edWisor
Get Skilled.Get Hired

Learn job skills | Work on industry projects | Get guaranteed interviews

<u>Santander Customer Transaction Prediction</u>

Submitted by: Devendra Singh

Jan, 26th ‘ 2020

Table of Contents

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 should help this **bank** identify which **customers** will make a **specific transaction** in the future, irrespective of the amount of money transacted. The data provided for this competition has the same structure as the real data we have available to solve this **problem**.

Data Details -

Both Train and Test dataset have same number of samples 200000, but both are having different number features. Because train dataset has target feature.

1. **Train Set:**

   - ID_code: Identifier for the transaction or customer
   - Target: Label feature for training
   - 202 are training features for model like var_1, var_2,.... , var_199

2. **Test Set:**

   - ID_code: Identifier for the transaction or customer
   - 202: are testing features same as training features for model like var_1, var_2,.... , var_199

3. Dataset shape

4. Dataset is imbalanced as class 0 has approx. 175000 sample and class 1 has near about 25000 sample.

Aim -

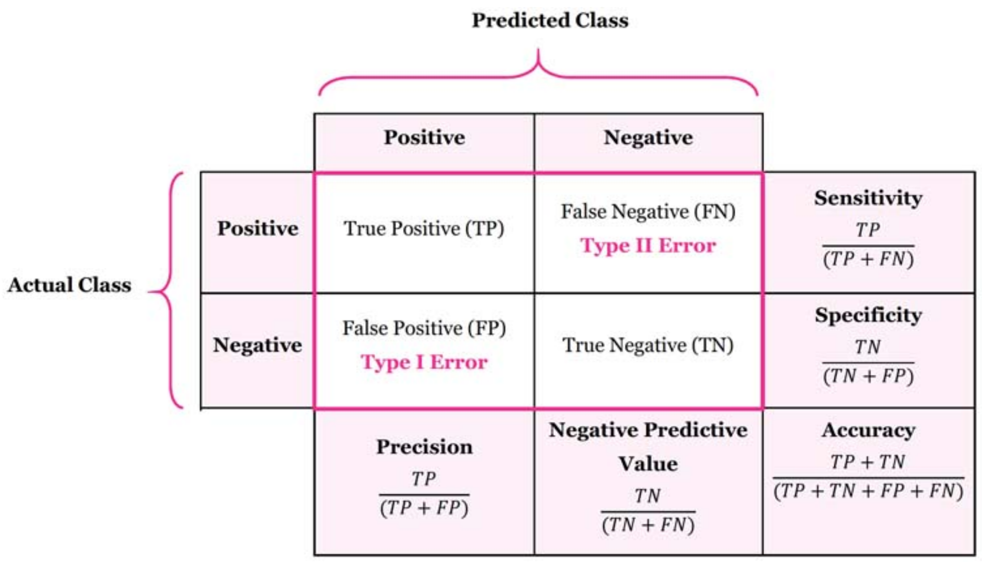
We are provided with an **anonymized dataset containing numeric feature variables**, the binary **target** column, and a string **ID_code** column.

The task is to predict the value of **target column** in the test set.

Evaluation Metrics -

This is a classification problem and we need to understand **confusion matrix** for getting evaluation metrics. It is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.
It is extremely useful for measuring **Recall**, **Precision**, **Accuracy** and most importantly **AUC-ROC** Curve.

- **True Positive:** You predicted positive and it's true.
- **True Negative:** You predicted negative and it's true.
- **False Positive:** (Type 1 Error) You predicted positive and it's false.
- **False Negative:** (Type 2 Error) You predicted negative and it's false.



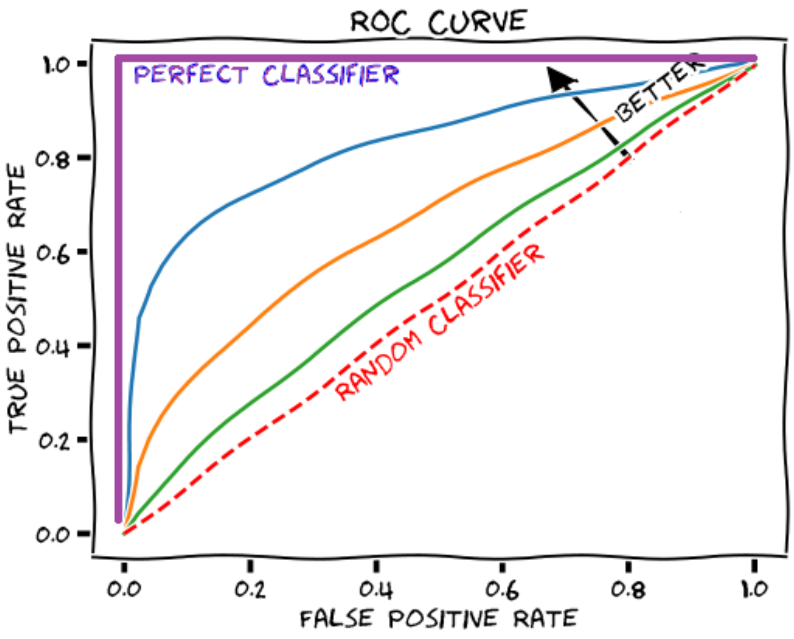
		Predicted Class		
		Positive	**Negative**	
Actual Class	**Positive**	True Positive (TP)	False Negative (FN) **Type II Error**	**Sensitivity** $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) **Type I Error**	True Negative (TN)	**Specificity** $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	**Negative Predictive Value** $\frac{TN}{(TN+FN)}$	**Accuracy** $\frac{TP+TN}{(TP+TN+FP+FN)}$

Based on confusion matrix we have following evaluation metrics-

- **Recall**- Out of all the positive classes, how much we predicted correctly. It should be high as possible.
- **Precision**- Out of all the classes, how much we predicted correctly. It should be high as possible.
- **F-measure**- It is difficult to compare two models with low precision and high recall or vice versa. So to make them comparable, we use F-Score. F-score helps to measure Recall and Precision at the same time. It uses Harmonic Mean in place of Arithmetic Mean by punishing the extreme values more.

ROC-AUC Curve

It is one of the most important evaluation metrics for checking any classification model's performance. It is also written as **AUROC** (Area Under the Receiver Operating Characteristics)
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, the better the model is at predicting 0s as 0s and 1s as 1s.



The x-axis of a ROC curve is the **false positive rate**, and the y-axis of a ROC curve is the **true positive rate**.

The AUROC is calculated as the **area under the ROC curve**. A ROC curve shows the trade-off between true positive rate (TPR) and false positive rate (FPR) across different decision thresholds.

The worst **AUROC** is 0.5, and the best AUROC is 1.0.

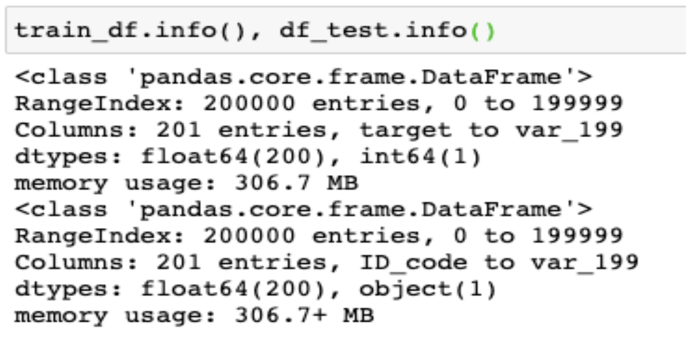
- **0.5** (area under the red dashed line in the figure above) corresponds to a **coin flip**, i.e. a useless model.
- less than **0.7** is **sub-optimal performance**
- **0.70 – 0.80** is **good performance**
- **greater than 0.8** is **excellent performance**
- 1.0 (area under the purple line in the figure above) corresponds to a **perfect classifier**

Exploratory Data Analysis (EDA)

In this section, we'll analysis how to use graphical and numerical techniques to begin uncovering the structure of your data.

