Project Name – Santander Customer Transaction Prediction

Abhishek sharma

3 September 2019

Contents

1 Introduction

1.1 Problem statement

1.2 Exploratory Data Analysis

* Loading dataset and libraries
* Data cleaning
* Typecasting the attributes

* Target classes count
* Missing value analysis

2. Attributes Distributions and trends

* Distribution of train attributes
* Distribution of test attributes
* Mean distribution of attributes
* Standard deviation distribution of attributes
* Skewness distribution of attributes
* Kurtosis distribution of attributes
* Outliers analysis

3. Correlation matrix

4. Split the dataset into train and test dataset

5. Modelling the training dataset

* Logistic Regression Model
* ROSE Model

* LightGBM Model

6. Cross Validation Prediction

* Logistic Regression CV Prediction

* ROSE CV Prediction

*LightGBM CV Prediction

7. Model performance on test dataset

* Logistic Regression Prediction

* ROSE Prediction

* LightGBM Prediction

8. Model Evaluation Metrics

* Confusion Matrix

* ROC_AUC score

9. Choosing best model for predicting customer transaction

Refrences

Chapter 1

Introduction

1.1 Problem Statement

this project, the bank is asking data scientists to do binary classification using machine learning algorithms. Typical binary classification problems include: can a customer pay this loan? Is this transaction fraud? Will a customer make a specific transaction?Two data sets: training set and testing set are given. In training data set, there are 202 columns including: ID code, target and 200 variables. In testing data set, there are 201 columns excluding target comparing with training data set. Both data sets have 200000 clients. The goal is to predict the target value (0/1) for testing set.

1.2 Exploratory Data Analysis

Loading the data sets:

Filter Cols: 1 - 50

	ID_code	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10
1	train_0	1	8.9255	-6.7863	11.9081	5.0930	11.4607	-9.2834	5.1187	18.6266	-4.9200	5.7470	2.9252
2	train_1	1	11.5006	-4.1473	13.8588	5.3890	12.3622	7.0433	5.6208	16.5338	3.1468	8.0851	-0.403
3	train_2	1	8.6093	-2.7457	12.0805	7.8928	10.5825	-9.0837	6.9427	14.6155	-4.9193	5.9525	-0.324
4	train_3	1	11.0604	-2.1518	8.9522	7.1957	12.5846	-1.8361	5.8428	14.9250	-5.8609	8.2450	2.3061
5	train_4	1	9.8369	-1.4834	12.8746	6.6375	12.2772	2.4486	5.9405	19.2514	6.2654	7.6784	-9.445
6	train_5	1	11.4763	-2.3182	12.6080	8.6264	10.9621	3.5609	4.5322	15.2255	3.5855	5.9790	0.8010
7	train_6	1	11.8091	-0.0832	9.3494	4.2916	11.1355	-8.0198	6.1961	12.0771	-4.3781	7.9232	-5.128
8	train_7	1	13.5580	-7.9881	13.8776	7.5985	8.6543	0.8310	5.6890	22.3262	5.0647	7.1971	1.4532
9	train_8	1	16.1071	2.4426	13.9307	5.6327	8.8014	6.1630	4.4514	10.1854	-3.1882	9.0827	0.9501
10	train_9	1	12.5088	1.9743	8.8960	5.4508	13.6043	-16.2859	6.0637	16.8410	0.1287	7.9682	0.8787

Untitled1 × santandar project.R* × train_df × test_df ×

Filter Cols: 1 - 50

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11
1	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	8.8100	-2.0248	-4.355
2	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	5.9739	-1.3809	-0.331
3	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	8.3442	-4.7057	-3.042
4	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	7.4578	0.0095	-5.065
5	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	7.1437	5.1025	-3.282
6	test_5	5.9862	-2.2913	8.6058	7.0685	14.2465	-8.6761	4.2467	14.7632	1.8790	7.2842	-4.9194	-9.186
7	test_6	8.4624	-6.1065	7.3603	8.2627	12.0104	-7.2073	4.1670	13.0809	-4.3004	6.3181	3.3959	-2.020
8	test_7	17.3035	-2.4212	13.3989	8.3998	11.0777	9.6449	5.9596	17.8477	-4.8068	7.4643	4.0355	1.618
9	test_8	6.9856	0.8402	13.7161	4.7749	8.6784	-13.7607	4.3386	14.5843	2.5883	7.2215	9.3750	8.404
10	test_9	10.3811	-6.9348	14.6690	9.0941	11.9058	-10.8018	3.4508	20.2816	-1.4112	6.7401	0.3727	-4.191

< Dimension of train data: 200000 202

< Summary of the dataset

< Typecasting the target variable

#convert to factor

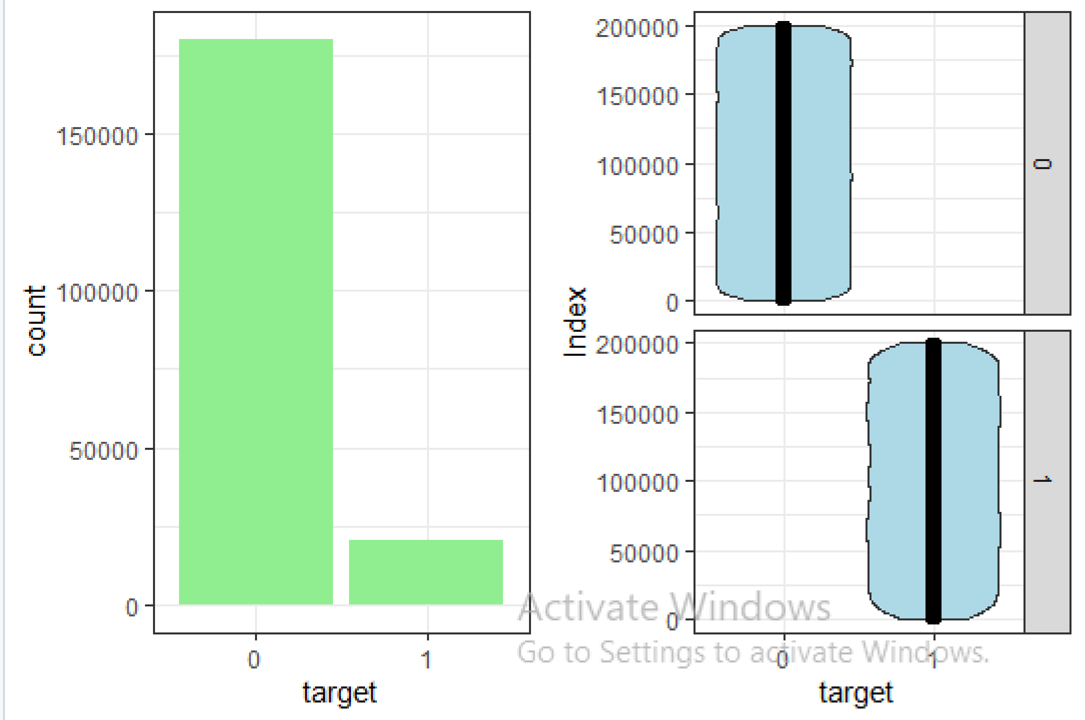
<Target classes count in train data

#Count of target classes

#Percenatge counts of target classes

#Bar plot for count of target classes

#Violin with jitter plots for target classes

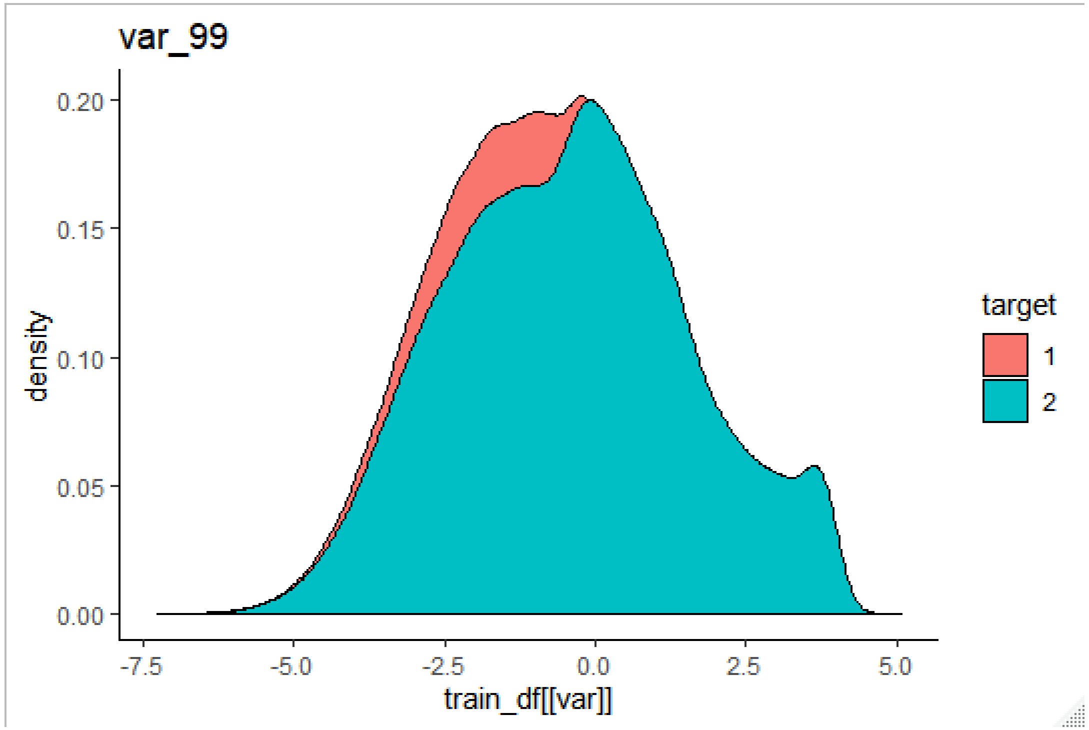


Take aways:

< We have a unbalanced data,where 90% of the data is the data of number of customers those will not make a transaction and 10% of the data is those who will make a transaction.
< Look at the violin plots seems that their is no relationship between the target with the index of the train dataframe. This is more dominated by the zero targets then for the ones.
< Look at the jitter plots with violin plots. We can observed that targets looks uniformaly distributed over the indexs of the dataframe.

