CSE 258 Credit Card Fraud Detection

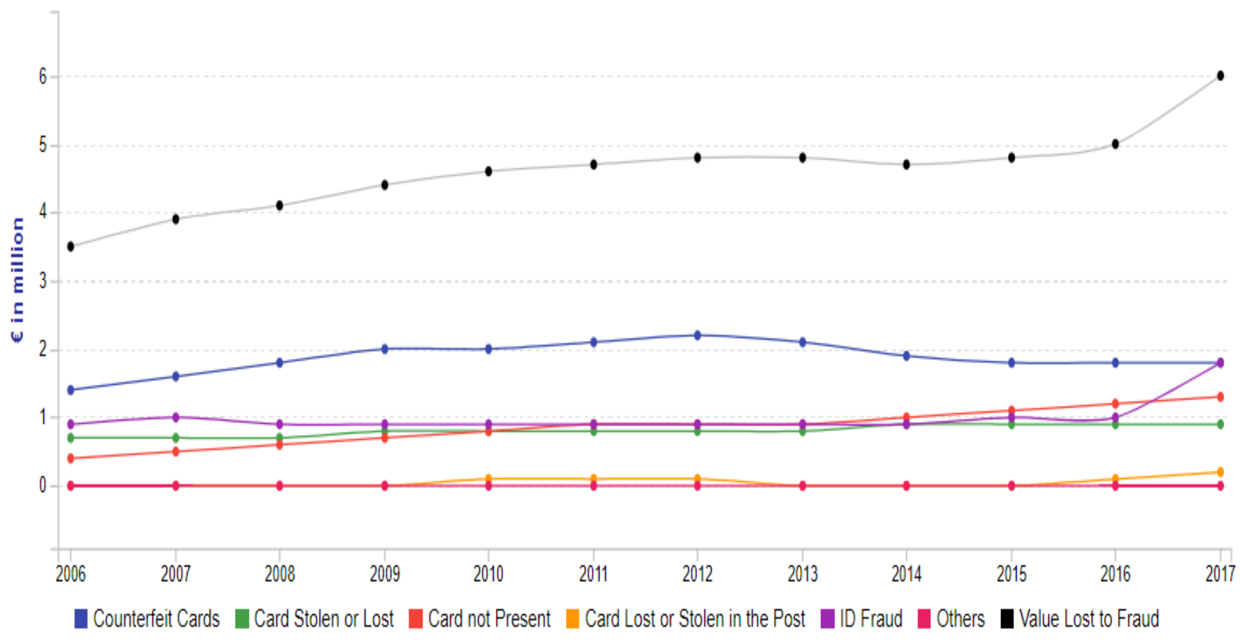
Visalakshi Meenakshi Sundaram

Section 1: Descriptives of the data

1.1 Addressing What, Why

What: The data represents the credit card transaction details of various customers over two days in time. This was collected in September 2013 in Europe. We have a total of 284,807 transactions and 31 features.

Why: With the ease of digital transactions for consumers comes the biggest challenge for banking sector, the rise in fraudulent transactions. FICO (https://www.fico.com/europeanfraud/) reports that there has been substantial increase in digital fraud in 2017 and shows the trend from 2006. The below graph shows the increase of fraud(in millions) for Austria year over year:



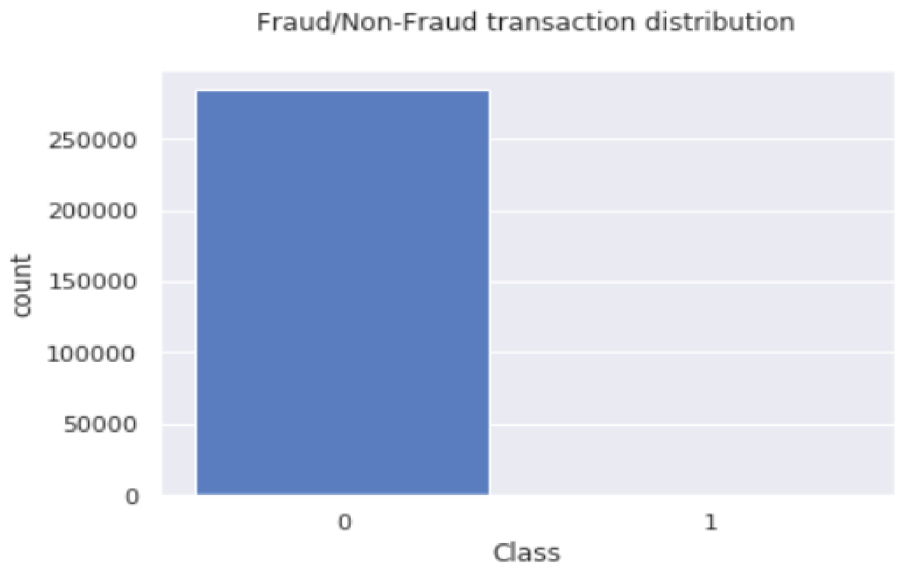
Given the social value of the project, there are also technical elements which made the project even more challenging.
i) Transactions - The data is highly unbalanced, the fraud transactions account only for 0.172% of all transactions.
ii) Features - There are 28 features which are already PCA (Principal Component Analysis)

transformed to maintain anonymity of the transactions. We are limited to apply any decision making on these set of features. We only have Time and Amount variables other than the PCA transformed numerical features.