- Data Collection
- Visualization
- Data Pre-processing
- Data Cleaning

1. Information about train and test data

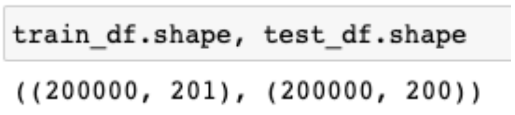


```
train_df.info(), df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Columns: 201 entries, target to var_199
dtypes: float64(200), int64(1)
memory usage: 306.7 MB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Columns: 201 entries, ID_code to var_199
dtypes: float64(200), object(1)
memory usage: 306.7+ MB
```

2. Shape of train and test data

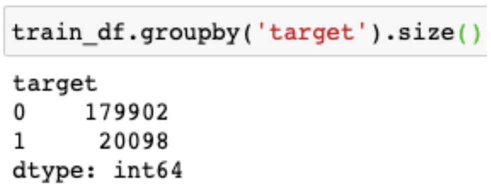
(200000,201) and (200000,200)



```
train_df.shape, test_df.shape
```

```
((200000, 201), (200000, 200))
```

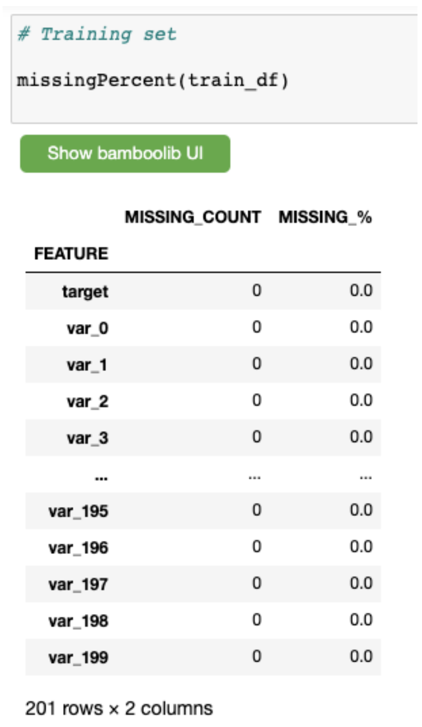
3. Distribution of target value in train data



```
train_df.groupby('target').size()
```

```
target
0    179902
1     20098
dtype: int64
```

4. Missing Values



```
# Training set

missingPercent(train_df)
```

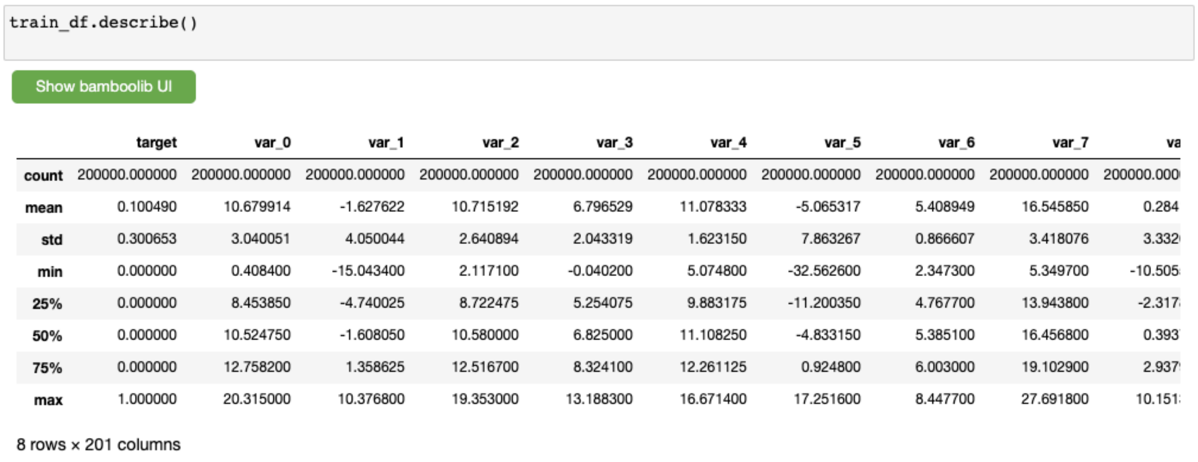
Show bamboolib UI

FEATURE	MISSING_COUNT	MISSING_%
target	0	0.0
var_0	0	0.0
var_1	0	0.0
var_2	0	0.0
var_3	0	0.0
...	...	...
var_195	0	0.0
var_196	0	0.0
var_197	0	0.0
var_198	0	0.0
var_199	0	0.0

201 rows × 2 columns

There are no missing values in train and test data both.

5. Description of train data



```
train_df.describe()
```

Show bamboolib UI

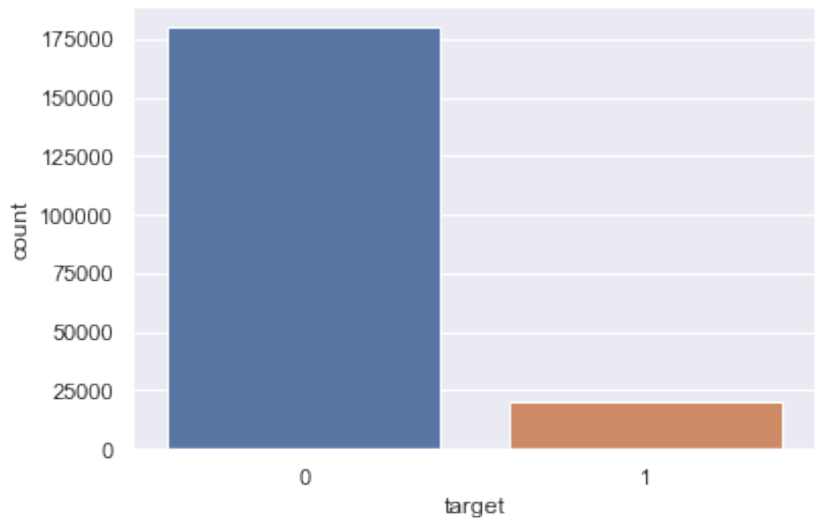
	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	va
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000
mean	0.100490	10.679914	-1.627622	10.715192	6.796529	11.078333	-5.065317	5.408949	16.545850	0.284
std	0.300653	3.040051	4.050044	2.640894	2.043319	1.623150	7.863267	0.866607	3.418076	3.332
min	0.000000	0.408400	-15.043400	2.117100	-0.040200	5.074800	-32.562600	2.347300	5.349700	-10.505
25%	0.000000	8.453850	-4.740025	8.722475	5.254075	9.883175	-11.200350	4.767700	13.943800	-2.317
50%	0.000000	10.524750	-1.608050	10.580000	6.825000	11.108250	-4.833150	5.385100	16.456800	0.393
75%	0.000000	12.758200	1.358625	12.516700	8.324100	12.261125	0.924800	6.003000	19.102900	2.937
max	1.000000	20.315000	10.376800	19.353000	13.188300	16.671400	17.251600	8.447700	27.691800	10.151

8 rows × 201 columns

Describe function shows statistics of data.