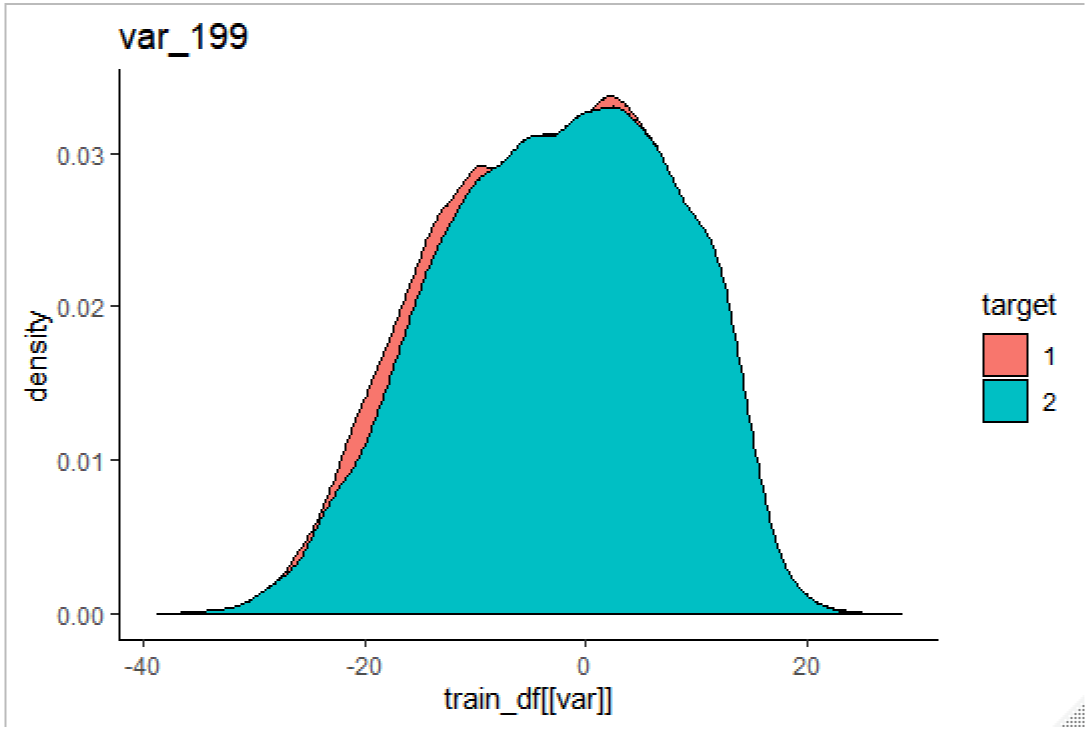
< distribution of train attributes from 3 to 102

1 to 99 bar plot



< distribution of train attributes from 103 to 202

103 to 199 plot



Take aways:

(a) We can observed that their is a considerable number of features which are significantly have different distributions for two target variables. For example like var_0,var_1,var_9,var_198 var_180 etc.

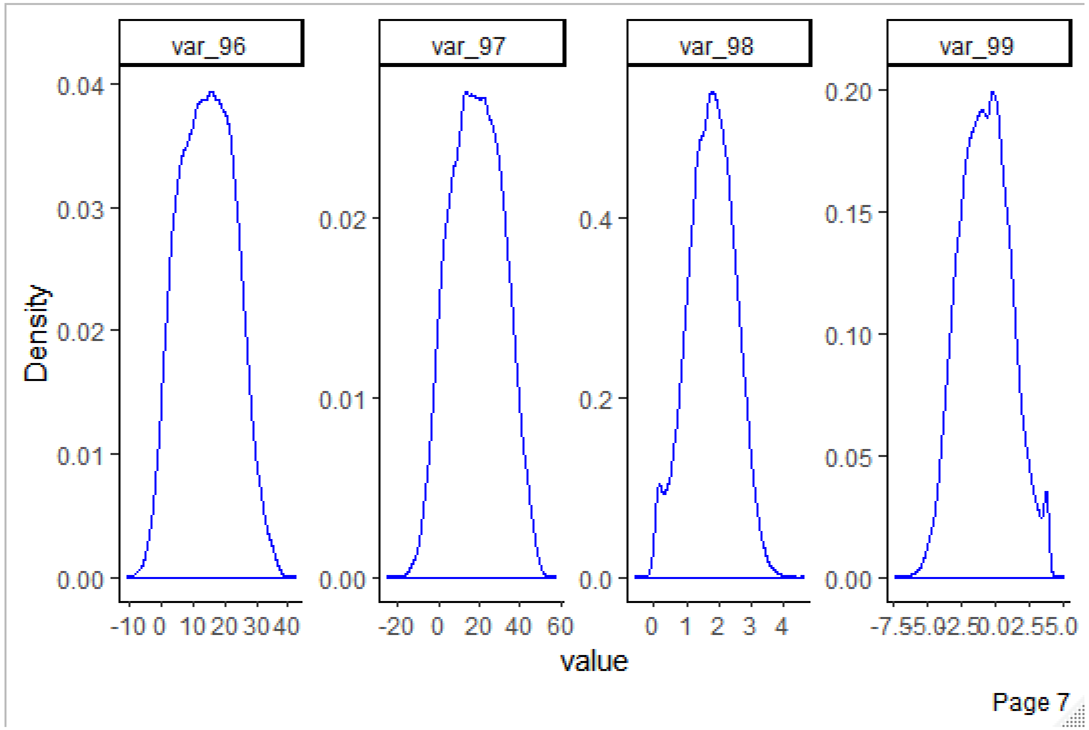
(b) We can observed that their is a considerable number of features which are significantly have same distributions for two target variables. For example like var_3,var_7,var_10,var_171,var_185 etc.

Importing the test data

	ID_code	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11
1	test_0	11.0656	7.7798	12.9536	9.4292	11.4327	-2.3805	5.8493	18.2675	2.1337	8.8100	-2.0248	-4.355
2	test_1	8.5304	1.2543	11.3047	5.1858	9.1974	-4.0117	6.0196	18.6316	-4.4131	5.9739	-1.3809	-0.331
3	test_2	5.4827	-10.3581	10.1407	7.0479	10.2628	9.8052	4.8950	20.2537	1.5233	8.3442	-4.7057	-3.042
4	test_3	8.5374	-1.3222	12.0220	6.5749	8.8458	3.1744	4.9397	20.5660	3.3755	7.4578	0.0095	-5.065
5	test_4	11.7058	-0.1327	14.1295	7.7506	9.1035	-8.5848	6.8595	10.6048	2.9890	7.1437	5.1025	-3.282
6	test_5	5.9862	-2.2913	8.6058	7.0685	14.2465	-8.6761	4.2467	14.7632	1.8790	7.2842	-4.9194	-9.186
7	test_6	8.4624	-6.1065	7.3603	8.2627	12.0104	-7.2073	4.1670	13.0809	-4.3004	6.3181	3.3959	-2.020
8	test_7	17.3035	-2.4212	13.3989	8.3998	11.0777	9.6449	5.9596	17.8477	-4.8068	7.4643	4.0355	1.618
9	test_8	6.9856	0.8402	13.7161	4.7749	8.6784	-13.7607	4.3386	14.5843	2.5883	7.2215	9.3750	8.404
10	test_9	10.3811	-6.9348	14.6690	9.0941	11.9058	-10.8018	3.4508	20.2816	-1.4112	6.7401	0.3727	-4.191

< Dimension of test dataset: 200000, 201

< Let us see distribution of test attributes from 2 to 101 , after that



<< Let us see distribution of test attributes from 102 to 201, then


Attribute
Attribute
Attribute
Attribute
Attribute
Attribute
Attribute
Attribute
Attribute
Attribute

Take aways:

(a) We can observed that their is a considerable number of features which are significantly have different distributions. For example like var_0,var_1,var_9,var_180 var_198 etc.

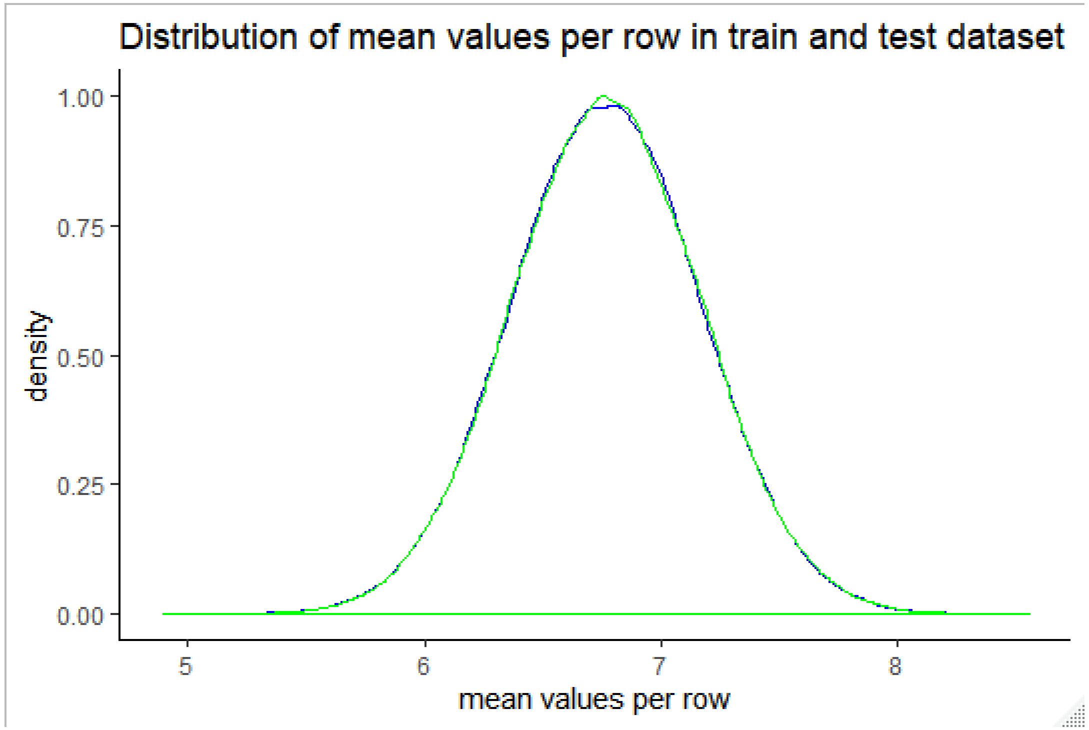
(b) We can observed that their is a considerable number of features which are significantly have same distributions. For example like var_3,var_7,var_10,var_171,var_185,var_192 etc.