1.2 Exploratory Data Analysis

1.2.1 Visualize the fraud/normal transactions imbalance:

```
The number of non-fradulent/normal transactions: 284315
The number of fradulent transactions: 492
The percentage of non-fradulent/normal transactions: 0.99827
The percentage of fradulent transactions: 0.00173
```



We can see that the data is heavily imbalanced from the graph above.

1.2.2 Check for Missing Values

```
# check missing value - but this will only capture missing values if they are in the form of Nan
print("Are there any missing values in the form of NaN?", df.isna().values.any())
```

```
Are there any missing values in the form of NaN? False
```

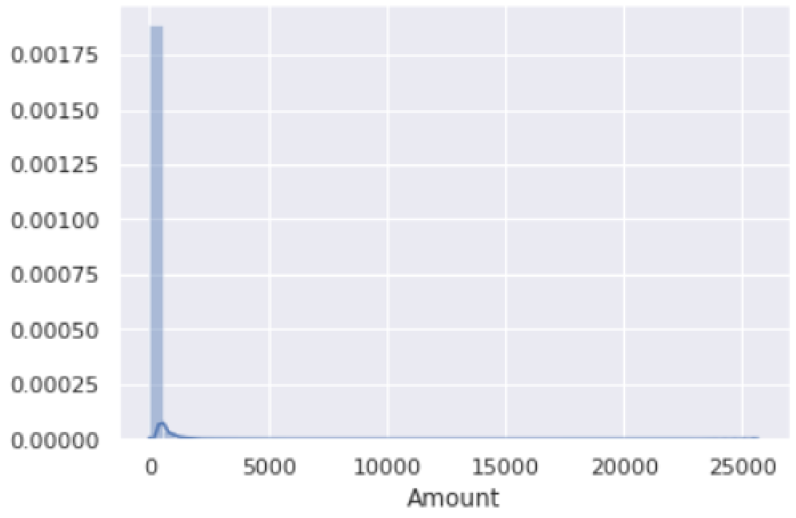
```
print("The number of transactions with amount = 0           : ",sum(df.Amount == 0))
print("The number of fradulent transactions with amount = 0 : ", sum(df[df['Amount']==0].Class ==1))
print("The number of normal transactions with amount = 0    : ",sum(df[df['Amount']==0].Class ==0))
```

```
The number of transactions with amount = 0           :  1825
The number of fradulent transactions with amount = 0 :  27
The number of normal transactions with amount = 0    :  1798
```

There were no null values even though there were some records where the amount feature had values of zero.

