767
4	0	-0.277759	0.035521	0.818892	-0.078032	0.738993	0.955611	0.615109	1.794813	0.090006

Train, Validation Data Split

Using train_test_split from scikit learn module, we split the data into 80-20 ratio by stratifying the data on Target Class basis.

Shape of train data after split:

x_train	x_test	y_train	y_test
(160000, 182)	(40000, 182)	(160000,)	(40000,)

Model Development:

For modelling we are using 4 different algorithms, to train on data & predict for validation data. On the basis of metric scores the best algorithm will be selected. The algorithms are:

1. Logistic Regression.

Logistic regression is a statistical model for predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

We are using **LogisticRegression** model from sklearn.linear_model module.

Metric Scores of model prediction on validation data

```
ROC_AUC Score:  0.63
Precision Score:  0.68
Recall Score:  0.27
```

2. Decision Tree Classifier.

Decision tree classifier is a supervised learning algorithm used for classification problem. A decision tree is a flowchart like tree structure, where each node in the **tree** specifies a test on an attribute, each branch descending from that node corresponds to one of the possible values for that attribute.

We are using **DecisionTreeClassifier** model from sklearn.tree module.

Metric Scores of model prediction on validation data

```
ROC_AUC Score:  0.59
Precision Score:  0.19
Recall Score:  0.2
```

3. Random Forest Classifier.

Random Forest Classifier is an ensemble learning method for classification that operates by creating decision trees on randomly selected data samples gets prediction from each tree and selects the best solution by means of voting

We are using **RandonForestClassifier** model from sklearn.ensemble module.

Metric Scores of model prediction on validation data

```
ROC_AUC Score:  0.51
Precision Score:  0.49
Recall Score:  0.02
```

4. Naïve Bayes.

Naïve Bayes is a statistical technique based on Bayes theorem. It is one of the simplest supervised learning algorithms. Naïve Bayes Classifier is the fast, accurate & reliable algorithm. Naïve Bayes classifier have high accuracy & speed on large Datasets.
Naive Bayes classifier assumes that the effect of a particular feature in a class is independent of other features.

We are using **GaussianNB** model from sklearn. naive_bayes module.

Metric Scores of model prediction on validation data

```
ROC_AUC Score:  0.67
Precision Score:  0.71
Recall Score:  0.36
```

Observations from above modelling:

- **On the basis of Precision, Recall & ROC_AUC score, the best performing model is Naive Bayes model.**

  ROC_AUC Score: 0.67

  Precision Score: 0.71

  Recall Score: 0.36

- **Selecting Naïve Bayes Algorithms for further processes.**

The Precision & ROC_AUC score are quite average, but Recall score is quite low, which means there is High False Negative Rate.

Since the data contains imbalanced target class, we are getting high false negative rate in our predictions. The above algorithms designed in a way that they favours the majority class in predictions. So to improve the predictions scores we have to deal with imbalanced situation.

There are 2 ways to deal with imbalanced dataset:

1. Sampling Techniques. [Over Sampling, Under Sampling, SMOTE Sampling].
2. Using Ensemble Technique based Algorithms (LightGBM).

Treating Imbalanced Dataset

1. Sampling Techniques.

- **Over sampling the lower class with Naïve Bayes model.**

  Over sampling technique is to randomly sample the minority class data repeatedly up to the length of majority class data, so that the dataset should contains equal no of cases for both the classes.

  We are using **resample** module from scikit.utils to sample the data.

  - ROC_AUC Score: 0.81
  - Precision Score: 0.81
  - Recall Score: 0.8

- **Under sampling the higher class with Naïve Bayes model.**

  Under sampling technique is to randomly sample the majority class data equal to the length of minority class data, so that the dataset should contains equal no of cases for both the classes.

  We are using **resample** module from scikit.utils to sample the data.

  - ROC_AUC Score: 0.8
  - Precision Score: 0.8
  - Recall Score: 0.79

- **SMOTE sampling with Naïve Bayes model.**

  Synthetic Minority Over-sampling Technique (SMOTE) is an oversampling technique that generates synthetic samples from the minority class. It is used to obtain a synthetically class-balanced or nearly class-balanced training set, which is then used to train the classifier.

  We are using **SMOTE** module from imblearn.over_sampling to sample the data.

  - ROC_AUC Score: 0.53
  - Precision Score: 0.19
  - Recall Score: 0.12

Observations from above:

Over Sampling & Under Sampling performs almost equally, while SMOTE performs poorly. Selecting Over sampling with Naive Bayes Model for further processes.

2. **Ensemble technique-LightGBM Algorithm.**
   LightGBM is a Gradient Boosting ensemble model which is faster in speed & accuracy as compared to other boosting models. It is capable of performing equally good with large datasets with a significant reduction in training time as compared to other boosting models.
   It require parameter tuning to provide good result.

   Parameters (Tuned) opted for modelling:

```
param = {
    'bagging_freq': 5,  'bagging_fraction': 0.5,  'boost_from_average':False,
    'boost': 'gbdt',    'feature_fraction': 0.08, 'learning_rate': 0.01,
    'max_depth': -1,    'metric':'auc',             'min_data_in_leaf': 80,
    'min_sum_hessian_in_leaf': 10.0,'num_leaves': 50,  'num_threads': 20,
    'tree_learner': 'serial',   'objective': 'binary',       'verbosity': 1,
    'max_bin': 100, 'subsample_for_bin': 100, 'subsample': 1,
    'subsample_freq': 1, 'colsample_bytree': 0.8, 'min_split_gain': 0.45,
    'min_child_weight': 1, 'min_child_samples': 5, 'is_unbalance':True,
}
```

```
ROC_AUC Score:  0.95
Precision Score:  0.93
Recall Score:  0.98
```

Observations from above:

On the basis of Precision, Recall & ROC_AUC Score, the best performing model is LightGBM model.

Dumping Models

The best performing model is LightGBM model with oversampled Data. Thus saving the LightGBM model along with the list of columns to drop & StandardScaler model by using pickle module to be used for the deployment of the prediction model.

```
pickle.dump(lgb_model,open('Santander_Prediction_model.model','wb'))

pickle.dump(col_reduced,open('columns_to_drop.list','wb'))

pickle.dump(scaler,open('scaler.model','wb'))
```

Conclusion

This was a classification problem on a typically unbalanced dataset with no missing values. Predictor variables are anonymous and numeric and target variable is categorical. Visualising descriptive features and finally I got to know that these variables are not correlated among themselves. After that I decided to treat imbalanced dataset and built different models with original data and chosen LightGBM as my final model with final value of AUC-Score is 0.95.