< Let us see distribution of mean values per row and column in train and test dataset

#Applying the function to find mean values per row in train and test data.

#Distribution of mean values per row in train data

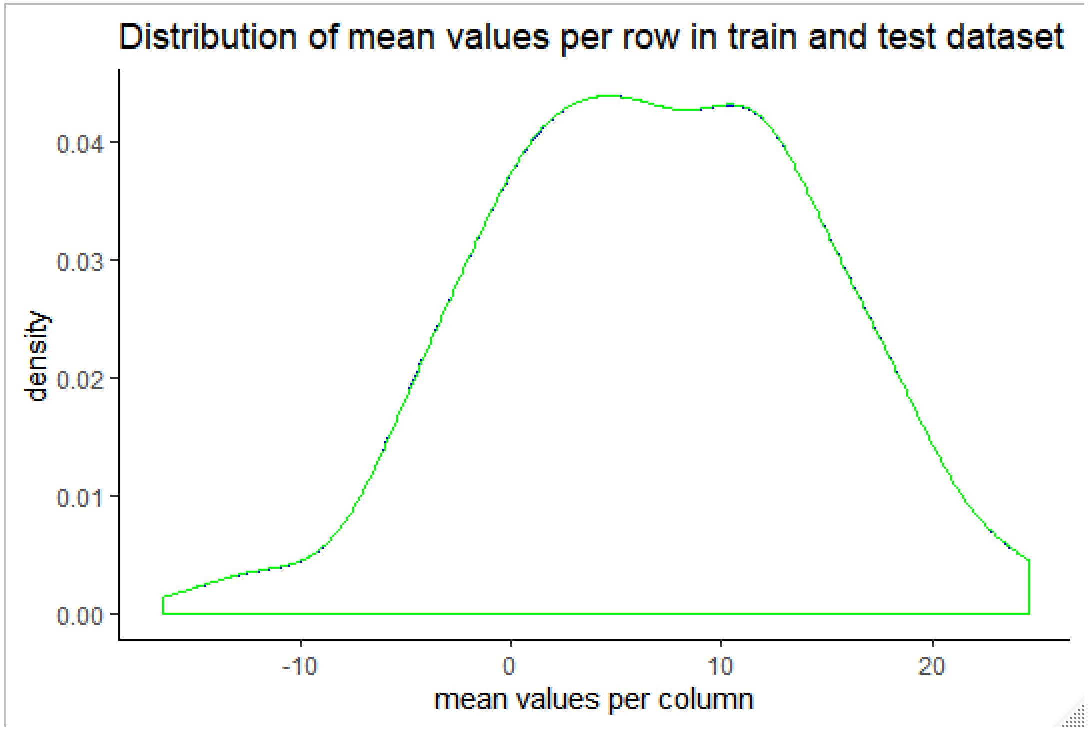
#Distribution of mean values per row in test data



<#>Applying the function to find mean values per column in train and test data.

#Distribution of mean values per column in train data

#Distribution of mean values per column in test data

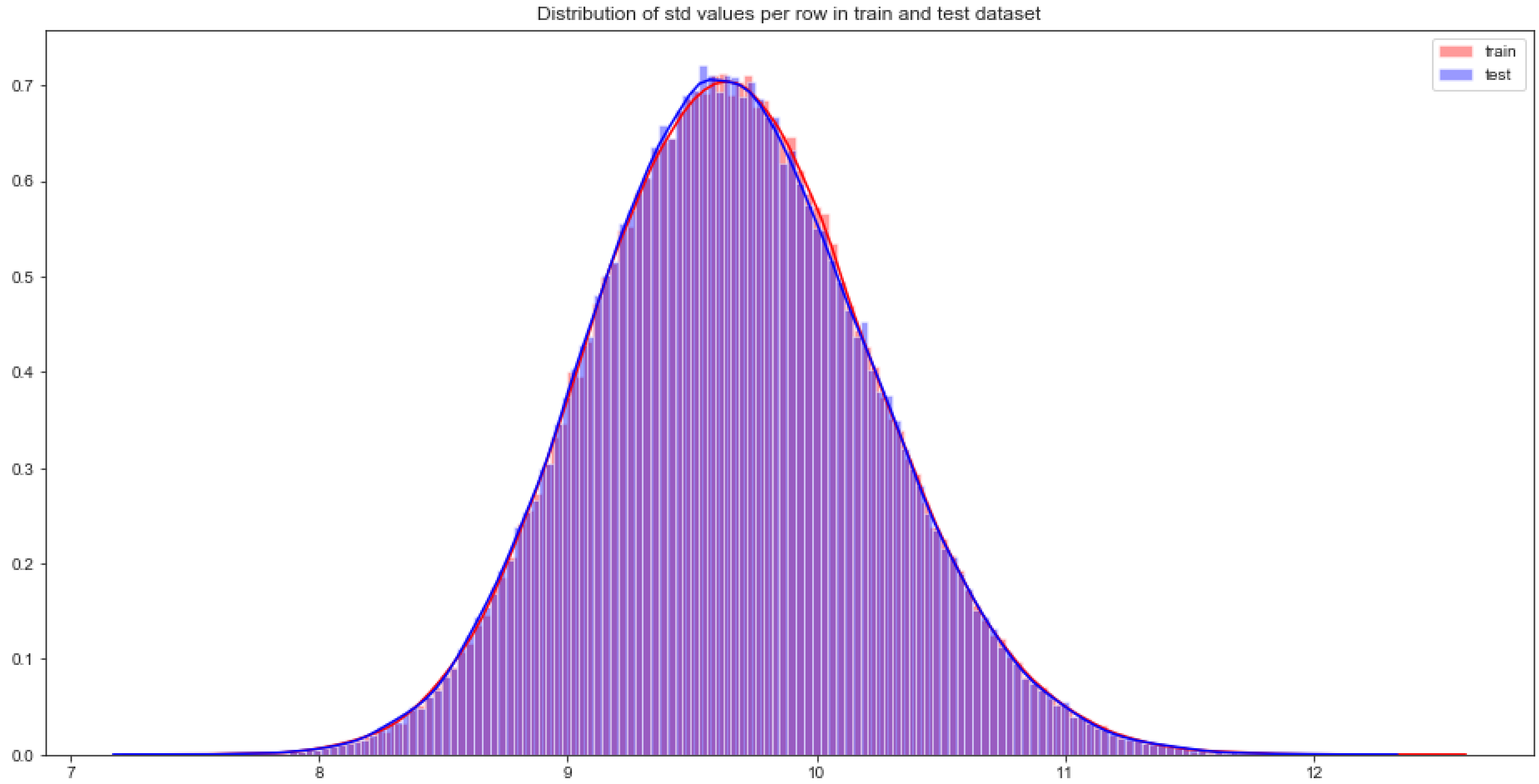


<< Let us see distribution of standard deviation values per row and column in train and test dataset

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

#Distribution of sd values per row in train data

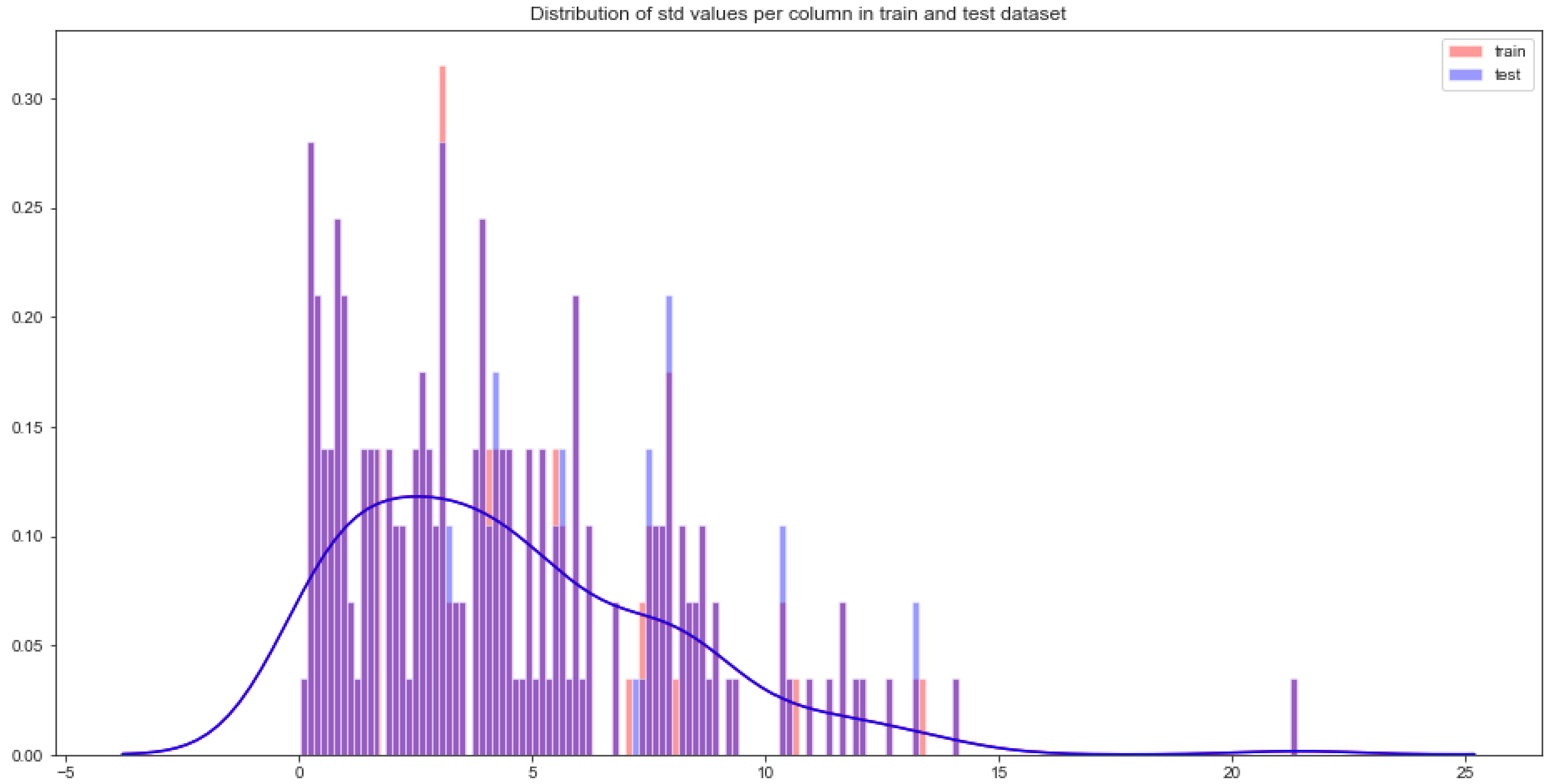
#Distribution of sd values per row in test data



<#>Applying the function to find sd values per column in train and test data.