1.2.3 Amount Variable

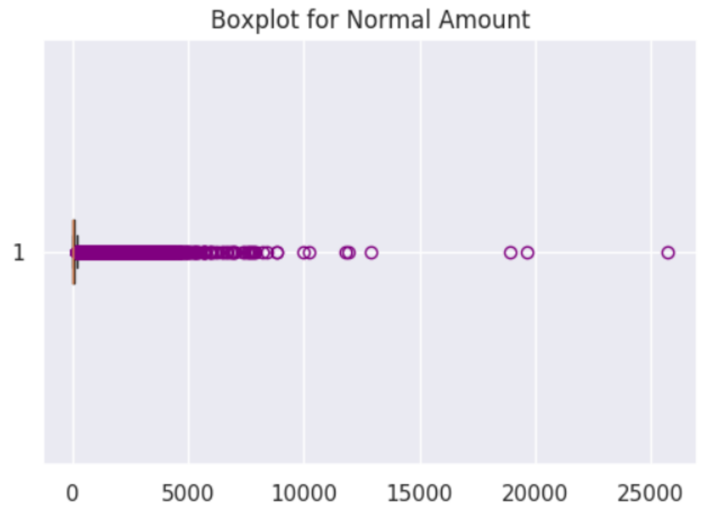
```
(0.0, 25691.16)
```



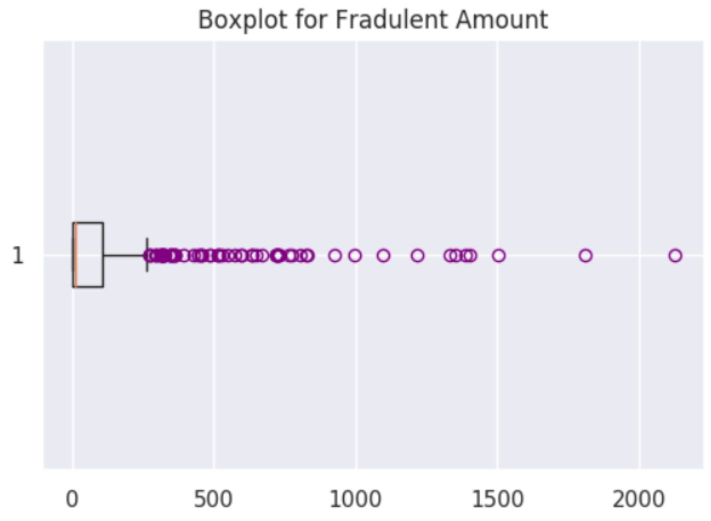
We can see that the amounts are heavily skewed to the right, with majority of transactions having amounts less than 500. However, there are many large outliers.

Then looking at the amount distributions in fraud and non-fraud classes separately, we don't see larger transactions in the fraud set. Instead, the extremely high transactions lie in the non-fraudulent class.

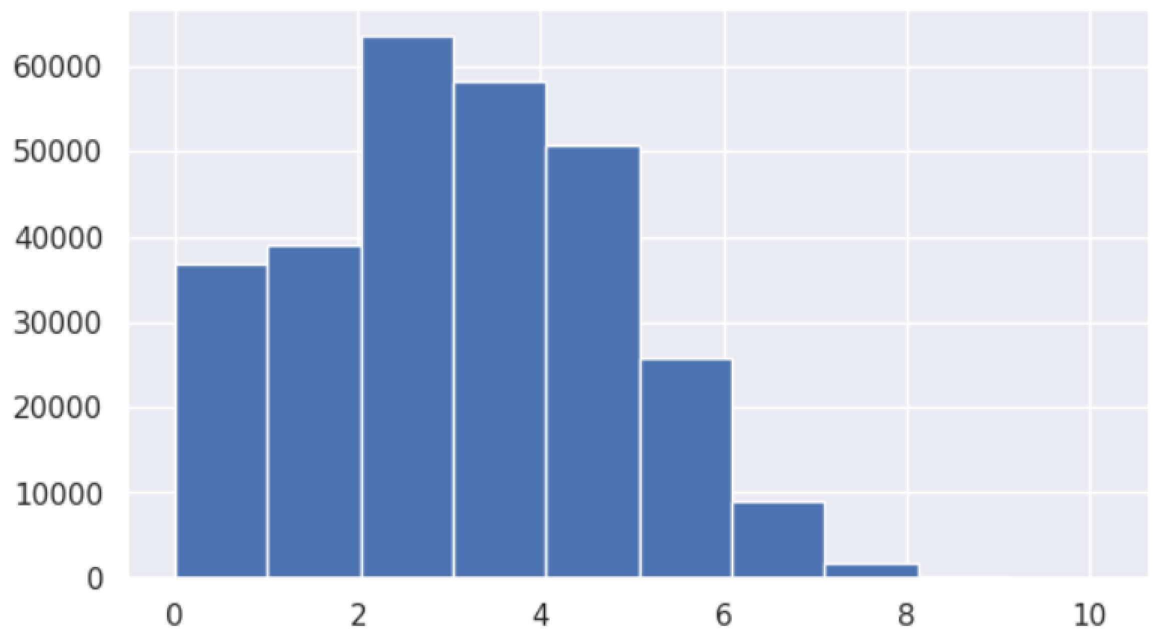
```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```



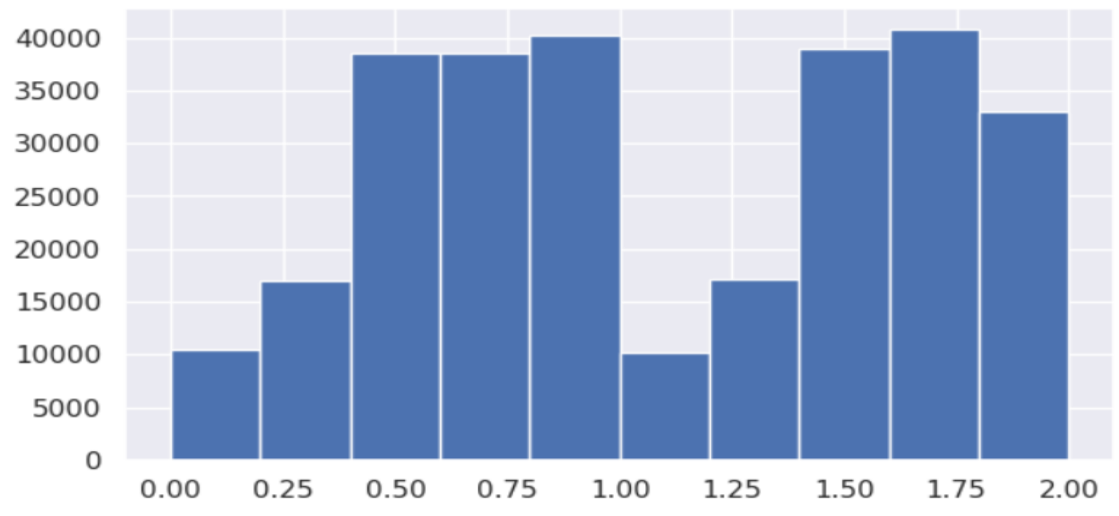
```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```