2. Visualization

a) Distribution of Target Values in train set

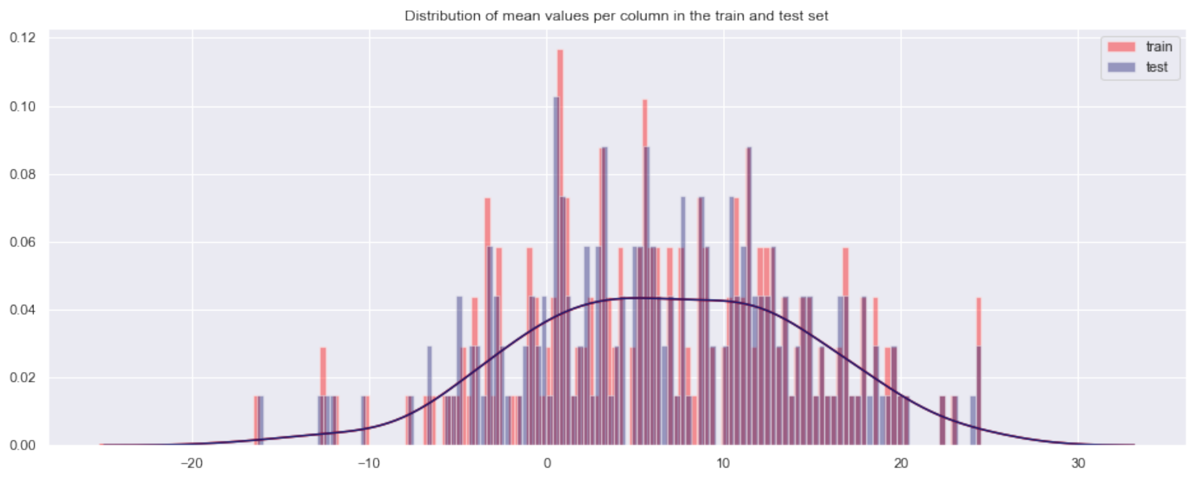


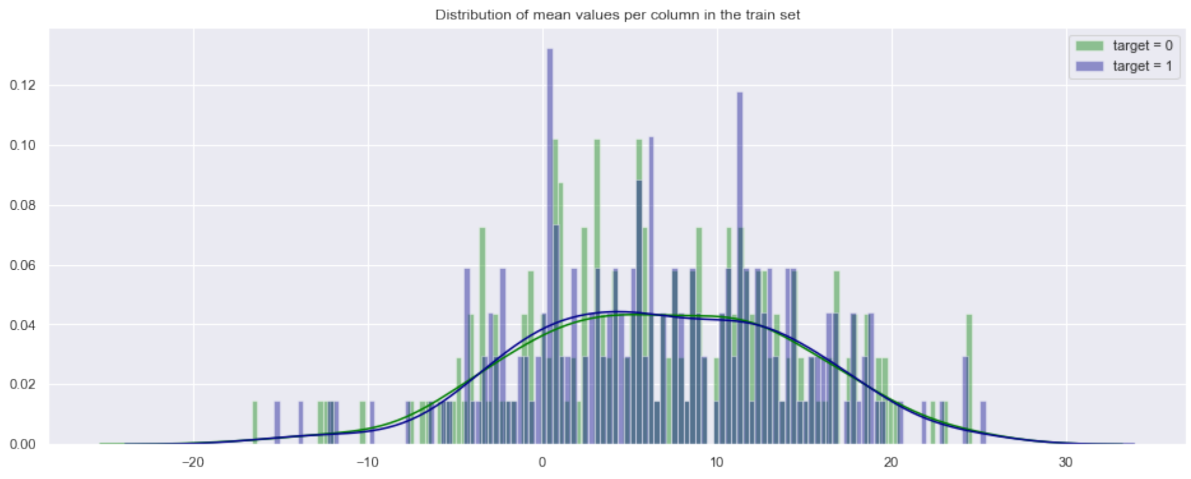
There are **10.049%** target values with 1and data is unbalanced with respect with **target** value.

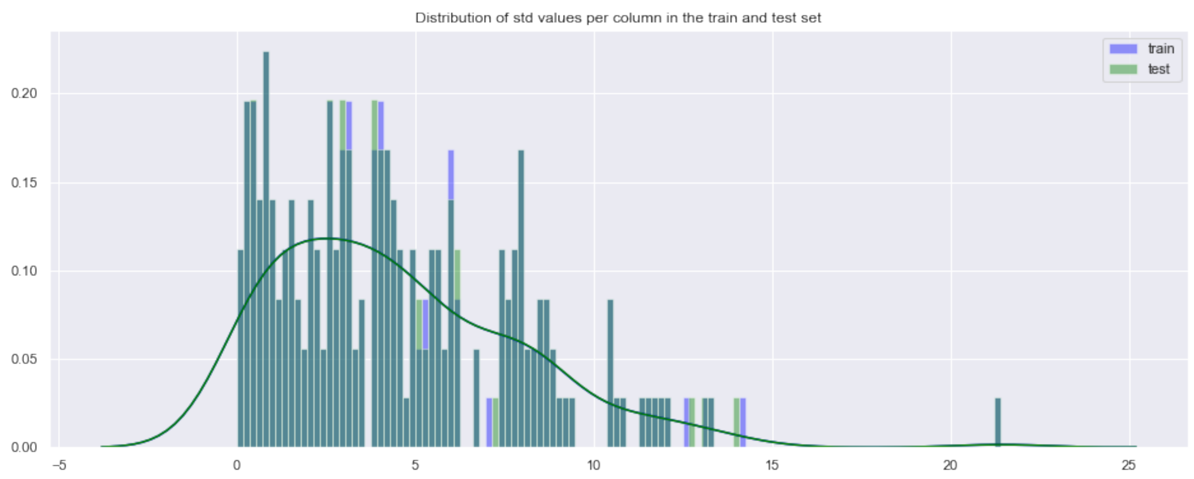
b)

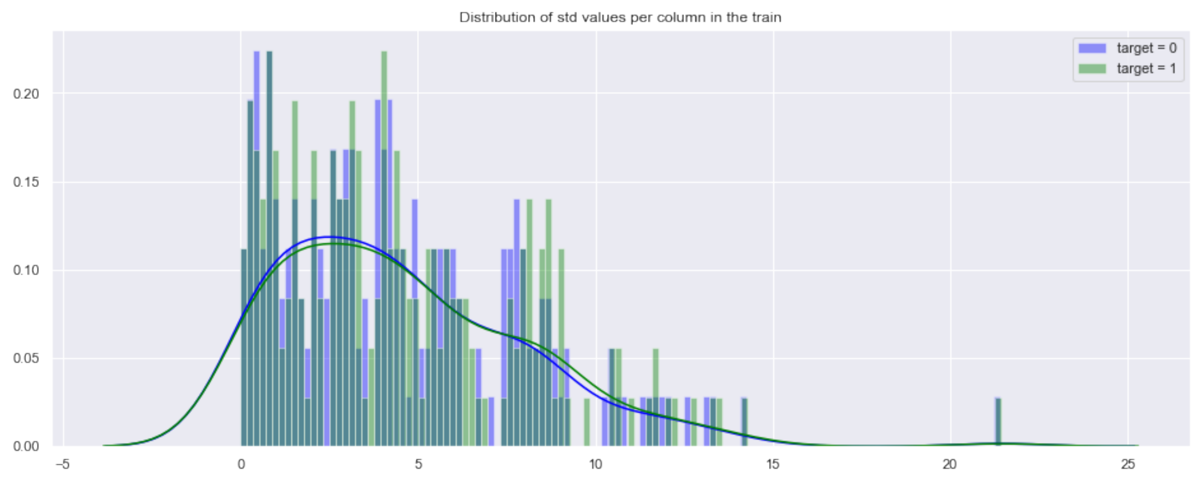
i) The distribution of the mean, Standard Deviation and skewness values per column in the train and test set.