#Distribution of sd values per column in train data

#Distribution of sd values per column in test data

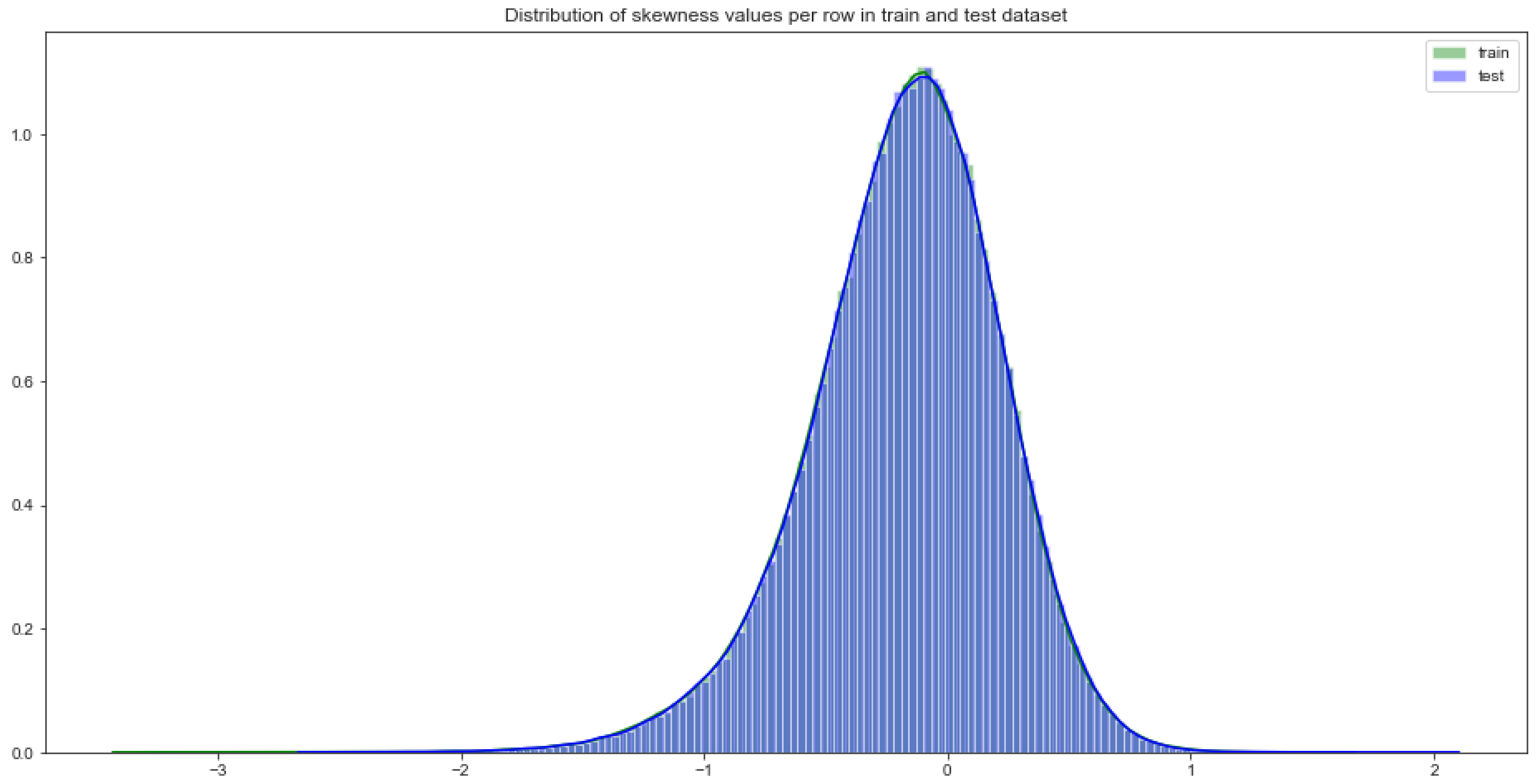


<< Let us see distribution of skewness values per row and column in train and test dataset

#Applying the function to find skewness values per row in train and test data.

#Distribution of skewness values per row in train data

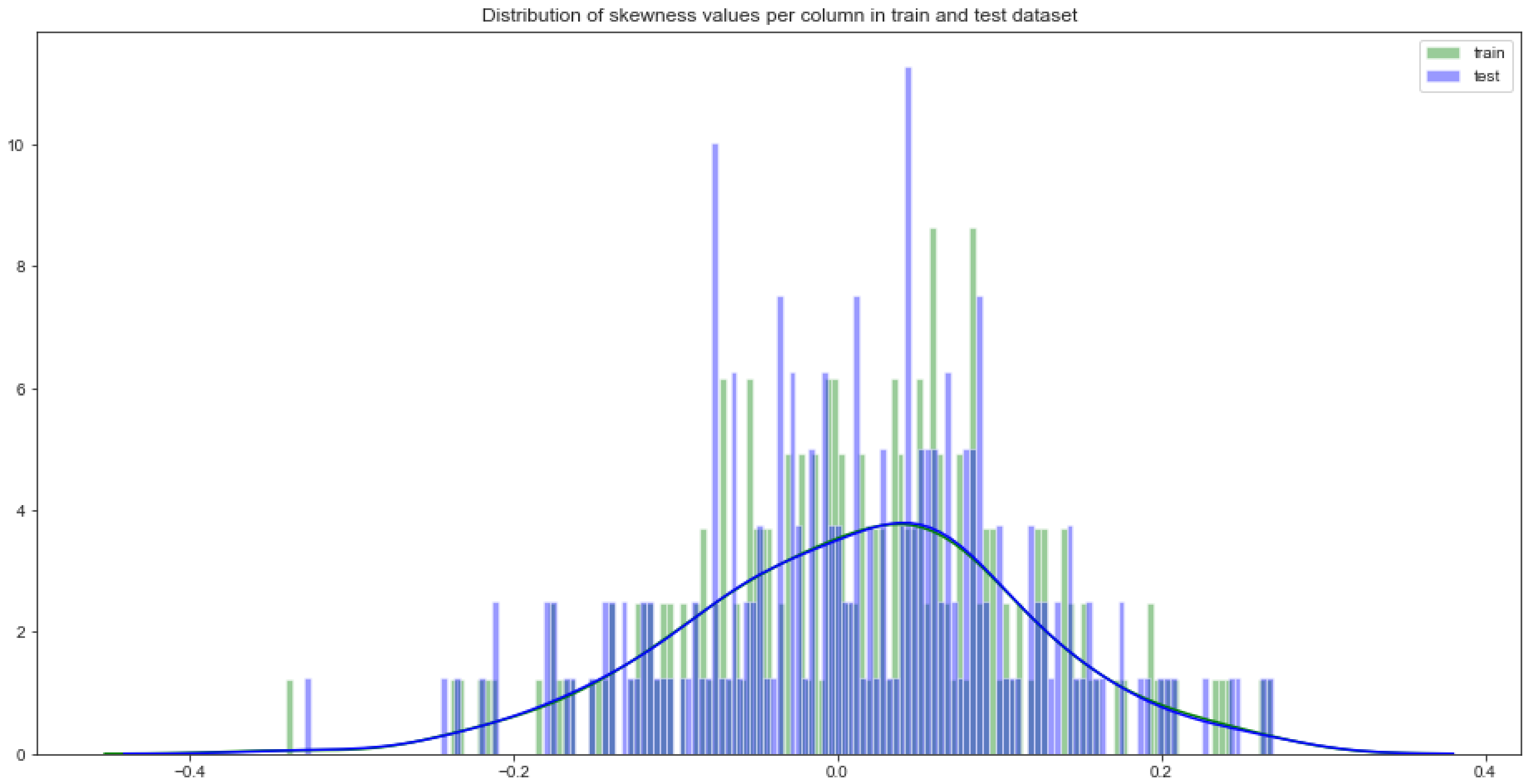
#Distribution of skewness values per ruw in test data



<#>Applying the function to find skewness values per column in train and test data.