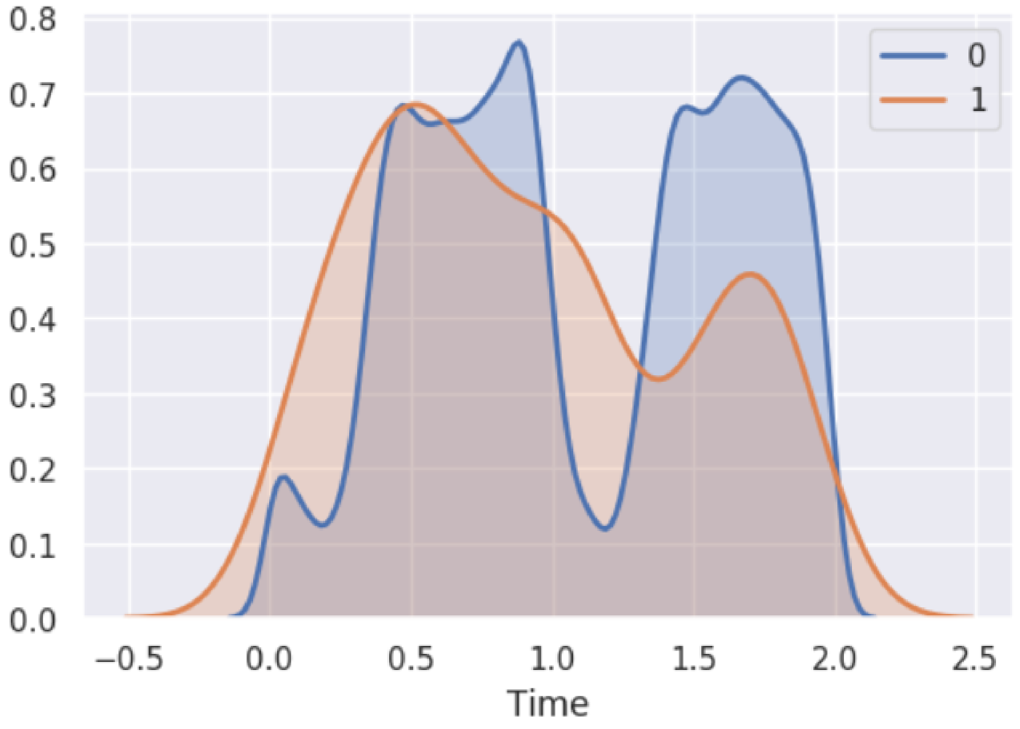
To address this skewness, we log transformed the amounts and the distribution looks good. We'll use log(amount) in our analysis later.