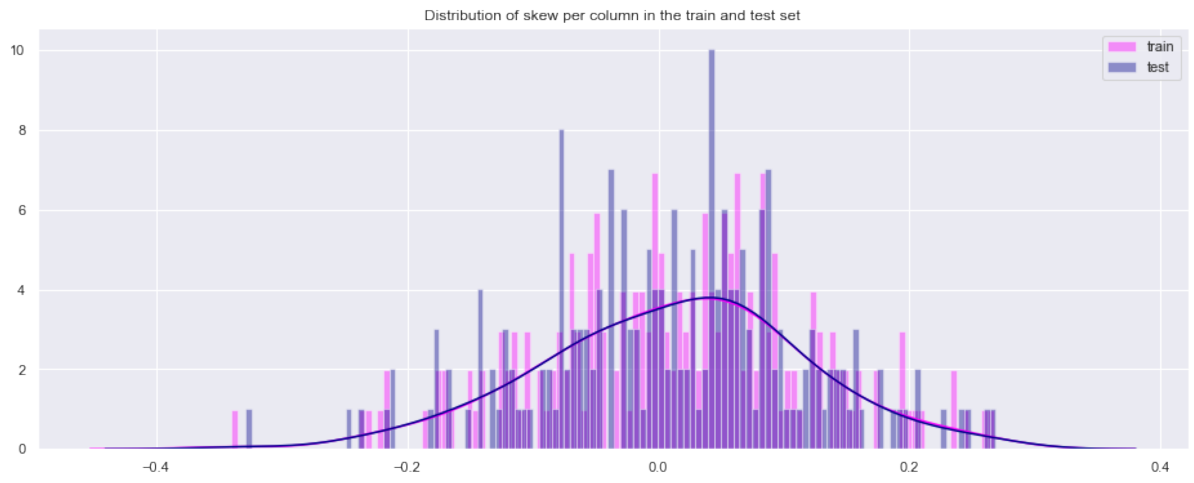
ii) The distribution of the mean, Standard Deviation and skewness values per column in the train set with target values 0 and 1.

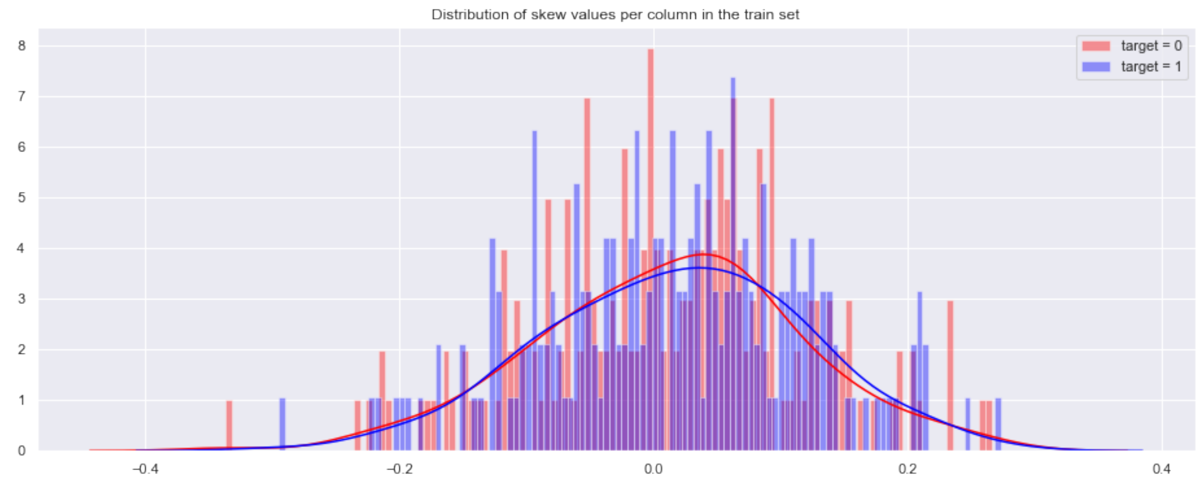




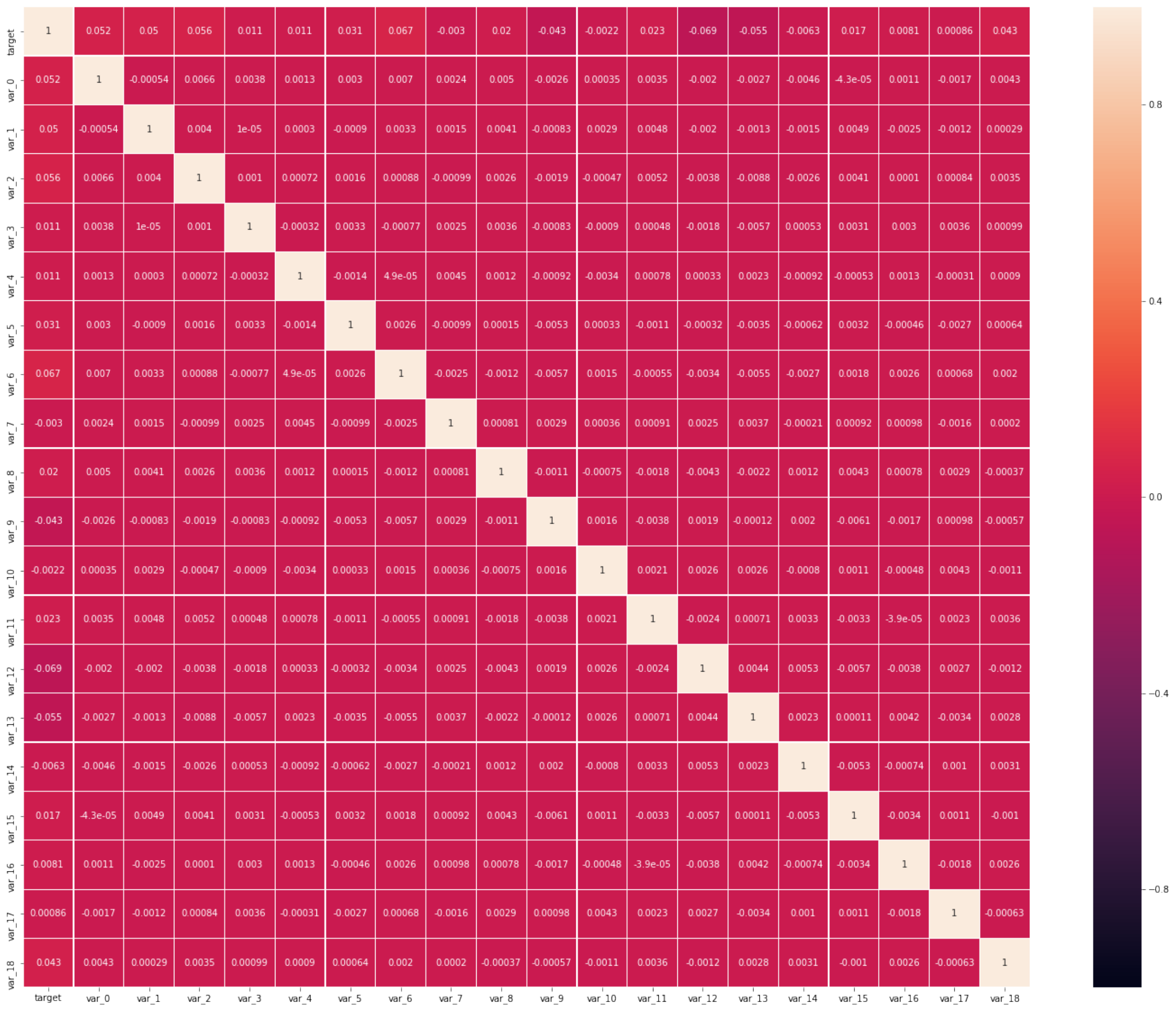

Distribution of std values per column in the train and test set
train
test
0.20
0.15
0.10
0.05
0.00
−5
0
5
10
15
20
25