#Distribution of skewness values per column in train data

#Distribution of skewness values per column in test data

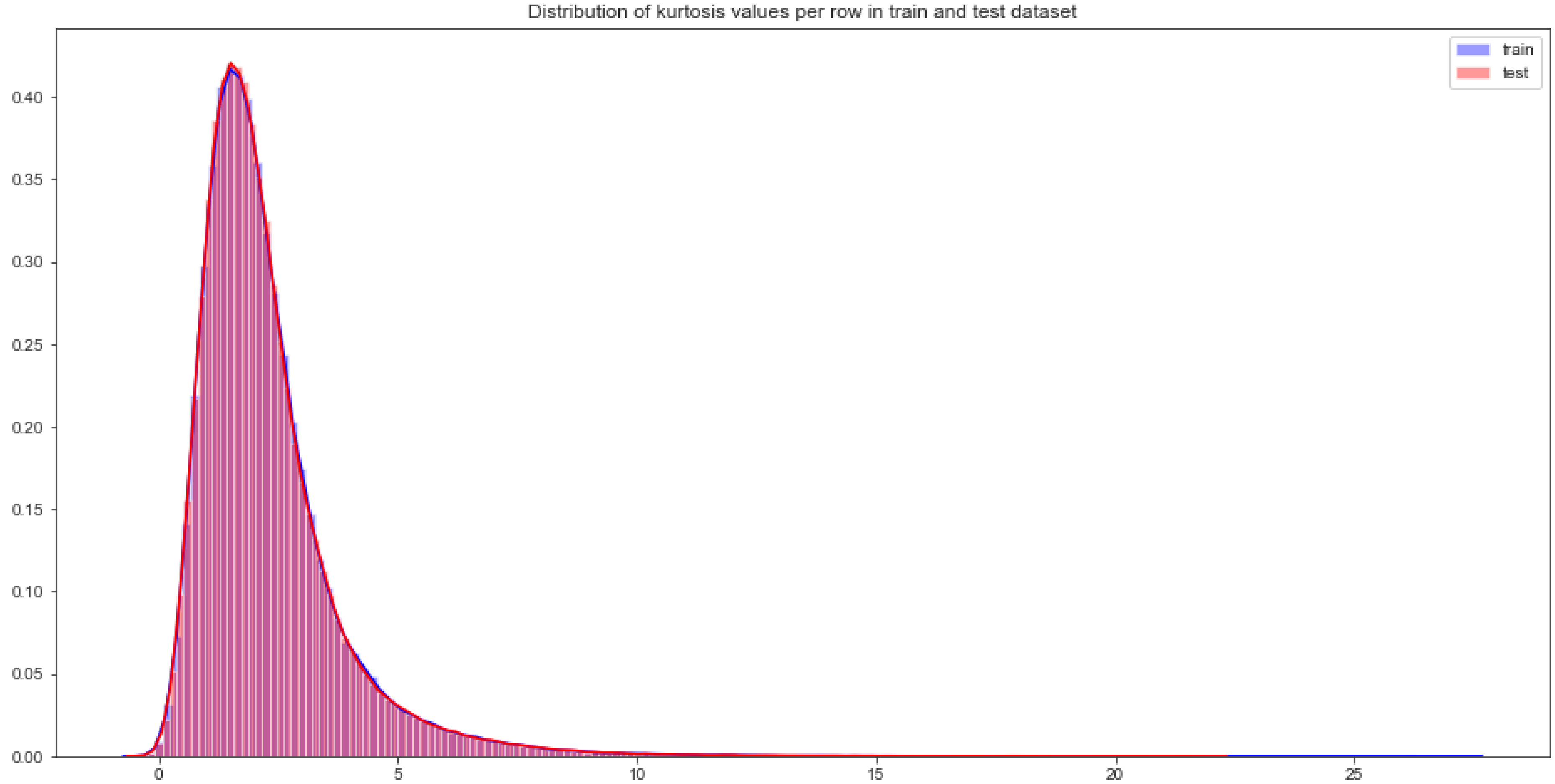


<< Let us see distribution of kurtosis values per row and column in train and test dataset

#Applying the function to find kurtosis values per row in train and test data.

#Distribution of kurtosis values per row in train data

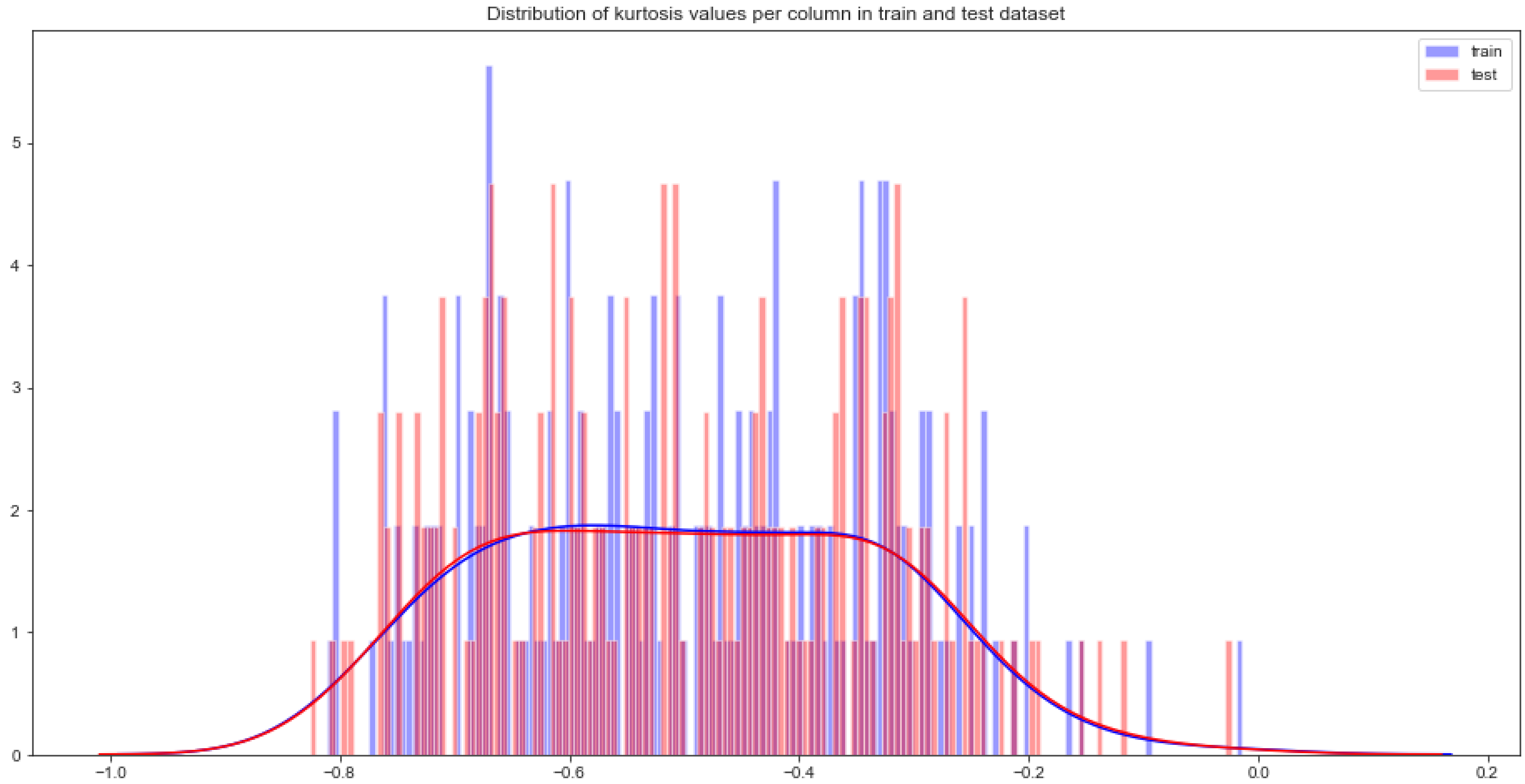
#Distribution of kurtosis values per row in test data



<#>Applying the function to find kurtosis values per column in train and test data.

#Distribution of kurtosis values per column in train data

#Distribution of kurtosis values per column in test data



<<Let us do Missing value analysis: In statistics, missing data, or missing values, occur when no data value is stored for the variable in an observation. Missing data are a common occurrence and can have a significant effect on the conclusions that can be drawn from the data.

#Finding the missing values in train data - 0

#Finding the missing values in test data - 0

No missing values are present in both train and test data.

Now Let us see

<< Correlation: **correlation** is a statistical measure that indicates the extent to which two or more variables fluctuate together. A positive **correlation** indicates the extent to which those variables increase or decrease in parallel; a negative **correlation** indicates the extent to which one variable increases as the other decreases.

Correlations in train data

We can observed that the correlation between the train attributes is very small.

```
%%time

#Correlations in train attributes

train_attributes=train_df.columns.values[2:202]

train_correlations=train_df[train_attributes].corr().abs().unstack().sort_values(kind='quicksort').reset_index()

train_correlations=train_correlations[train_correlations['level_0']!=train_correlations['level_1']]

print(train_correlations.head(10))

print(train_correlations.tail(10))
```

```
level_0 level_1 0 0 var_75 var_191 2.703975e-08 1 var_191 var_75 2.703975e-08 2 var_173 var_6 5.942735e-08 3 var_6 var_173 5.942735e-08 4 var_126 var_109 1.313947e-07 5 var_109 var_126 1.313947e-07 6 var_144 var_27 1.772502e-07 7 var_27 var_144 1.772502e-07 8 var_177 var_100 3.116544e-07 9 var_100 var_177 3.116544e-07 level_0 level_1 0 39790 var_183 var_189 0.009359 39791 var_189 var_183 0.009359 39792 var_174 var_81 0.009490 39793 var_81 var_174 0.009490 39794 var_81 var_165 0.009714 39795 var_165 var_81 0.009714 39796 var_53 var_148 0.009788 39797 var_148 var_53 0.009788 39798 var_26 var_139 0.009844 39799 var_139 var_26 0.009844 Wall time: 36.7 s
```

Correlations in test data

We can observed that the correlation between the test attributes is very small.

```
level_0 level_1 0 0 var_154 var_175 1.477268e-07 1 var_175 var_154 1.477268e-07 2 var_188 var_113
1.639749e-07 3 var_113 var_188 1.639749e-07 4 var_131 var_8 4.695407e-07 5 var_8 var_131 4.695407e-07 6
var_60 var_189 9.523709e-07 7 var_189 var_60 9.523709e-07 8 var_159 var_96 1.147835e-06 9 var_96 var_159
1.147835e-06 level_0 level_1 0 39790 var_122 var_164 0.008513 39791 var_164 var_122 0.008513 39792 var_164
var_2 0.008614 39793 var_2 var_164 0.008614 39794 var_31 var_132 0.008714 39795 var_132 var_31 0.008714
39796 var_96 var_143 0.008829 39797 var_143 var_96 0.008829 39798 var_139 var_75 0.009868 39799 var_75
var_139 0.009868 Wall time: 35.4 s
```

<<Feature engineering: Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. Feature engineering is fundamental to the application of machine learning, and is both difficult and expensive.