1.2.4 Time Variable

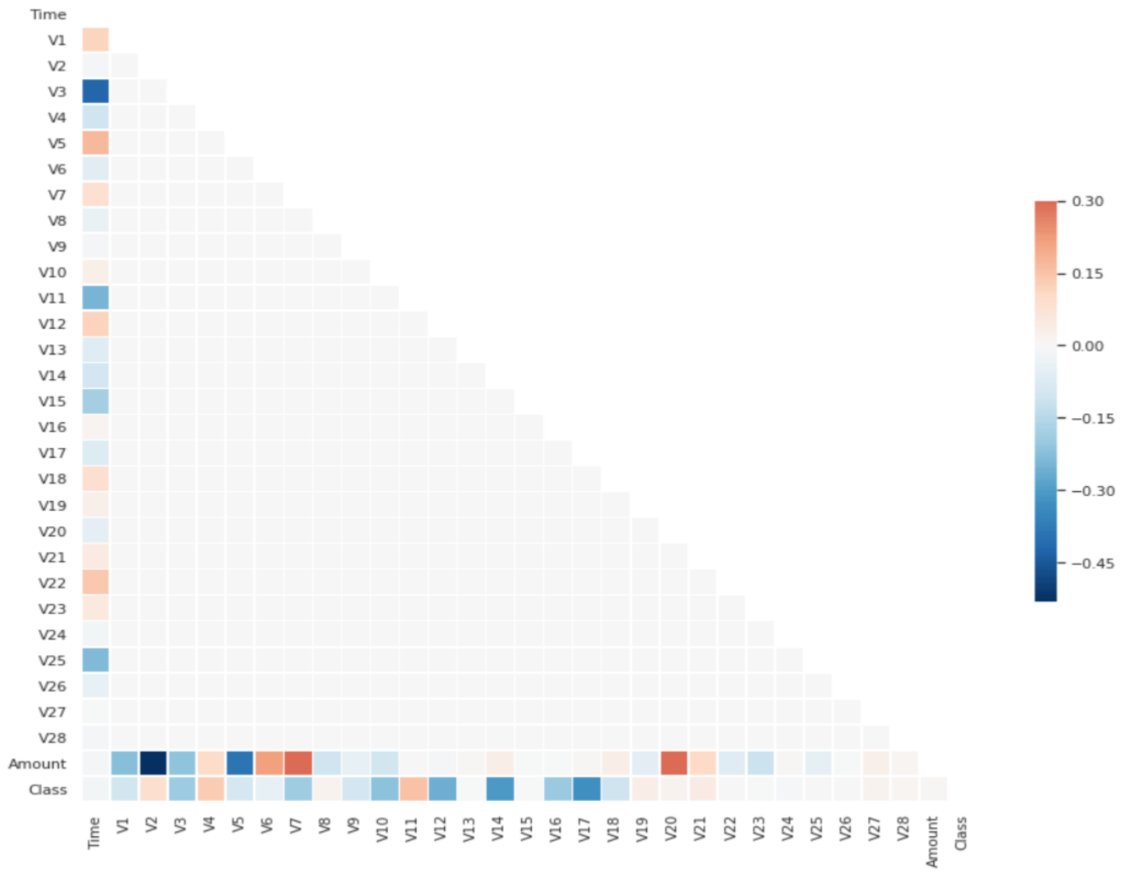


The time is measured as elapse time in seconds after first transaction. We plot the distribution after transforming it to days and found nothing but daily pattern.



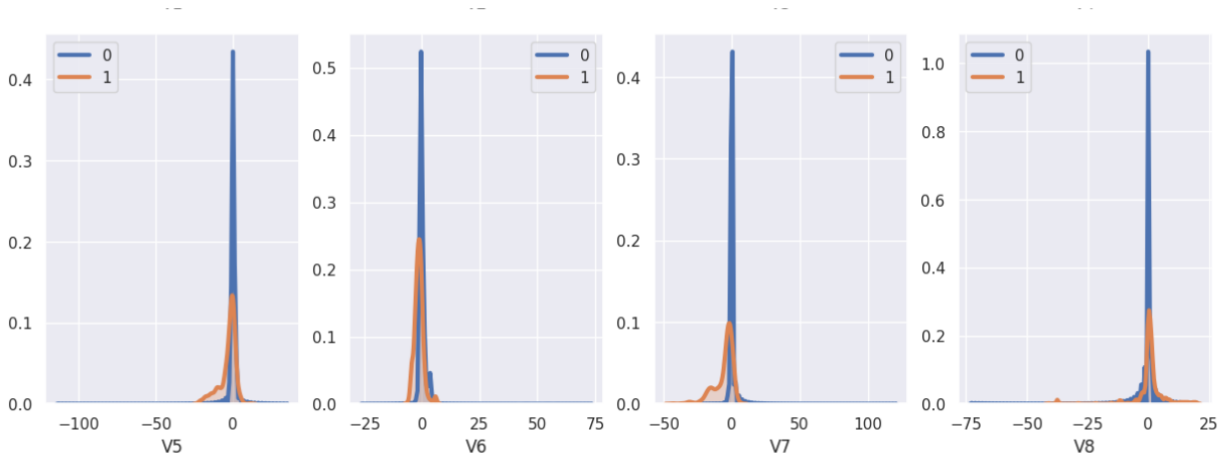
Also, comparing the density plots of fraud and normal transactions along time, nothing stands out significantly. Since we are not given any information about the timezone of the transactions origin or time stamp or any geography related information, we conclude this variable won't contribute to the prediction and thus remove it from the analysis.