Distribution of std values per column in the train
target = 0
target = 1
0.20
0.15
0.10
0.05
0.00
−5
0
5
10
15
20
25


Distribution of skew per column in the train and test set
train
test
10
8
6
4
2
0
−0.4
−0.2
0.0
0.2
0.4


Distribution of skew values per column in the train set
target = 0
target = 1
8
7
6
5
4
3
2
1
0
−0.4
−0.2
0.0
0.2
0.4

c) Heat Map showing correlation of independent variable of first 15 variables.



	target	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	var_10	var_11	var_12	var_13	var_14	var_15	var_16	var_17	var_18
target	1	0.052	0.05	0.056	0.011	0.011	0.031	0.067	-0.003	0.02	-0.043	-0.0022	0.023	-0.069	-0.055	-0.0063	0.017	0.0081	0.00086	0.043
var_0	0.052	1	-0.00054	0.0066	0.0038	0.0013	0.003	0.007	0.0024	0.005	-0.0026	0.00035	0.0035	-0.002	-0.0027	-0.0046	-4.3e-05	0.0011	-0.0017	0.0043
var_1	0.05	-0.00054	1	0.004	1e-05	0.0003	-0.0009	0.0033	0.0015	0.0041	-0.00083	0.0029	0.0048	-0.002	-0.0013	-0.0015	0.0049	-0.0025	-0.0012	0.00029
var_2	0.056	0.0066	0.004	1	0.001	0.00072	0.0016	0.00088	-0.00099	0.0026	-0.0019	-0.00047	0.0052	-0.0038	-0.0088	-0.0026	0.0041	0.0001	0.00084	0.0035
var_3	0.011	0.0038	1e-05	0.001	1	-0.00032	0.0033	-0.00077	0.0025	0.0036	-0.00083	-0.0009	0.00048	-0.0018	-0.0057	0.00053	0.0031	0.003	0.0036	0.00099
var_4	0.011	0.0013	0.0003	0.00072	-0.00032	1	-0.0014	4.9e-05	0.0045	0.0012	-0.00092	-0.0034	0.00078	0.00033	0.0023	-0.00092	-0.00053	0.0013	-0.00031	0.0009
var_5	0.031	0.003	-0.0009	0.0016	0.0033	-0.0014	1	0.0026	-0.00099	0.00015	-0.0053	0.00033	-0.0011	-0.00032	-0.0035	-0.00062	0.0032	-0.00046	-0.0027	0.00064
var_6	0.067	0.007	0.0033	0.00088	-0.00077	4.9e-05	0.0026	1	-0.0025	-0.0012	-0.0057	0.0015	-0.00055	-0.0034	-0.0055	-0.0027	0.0018	0.0026	0.00068	0.002
var_7	-0.003	0.0024	0.0015	-0.00099	0.0025	0.0045	-0.00099	-0.0025	1	0.00081	0.0029	0.00036	0.00091	0.0025	0.0037	-0.00021	0.00092	0.00098	-0.0016	0.0002
var_8	0.02	0.005	0.0041	0.0026	0.0036	0.0012	0.00015	-0.0012	0.00081	1	-0.0011	-0.00075	-0.0018	-0.0043	-0.0022	0.0012	0.0043	0.00078	0.0029	-0.00037
var_9	-0.043	-0.0026	-0.00083	-0.0019	-0.00083	-0.00092	-0.0053	-0.0057	0.0029	-0.0011	1	0.0016	-0.0038	0.0019	-0.00012	0.002	-0.0061	-0.0017	0.00098	-0.00057
var_10	-0.0022	0.00035	0.0029	-0.00047	-0.0009	-0.0034	0.00033	0.0015	0.00036	-0.00075	0.0016	1	0.0021	0.0026	0.0026	-0.0008	0.0011	-0.00048	0.0043	-0.0011
var_11	0.023	0.0035	0.0048	0.0052	0.00048	0.00078	-0.0011	-0.00055	0.00091	-0.0018	-0.0038	0.0021	1	-0.0024	0.00071	0.0033	-0.0033	-3.9e-05	0.0023	0.0036
var_12	-0.069	-0.002	-0.002	-0.0038	-0.0018	0.00033	-0.00032	-0.0034	0.0025	-0.0043	0.0019	0.0026	-0.0024	1	0.0044	0.0053	-0.0057	-0.0038	0.0027	-0.0012
var_13	-0.055	-0.0027	-0.0013	-0.0088	-0.0057	0.0023	-0.0035	-0.0055	0.0037	-0.0022	-0.00012	0.0026	0.00071	0.0044	1	0.0023	0.00011	0.0042	-0.0034	0.0028
var_14	-0.0063	-0.0046	-0.0015	-0.0026	0.00053	-0.00092	-0.00062	-0.0027	-0.00021	0.0012	0.002	-0.0008	0.0033	0.0053	0.0023	1	-0.0053	-0.00074	0.001	0.0031
var_15	0.017	-4.3e-05	0.0049	0.0041	0.0031	-0.00053	0.0032	0.0018	0.00092	0.0043	-0.0061	0.0011	-0.0033	-0.0057	0.00011	-0.0053	1	-0.0034	0.0011	-0.001
var_16	0.0081	0.0011	-0.0025	0.0001	0.003	0.0013	-0.00046	0.0026	0.00098	0.00078	-0.0017	-0.00048	-3.9e-05	-0.0038	0.0042	-0.00074	-0.0034	1	-0.0018	0.0026
var_17	0.00086	-0.0017	-0.0012	0.00084	0.0036	-0.00031	-0.0027	0.00068	-0.0016	0.0029	0.00098	0.0043	0.0023	0.0027	-0.0034	0.001	0.0011	-0.0018	1	-0.00063
var_18	0.043	0.0043	0.00029	0.0035	0.00099	0.0009	0.00064	0.002	0.0002	-0.00037	-0.00057	-0.0011	0.0036	-0.0012	0.0028	0.0031	-0.001	0.0026	-0.00063	1

We can make few observations here:

- standard deviation is relatively large for both train and test variable data
- mean, std values for train and test data looks quite close
- mean values are distributed over a large range

Dealing with these imbalanced datasets

Imbalanced classes are a common problem in machine learning classification where there are a disproportionate ratio of observations in each class. Class imbalance can be found in many different areas including medical diagnosis, spam filtering, and fraud detection.

1. Change the performance metric

Accuracy is not the best metric to use when evaluating imbalanced datasets as it can be misleading. Metrics that can provide better insight include:

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

Since our main objective with the dataset is to prioritize accurately classifying fraud cases the recall score can be considered our main metric to use for evaluating outcomes.