Let us do some feature engineering by using

- Permutation importance
- Partial dependence plots

Variable importance

Variable importance is used to see top features in dataset based on mean decreses gini.

<<Let us build simple model to find features which are more important.

#Split the training data using simple random sampling

#train data

#validation data

#dimension of train and validation data

<<Random forest classifier

rf_model=RandomForestClassifier(n_estimators=10,random_state=42)

#fitting the model

rf_model.fit(X_train,y_train)

#Training the Random forest classifier

#convert to int to factor

#setting the mtry

#setting the tunegrid

#fitting the ranndom forest

<<Feature importance by random forest

#Variable importance

Take away:

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

<<Partial dependence plots

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

Let us calculate partial dependence plots on random forest

Let us plot the learned dtr model

<<Let us calculate weights and show important features using eli5 library.

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

Let us see important features,
%%time
#Important features
eli5.show_weights(perm_imp,feature_names=X_valid.columns.tolist(),top=200)

output:
```

Weight	Feature
0.0004 ± 0.0002	var_81
0.0003 ± 0.0002	var_146
0.0003 ± 0.0002	var_109
0.0003 ± 0.0002	var_12
0.0002 ± 0.0001	var_110
0.0002 ± 0.0000	var_173
0.0002 ± 0.0001	var_174
0.0002 ± 0.0002	var_0
0.0002 ± 0.0002	var_26
0.0001 ± 0.0001	var_166
0.0001 ± 0.0001	var_169
0.0001 ± 0.0001	var_22
0.0001 ± 0.0001	var_99
0.0001 ± 0.0001	var_53

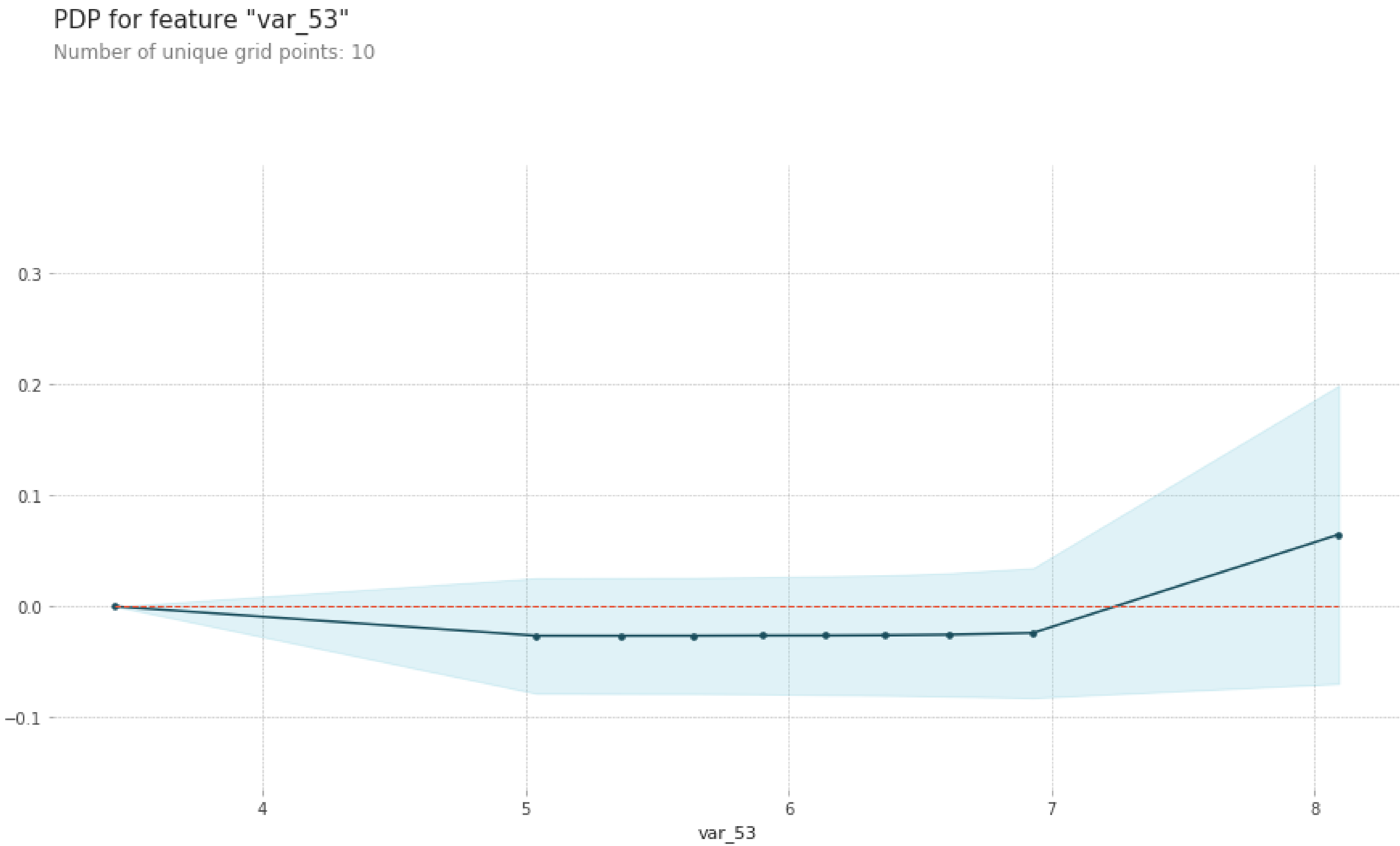
Weight	Feature
0.0001 ± 0.0001	var_8
0.0001 ± 0.0001	var_1
0.0001 ± 0.0000	var_37
0.0001 ± 0.0003	var_133
0.0001 ± 0.0000	var_152
0.0001 ± 0.0001	var_175
0.0001 ± 0.0001	var_88
0.0001 ± 0.0001	var_66
0.0001 ± 0.0001	var_184
0.0001 ± 0.0000	var_95
0.0001 ± 0.0001	var_176
0.0001 ± 0.0001	var_74
0.0001 ± 0.0001	

<<Partial dependence plot

Let us see impact of the main features which are discovered in the previous section by using pdp package.

#We will plot "var_53"

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


PDP for feature "var_53"
Number of unique grid points: 10
0.3
0.2
0.1
0.0
−0.1
4
5
6
7
8
var_53

*Take away:

The y_axis does not show the predictor value instead how the value changing with the change in given predictor variable.

The blue shaded area indicates the level of confidence of 'var_53'

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

<<Handling of imbalanced data

Now we are going to explore 5 different approaches for dealing with imbalanced datasets.

- Change the performance metric
- Oversample minority class
- Undersample majority class
- Synthetic Minority Oversampling Technique(SMOTE)
- Change the algorithm

Now let us start with simple Logistic regression model.

Logistic Regression model: In statistics, the logistic model is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc..

```
%%time
#Logistic regression model
lr_model=LogisticRegression(random_state=42)
#fitting the lr model
lr_model.fit(X_train,y_train)
```

```
Accuracy of model: 0.9145755339029131
```

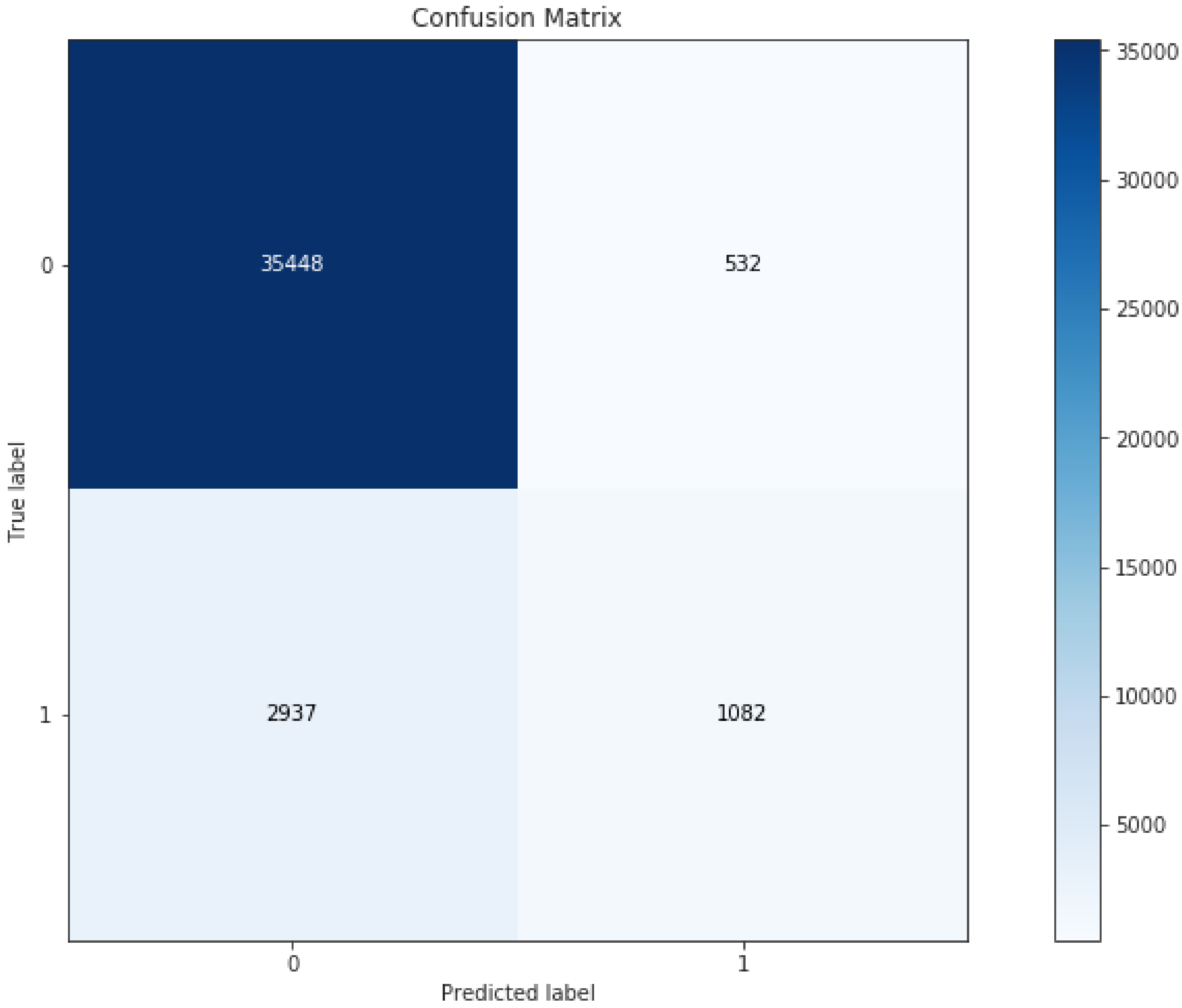
Cross validation prediction of lr_model