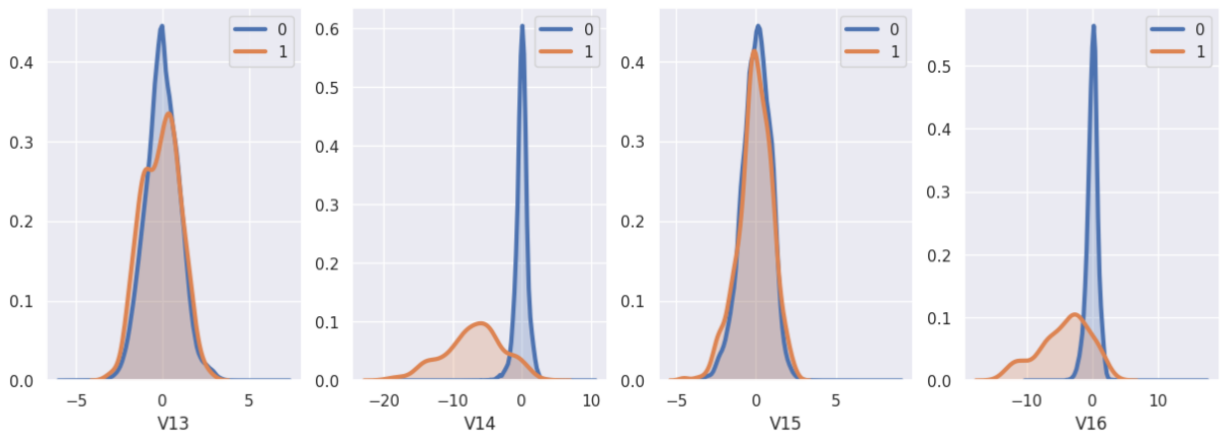
1.2.5 Correlation



There are no correlations among V1-V28 as they are PCA transformed. There are correlations among Amount and other PCA transformed variables.

1.2.6 PCA features and Class exploration





Since this is a classification problem, it raises the question that which of these features can really explain the differences between two classes. We tried to compare the density plots of V1-V28 between fraudulent and non-fraudulent classes and check for any obvious differences. It seems that $'V28',' V27',' V25',' V24',' V23',' V22'$, $'V21',' V20',' V15',' V13',' V8'$ don't explain the variances and we keep the plots of the ones that do.

Section 2: The Predictive Task

The predictive task here is clearly a classification problem. The goal is to find the optimal classification model that can tell fraudulent transactions from non-fraudulent ones.

Given the data set is already PCA transformed, we will attempt to do feature engineering and selection, involve sampling methods and proceed with classification algorithms. To compare classification algorithms, the evaluation metric should not only be recall but also precision. Both these are factored into F1 score which we will use to compare different models. The evaluation metric is defined as below. Also, we'll plot confusion matrix to present our results in a more straightforward way.

$$F_1 = 2 * (precision * recall)/(precision + recall)$$

The key problem to address here is the imbalance issue. We approached this problem through cost function adjustment and sampling techniques. Mainly 3 types of algorithms are used here, logistic regression, random forest and gradient boosting.