2. Change the algorithm

While in every machine learning problem, it’s a good rule of thumb to try a variety of algorithms, it can be especially **beneficial with imbalanced datasets**. Decision trees frequently perform well on imbalanced data. They work by learning a hierarchy of if/else questions. This can force both classes to be addressed.

Resampling Techniques

3. Oversampling Minority Class

Oversampling can be defined as **adding more copies of the minority class**. Oversampling can be a good choice when you don't have a ton of data to work with. A con to consider when under-sampling is that it can cause **overfitting and poor generalization** to your test set.
We will use the resampling module from Scikit-Learn to randomly replicate samples from the minority class.

Always split into test and train sets BEFORE trying any resampling techniques! Oversampling before splitting the data can allow the exact same observations to be present in both the test and train sets! This can allow our model to simply memorize specific data points and cause overfitting

4. Undersampling Majority Class

Undersampling can be defined as **removing some observations of the majority class**. Undersampling can be a good choice when you have a ton of data -think millions of rows. But a drawback to undersampling is that we are removing information that may be valuable.
We will again use the resampling module from Scikit-Learn to randomly remove samples from the majority class.

5. Boosting-Based Ensembling

Boosting is an ensemble technique to **combine weak learners to create a strong learner** that can make accurate predictions. Boosting starts out with a base classifier / weak classifier that is prepared on the training data.
The base learners / Classifiers are weak learners i.e. the prediction accuracy is only slightly **better than average**. A classifier learning algorithm is said to be weak when small changes in data induce big changes in the classification model.
In the next iteration, the new classifier **focuses on or places more weight** to those cases which were incorrectly classified in the last round.
I will use Recall, Precision, F1_score (harmonic mean of Precision and recall), AUC-ROC score along with Accuracy for model evaluation.

Algorithms

- **Logistic regression**

This is the classification problem. We can use logistic regression for this problem. Logistic Regression is used when the dependent variable(target) is categorical. It has a **bias** towards classes which have large number of instances. It tends to only **predict the majority class data**. The features of the minority class are treated as noise and are often ignored. Thus, there is a high probability of **misclassification** of the minority class as compared to the majority class.

- **Support Vector Machine**

“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both **classification** or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in **n-dimensional space** (where n is the number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the **hyper-plane** that differentiate the two classes very well. Support Vectors are simply the coordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line). It is effective in cases where the number of dimensions is greater than the number of samples. But it doesn’t perform well, when we have large data set because the required training time is higher.

- **Random Forest**

Random Forest is a bagging based **ensemble learning model**. Random forests is **slow** in generating predictions because it has **multiple decision trees**. Whenever it makes a prediction, all the trees in the forest have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming. Thus result (AUC-score) shown below for random Forest is not modified if num round would increase the score will be better but speed will be very slow. These results are shown for default values of Parameters.

- **KNN**

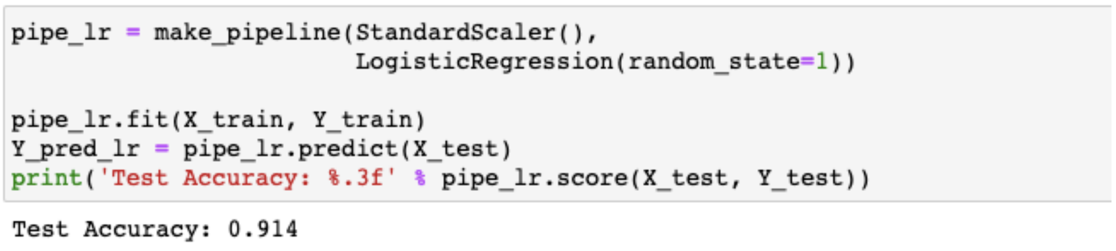
A k-nearest-neighbour algorithm, often abbreviated **k-nn**, is an approach to data **classification** that estimates how likely a data point is to be a member of one group or the other depending on what group the data points nearest to it are in.
The k-nearest-neighbour is an example of a **"lazy learner"** algorithm, meaning that it does not build a model using the training set until a query of the data set is performed.

- **LightGBM**

LightGBM is **Gradient Boosting ensemble model** which is faster in speed and accuracy as compared to bagging and adaptive boosting. It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST. But parameter tuning in LightGBM should be done carefully.

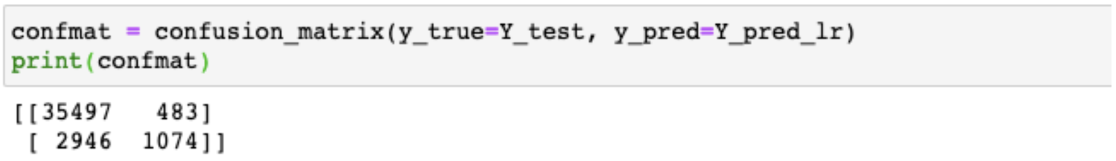
Results

1. Logistic Regression



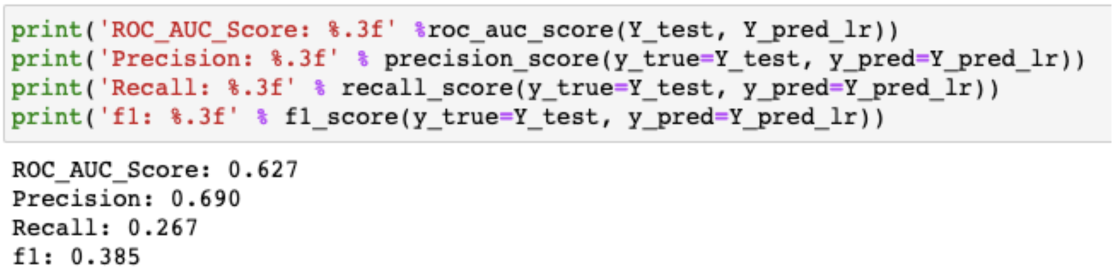
```
pipe_lr = make_pipeline(StandardScaler(),
                        LogisticRegression(random_state=1))

pipe_lr.fit(X_train, Y_train)
Y_pred_lr = pipe_lr.predict(X_test)
print('Test Accuracy: %.3f' % pipe_lr.score(X_test, Y_test))
```

```
Test Accuracy: 0.914
```



```
confmat = confusion_matrix(y_true=Y_test, y_pred=Y_pred_lr)
print(confmat)
```

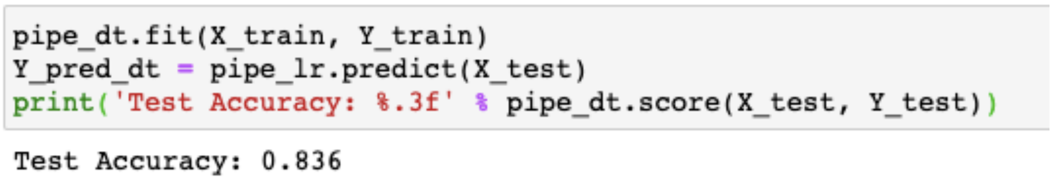
```
[[35497   483]
 [ 2946  1074]]
```



```
print('ROC_AUC_Score: %.3f' %roc_auc_score(Y_test, Y_pred_lr))
print('Precision: %.3f' % precision_score(y_true=Y_test, y_pred=Y_pred_lr))
print('Recall: %.3f' % recall_score(y_true=Y_test, y_pred=Y_pred_lr))
print('f1: %.3f' % f1_score(y_true=Y_test, y_pred=Y_pred_lr))
```