```
%%time
#Cross validation prediction
cv_predict=cross_val_predict(lr_model,X_valid,y_valid,cv=5)
#Cross validation score
cv_score=cross_val_score(lr_model,X_valid,y_valid,cv=5)
print('cross_val_score :',np.average(cv_score))
```

```
cross_val_score : 0.9132728216027003
```

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

Confusion matrix: n the field of machine learning and specifically the problem of statistical classification, a confusion matrix, also known as an error matrix, is a speci fic table layout that allows visualization of the performance of an algorithm, typically a supervised learning one.

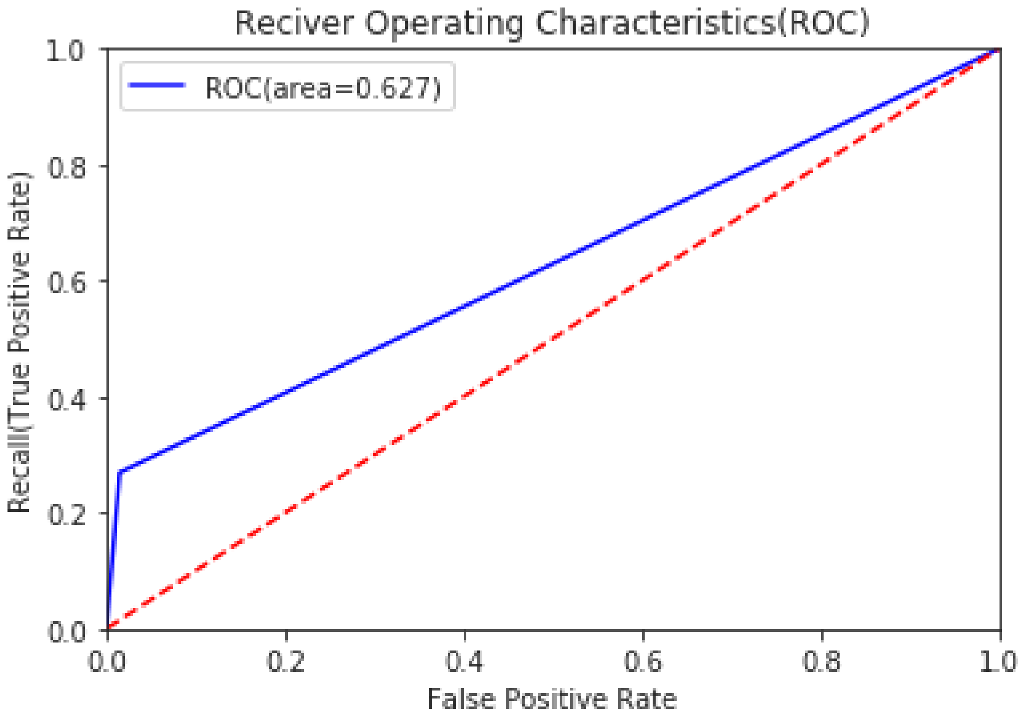


Confusion Matrix
35448
532
2937
1082
True label
Predicted label
0
1
0
1
35000
30000
25000
20000
15000
10000
5000

Reciever operating characteristics (ROC)-Area under curve(AUC) score and curve: In a **ROC curve** the true positive rate (Sensitivity) is plotted in function of the false positive rate (100-Specificity) for different cut-off points of a parameter. ... The **area under** the **ROC curve** (**AUC**) is a measure of how well a parameter can distinguish between two diagnostic groups (diseased/normal).

```
#ROC_AUC score
roc_score=roc_auc_score(y_valid,cv_predict)
print('ROC score :',roc_score)

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



AUC: 0.6272176035427053

When we compare the roc_auc_score and model accuracy , model is not performing well on imbalanced data.

Classification report

	precision	recall	f1-score	support
0	0.92	0.99	0.95	35980
1	0.67	0.27	0.38	4019

micro avg	0.91	0.91	0.91	39999
macro avg	0.80	0.63	0.67	39999
weighted avg	0.90	0.91	0.90	39999

We can observed that f1 score is high for number of customers those who will not make a transaction then the who will make a transaction. So, we are going to change the algorithm.

Model performance on test data: [0 0 0 ... 0 0 0]

Oversample minority class:

- It can be defined as adding more copies of minority class.
- It can be a good choice when we don't have a ton of data to work with.
- Drawback is that we are adding information.This may leads to overfitting and poor performance on test data.

Undersample majority class:

- It can be defined as removing some observations of the majority class.
- It can be a good choice when we have a ton of data -think million of rows.
- Drawback is that we are removing information that may be valuable.This may leads to underfitting and poor performance on test data.

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

Synthetic Minority Oversampling Technique(SMOTE)

SMOTE uses a nearest neighbors algorithm to generate new and synthetic data to used for training the model.

```
%%time
```

```
from imblearn.over_sampling import SMOTE
#Synthetic Minority Oversampling Technique
sm = SMOTE(random_state=42, ratio=1.0)
#Generating synthetic data points
X_smote,y_smote=sm.fit_sample(X_train,y_train)
X_smote_v,y_smote_v=sm.fit_sample(X_valid,y_valid)
```

<<Let us see how baseline logistic regression model performs on synthetic data points.

```
%%time
#Logistic regression model for SMOTE
smote=LogisticRegression(random_state=42)
#fitting the smote model
smote.fit(X_smote,y_smote)
```

Accuracy of model

```
smote_score=smote.score(X_smote,y_smote)
print('Accuracy of the smote_model :',smote_score)
```

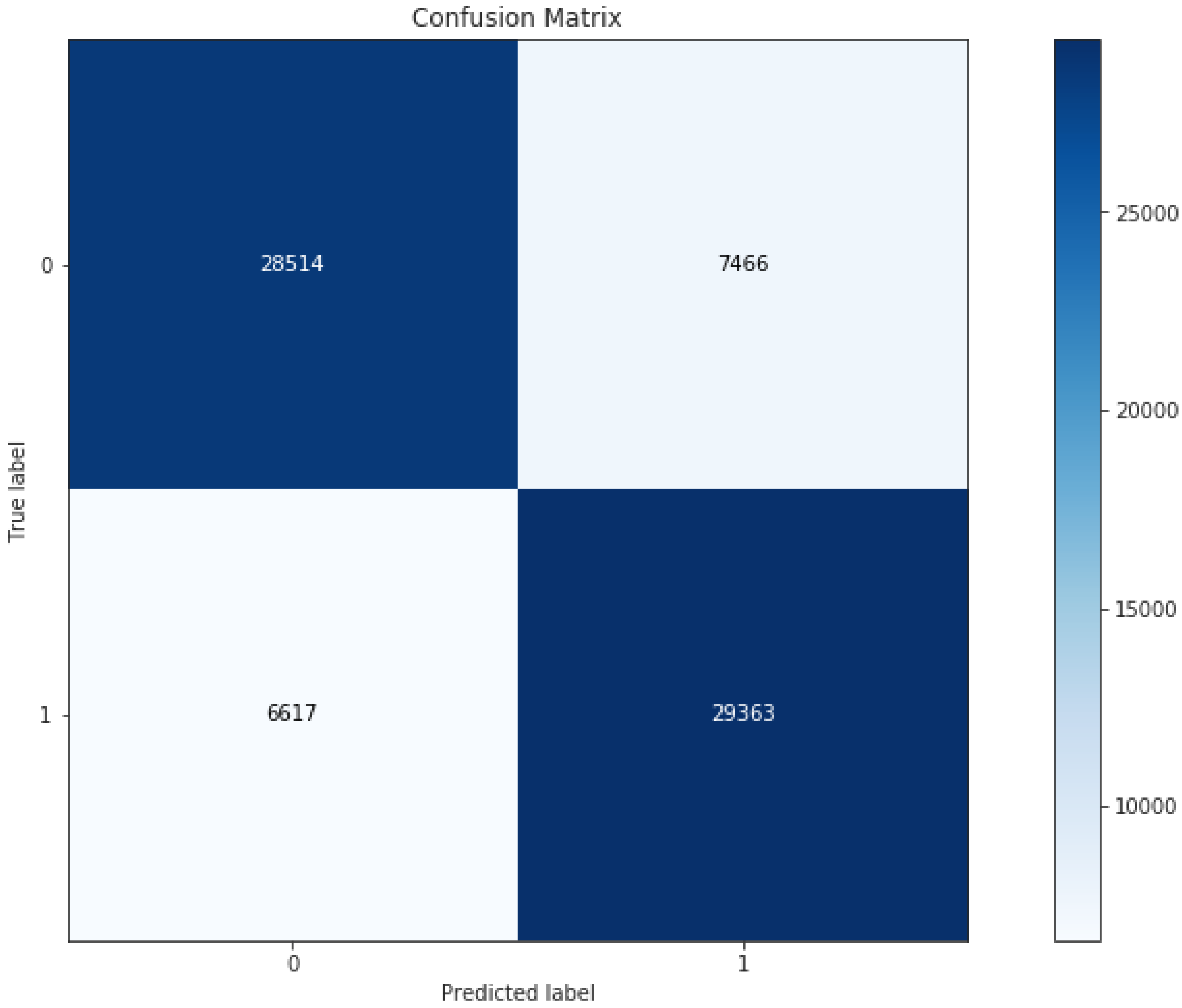
Accuracy of the smote_model : 0.7984012173260516