Section 3: Models

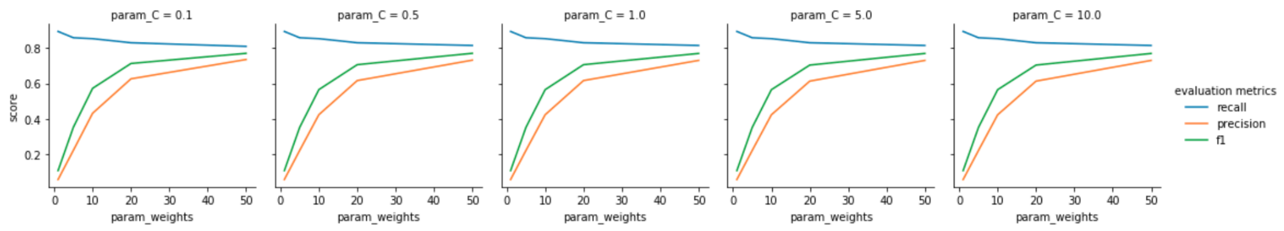
3.1 Logistic Regression

3.1.1 Adjust Cost Function Weights

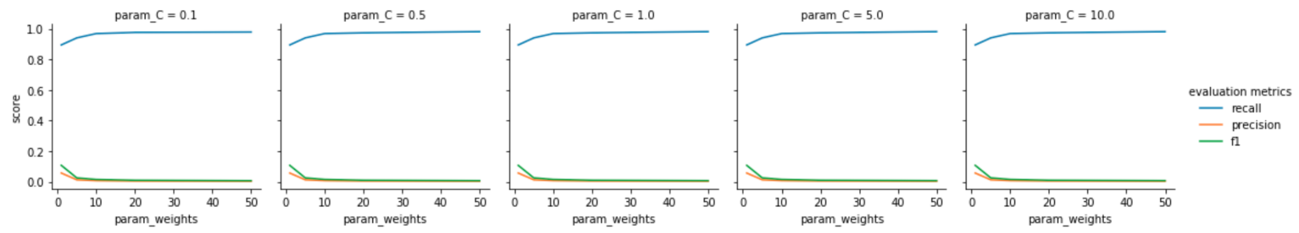
Since we are dealing with highly imbalance data set, we tried to adjust the cost function weights to cope with this problem. In searching for the optimal weights, as well as the optimal regularization parameters, we used GridSearchCV to tune the parameters. To be more specific, we first balanced the weights by positive and negative class size, then we associated a series of multipliers to positive class as below.

```
# use GridSearchCV to test class_weight and regularization parameters

neg_weight = len(y_train)/(2*np.bincount(y_train.ravel())[0])
pos_weight1 = len(y_train)/(2*np.bincount(y_train.ravel())[1])
pos_weight5 = len(y_train)/(2*np.bincount(y_train.ravel())[1]*5)
pos_weight10 = len(y_train)/(2*np.bincount(y_train.ravel())[1]*10)
pos_weight20 = len(y_train)/(2*np.bincount(y_train.ravel())[1]*20)
pos_weight50 = len(y_train)/(2*np.bincount(y_train.ravel())[1]*50)
```

Along the training process, we used F1 score, recall and prediction score as the main evaluation metrics. The result shows that as we increase the penalty weight, the recall rate, F1 and precision rate tend to merge around 0.8, meaning that we get a more balanced prediction results.

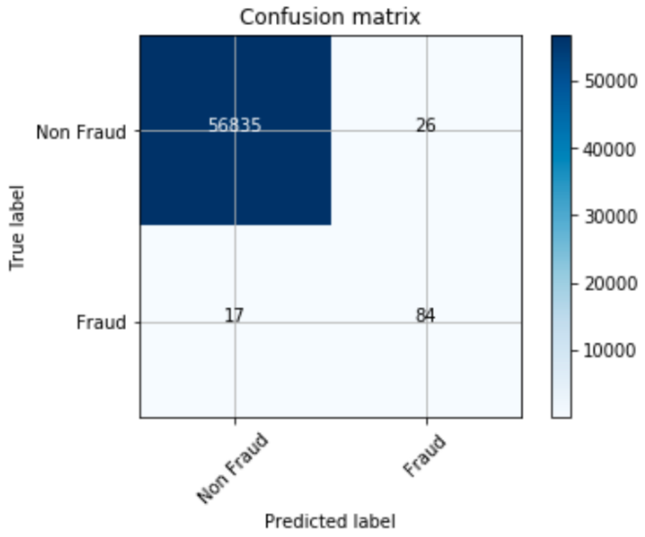


We also associated more weights to negative class. The false negatives got over penalized that the recall rate is over 0.9 but the precision rate is very low.



In this part of analysis, the optimal parameters we reach are a regularization parameter of 5 and class weight of 0: 0.5008595144512736, 1: 5.827237851662404. Below is our result.

```
Accuracy is: 0.9992275450415853
F1 score is: 0.7962085308056872
Recall is: 0.8316831683168316
Precision is: 0.7636363636363637
[[56835    26]
 [   17    84]]
```



The advantage of this model is that we can choose the weights to penalize false predictions and adjust the penalty degree as we want. But it's relatively hard to achieve both high recall rate and precision rate, as the parameters to work with are limited.

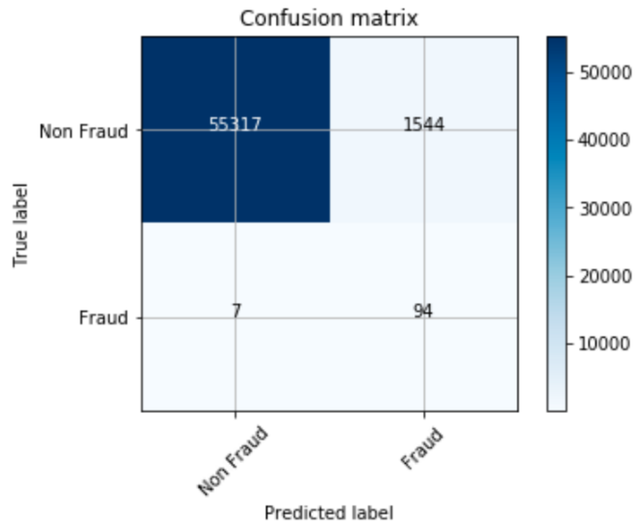
3.1.2 Over-sampling with SMOTE

By adjusting the cost function, the recall rate is not satisfactory as we leave around 17% of fraudulent cases undetected. Next, we try re-sampling techniques to balance out our training data before feeding into training process. Here we used SMOTE, which is one of the over-sampling techniques that will increase the positive class size to reach desired ratio, 1:1 in our case.

```
Before OverSampling, counts of label '1': [391]
Before OverSampling, counts of label '0': [227454]

After OverSampling, the shape of train_X: (454908, 18)
After OverSampling, the shape of train_y: (454908,)

After OverSampling, counts of label '1': 227454
After OverSampling, counts of label '0': 227454
```