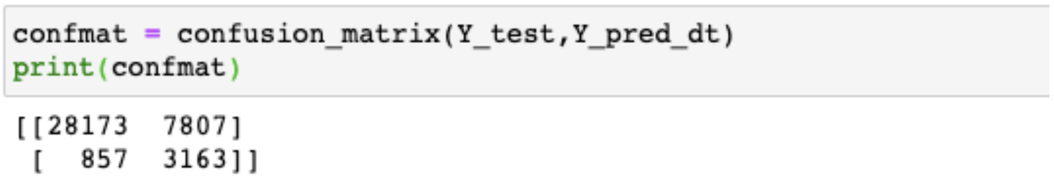
```
ROC_AUC_Score: 0.627
Precision: 0.690
Recall: 0.267
f1: 0.385
```

2. Decision Tree



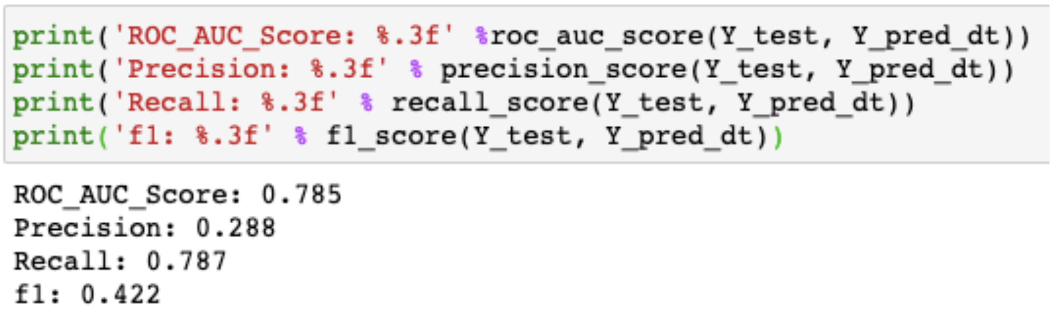
```
pipe_dt.fit(X_train, Y_train)
Y_pred_dt = pipe_lr.predict(X_test)
print('Test Accuracy: %.3f' % pipe_dt.score(X_test, Y_test))
```

```
Test Accuracy: 0.836
```



```
confmat = confusion_matrix(Y_test,Y_pred_dt)
print(confmat)
```

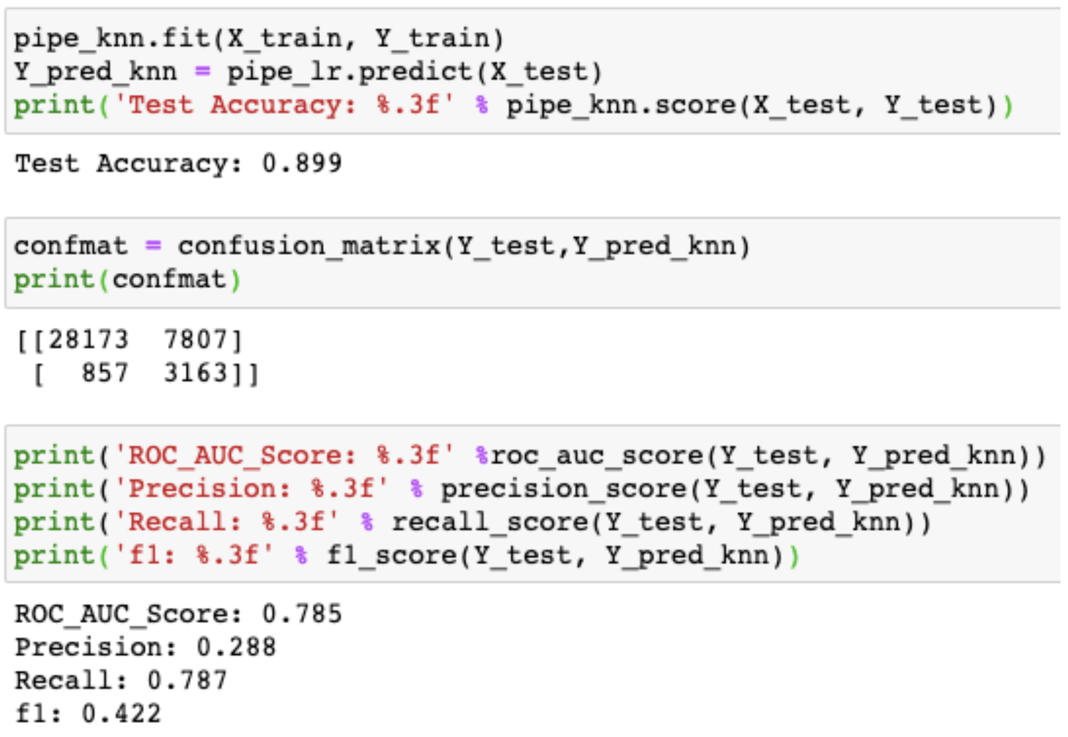
```
[[28173  7807]
 [  857  3163]]
```



```
print('ROC_AUC_Score: %.3f' %roc_auc_score(Y_test, Y_pred_dt))
print('Precision: %.3f' % precision_score(Y_test, Y_pred_dt))
print('Recall: %.3f' % recall_score(Y_test, Y_pred_dt))
print('f1: %.3f' % f1_score(Y_test, Y_pred_dt))
```

```
ROC_AUC_Score: 0.785
Precision: 0.288
Recall: 0.787
f1: 0.422
```

3. KNN



```
pipe_knn.fit(X_train, Y_train)
Y_pred_knn = pipe_lr.predict(X_test)
print('Test Accuracy: %.3f' % pipe_knn.score(X_test, Y_test))
```

```
Test Accuracy: 0.899
```

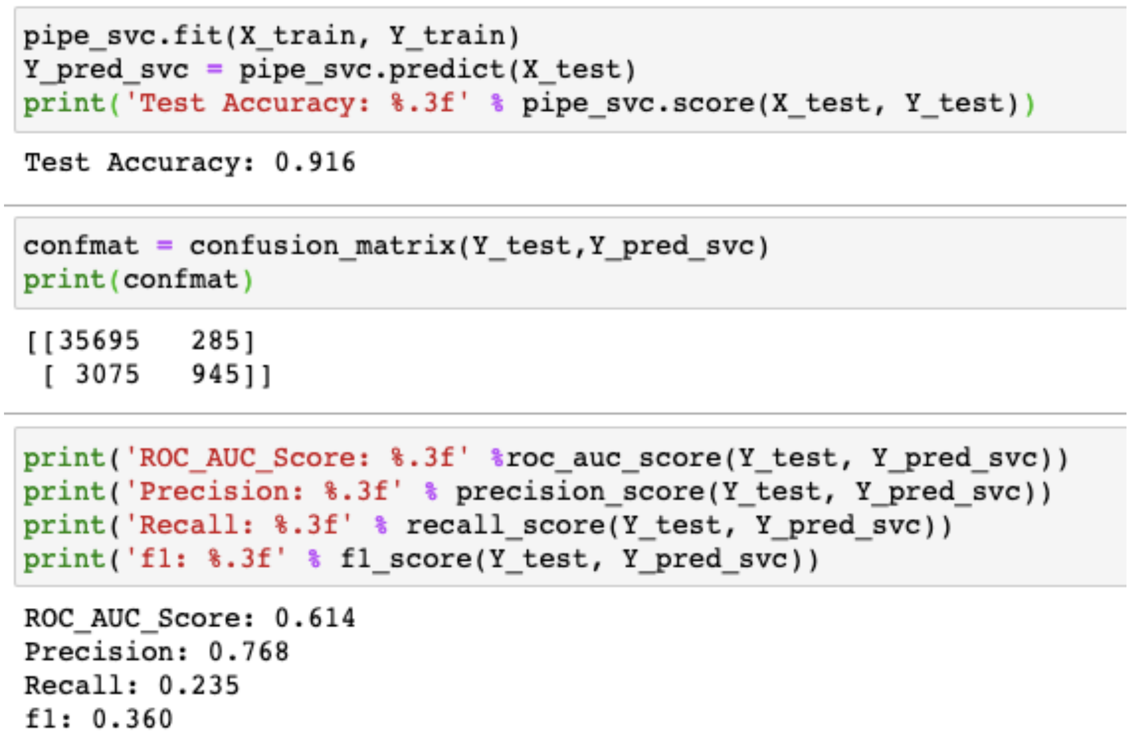
```
confmat = confusion_matrix(Y_test,Y_pred_knn)
print(confmat)
```

```
[[28173  7807]
 [  857  3163]]
```

```
print('ROC_AUC_Score: %.3f' %roc_auc_score(Y_test, Y_pred_knn))
print('Precision: %.3f' % precision_score(Y_test, Y_pred_knn))
print('Recall: %.3f' % recall_score(Y_test, Y_pred_knn))
print('f1: %.3f' % f1_score(Y_test, Y_pred_knn))
```

```
ROC_AUC_Score: 0.785
Precision: 0.288
Recall: 0.787
f1: 0.422
```