Cross validation prediction of smoth_model

```
%%time
#Cross validation prediction
cv_pred=cross_val_predict(smote,X_smote_v,y_smote_v,cv=5)
#Cross validation score
cv_score=cross_val_score(smote,X_smote_v,y_smote_v,cv=5)
print('cross_val_score :',np.average(cv_score))
```

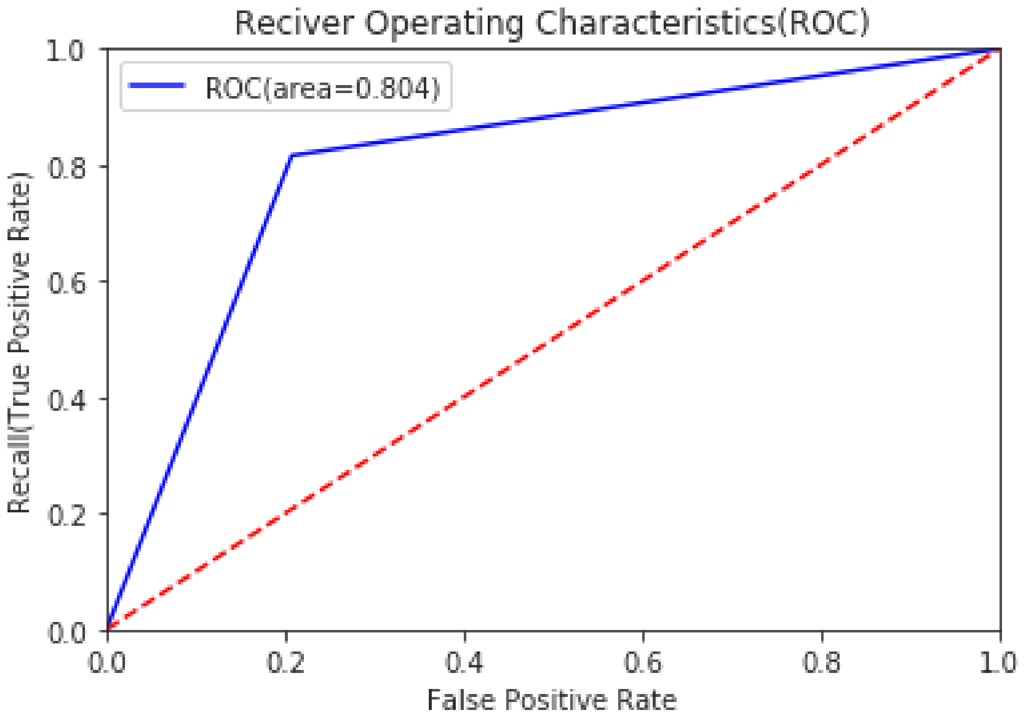
cross_val_score : 0.8042940522512507

Confusion matrix



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

ROC score : 0.8042940522512506



AUC: 0.8042940522512506

Classification report

	precision	recall	f1-score	support
0	0.81	0.79	0.80	35980
1	0.80	0.82	0.81	35980
micro avg	0.80	0.80	0.80	71960
macro avg	0.80	0.80	0.80	71960

weighted avg 0.80 0.80 0.80 71960

Model performance on test data: [1 1 0 ... 0 0 1]

We can observed that smote model is performing well on imbalance data compare to logistic regression.

<<LightGBM:

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

Let us build LightGBM model

```
#Training the model
#training data
lgb_train=lgb.Dataset(X_train,label=y_train)
#validation data
lgb_valid=lgb.Dataset(X_valid,label=y_valid)
```

choosing of hyperparameters

```
#Selecting best hyperparameters by tuning of different parameters
params={'boosting_type': 'gbdt',
          'max_depth' : -1, #no limit for max_depth if <0
          'objective': 'binary',
          'boost_from_average':False,
          'nthread': 20,
          'metric':'auc',
          'num_leaves': 50,
          'learning_rate': 0.01,
          'max_bin': 100,      #default 255
          'subsample_for_bin': 100,
          'subsample': 1,
          'subsample_freq': 1,
          'colsample_bytree': 0.8,
          'bagging_fraction':0.5,
          'bagging_freq':5,
          'feature_fraction':0.08,
```

```
          'min_split_gain': 0.45, #>0
          'min_child_weight': 1,
          'min_child_samples': 5,
          'is_unbalance':True,
          }
```

Training the lgbm model

```
num_rounds=10000
lgbm= lgb.train(params,lgb_train,num_rounds,valid_sets=[lgb_train,lgb_valid],verbose_eval=1000,early_stopping_rounds = 5000)
lgbm
```

```
Training until validation scores don't improve for 5000 rounds.
[1000]	training's auc: 0.939079	valid_1's auc: 0.882655
[2000]	training's auc: 0.958502	valid_1's auc: 0.887842
[3000]	training's auc: 0.971937	valid_1's auc: 0.889724
[4000]	training's auc: 0.981492	valid_1's auc: 0.890474
[5000]	training's auc: 0.988242	valid_1's auc: 0.890772
[6000]	training's auc: 0.992813	valid_1's auc: 0.890549
[7000]	training's auc: 0.995775	valid_1's auc: 0.890488
[8000]	training's auc: 0.997627	valid_1's auc: 0.890549
[9000]	training's auc: 0.998739	valid_1's auc: 0.890309
[10000]	training's auc: 0.999359	valid_1's auc: 0.889882
Did not meet early stopping. Best iteration is:
[10000]	training's auc: 0.999359	valid_1's auc: 0.889882
<lightgbm.basic.Booster at 0x7f41d061a898>
```

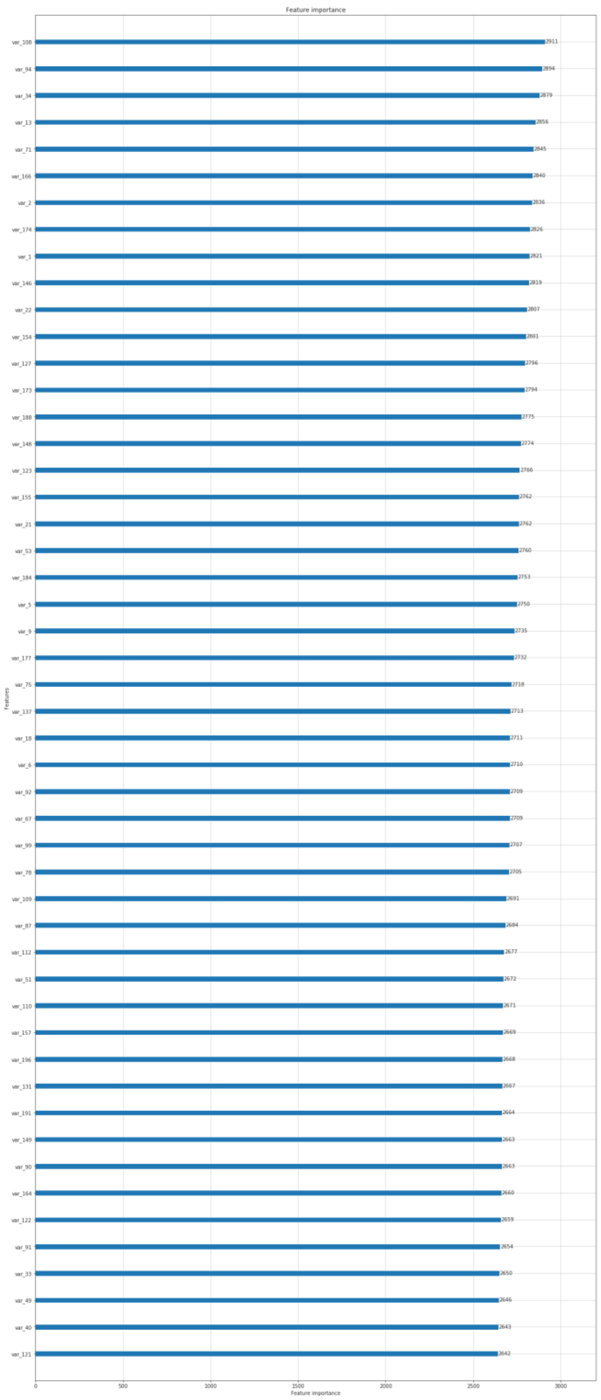
lgbm model performance on test data

```
X_test=test_df.drop(['ID_code'],axis=1)
#predict the model
#probability predictions
lgbm_predict_prob=lgbm.predict(X_test,random_state=42,num_iteration=lgbm.best_iteration)
#Convert to binary output 1 or 0
lgbm_predict=np.where(lgbm_predict_prob>=0.5,1,0)
print(lgbm_predict_prob)
print(lgbm_predict)
```

```
[0.38391682 0.41261082 0.33964579 ... 0.0058249  0.13459858 0.26750038]
```

```
[0 0 0 ... 0 0 0]
```

Let us plot the important features


Feature importance
var_108 2911
var_94 2894
var_34 2879
var_13 2856
var_71 2845
var_166 2840
var_2 2836
var_174 2826
var_1 2821
var_146 2819
var_22 2807
var_154 2801
var_127 2796
var_173 2794
var_188 2775
var_148 2774
var_123 2766
var_155 2762
var_21 2762
var_53 2760
var_184 2753
var_5 2750
var_9 2735
var_177 2732
var_75 2718
var_137 2713
var_18 2711
var_6 2710
var_92 2709
var_67 2709
var_99 2707
var_78 2705
var_109 2691
var_87 2684
var_112 2677
var_51 2672
var_110 2671
var_157 2669
var_196 2668
var_131 2667
var_191 2664
var_149 2663
var_90 2663
var_164 2660
var_122 2659
var_91 2654
var_33 2650
var_49 2646
var_40 2643
var_121 2642
Features
0
500
1000
1500
2000
2500
3000
Feature importance

Conclusion :

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

Final submission

ID_code	lgbm_predict_prob	lgbm_predict	
0	test_0	0.383917	0
1	test_1	0.412611	0
2	test_2	0.339646	0
3	test_3	0.255199	0
4	test_4	0.150399	0