4. SVM



```
pipe_svc.fit(X_train, Y_train)
Y_pred_svc = pipe_svc.predict(X_test)
print('Test Accuracy: %.3f' % pipe_svc.score(X_test, Y_test))
```

```
Test Accuracy: 0.916
```

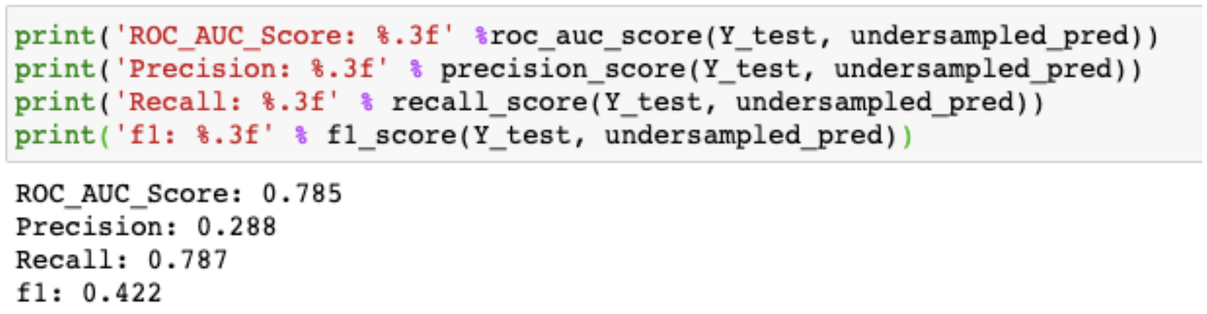
```
confmat = confusion_matrix(Y_test,Y_pred_svc)
print(confmat)
```

```
[[35695   285]
 [ 3075   945]]
```

```
print('ROC_AUC_Score: %.3f' %roc_auc_score(Y_test, Y_pred_svc))
print('Precision: %.3f' % precision_score(Y_test, Y_pred_svc))
print('Recall: %.3f' % recall_score(Y_test, Y_pred_svc))
print('f1: %.3f' % f1_score(Y_test, Y_pred_svc))
```

```
ROC_AUC_Score: 0.614
Precision: 0.768
Recall: 0.235
f1: 0.360
```

5. Under sampling and fit with Logistic Regression

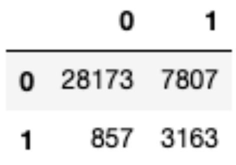


```
print('ROC_AUC_Score: %.3f' %roc_auc_score(Y_test, undersampled_pred))
print('Precision: %.3f' % precision_score(Y_test, undersampled_pred))
print('Recall: %.3f' % recall_score(Y_test, undersampled_pred))
print('f1: %.3f' % f1_score(Y_test, undersampled_pred))
```

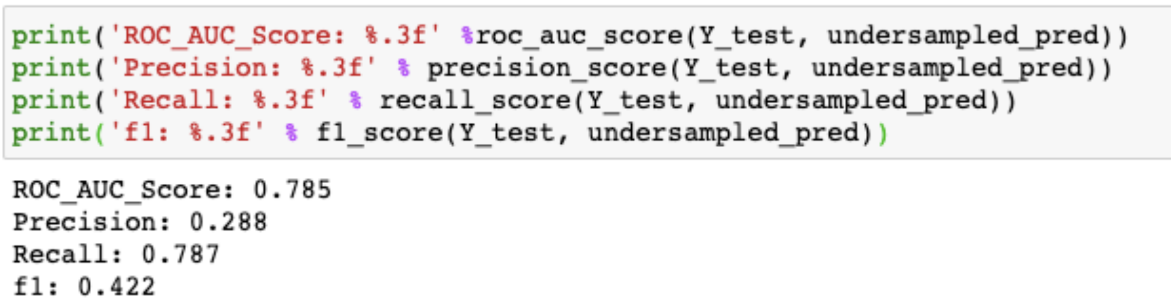
```
ROC_AUC_Score: 0.785
Precision: 0.288
Recall: 0.787
f1: 0.422
```

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. Visualizing 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 **Logistic Regression** for resampling of minority class as my final model then using the same model with feature engineered data we got **AUC-Score of 0.785**. Confusion matrix and evolution shown below:



	0	1
0	28173	7807
1	857	3163

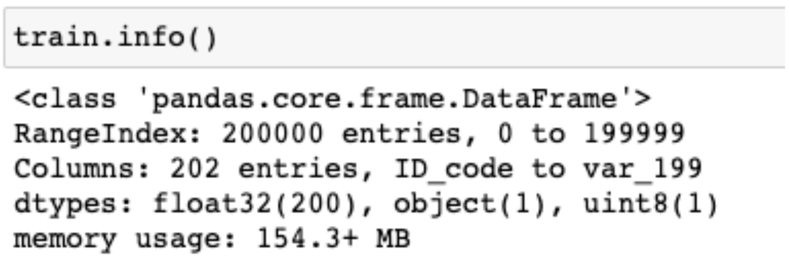


```
print('ROC_AUC_Score: %.3f' %roc_auc_score(Y_test, undersampled_pred))
print('Precision: %.3f' % precision_score(Y_test, undersampled_pred))
print('Recall: %.3f' % recall_score(Y_test, undersampled_pred))
print('f1: %.3f' % f1_score(Y_test, undersampled_pred))
```

```
ROC_AUC_Score: 0.785
Precision: 0.288
Recall: 0.787
f1: 0.422
```

Since dataset is very large (+300MB) it is very difficult to tune the parameter. It really takes a lot of time to execute even one algorithm but I done for logistic regression which shows improvement in f1 and AUC score.

Before performing data pre-processing and model fit & transform, I used method to reduce the memory of data size from *https://www.kaggle.com/arjanso/reducing-dataframe-memory-size-by-65*.
After this code my data now showing has 154+ MB memory which is quite comfortable while doing data pre-processing.



```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Columns: 202 entries, ID_code to var_199
dtypes: float32(200), object(1), uint8(1)
memory usage: 154.3+ MB
```

Thanks…..