Since the sampled data is already balanced, we don't tune class weights in this approach. Like earlier, we use GridSearchCV to pick the optimal regularization parameter. F1 is our evaluation metrics and we end up with 5 as the best lamda. The recall rate is 93% but the precision rate is 3%. The result is highly imbalanced and unsatisfactory.

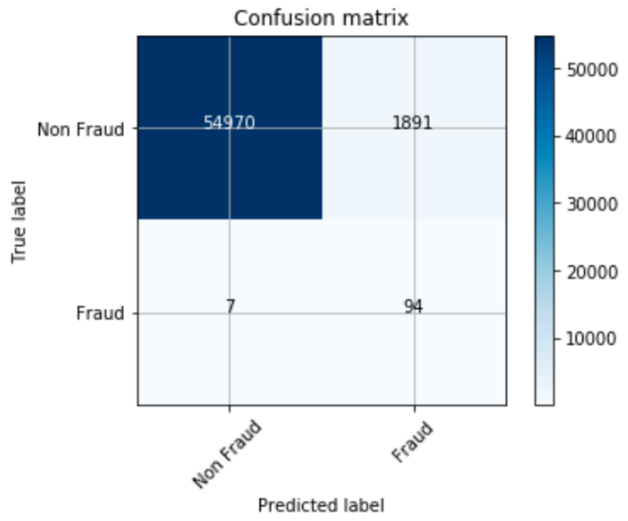
```
Accuracy is: 0.9347450473502334
Recall is: 0.9306930693069307
Precision is: 0.057387057387057384
[[55317  1544]
 [    7    94]]
```



3.1.3 Under-Sampling

Then instead of raising positive size, we tried under-sampling techniques where we cut down the negative size to reach a balanced data set. As previous, we used GridSearchCV to tune the regularization parameter and used F1 as evaluation metrics. Again, the high recall rate comes with a cost of low precision rate.

```
Accuracy is: 0.9501278772378516
Recall is: 0.9306930693069307
Precision is: 0.0473551637279597
[[54970  1891]
 [    7    94]]
```



3.2 Random Forest

For Random Search, each evaluation generates a new heap tree and gets a symbolic function from that heap tree, then it compares the MAE of this symbolic function to current best one, and selects the better one.

Random forest classification algorithm works based off decision tree model. Each decision tree in the forest selects a random set of features and a set of random of training records. After prediction, it selects the final output based on which classification has the maximum votes. If we have more trees in the model, it will avoid over-fitting issues and random forest has the power to handle a large data set with higher dimensions.

Steps followed in this approach:

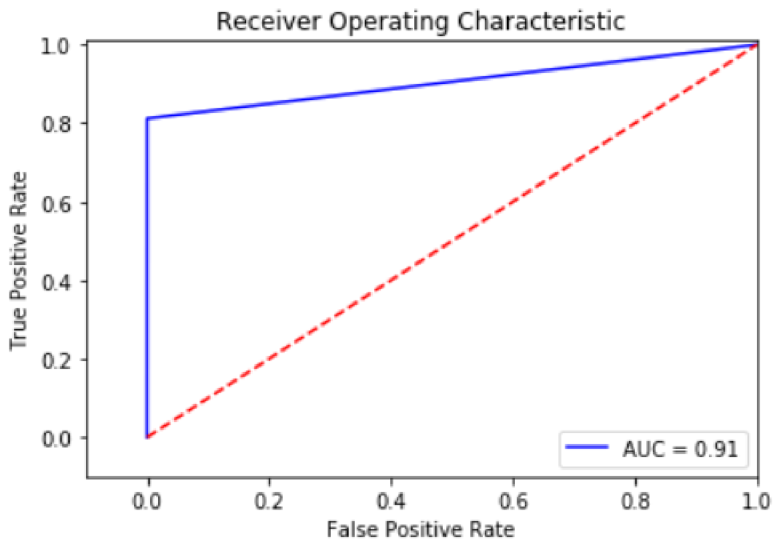
a). We used a GridSearchCV and tuned the maximum features and number of tree parameters using under-sampled data to get the best parameters. The scoring method used to determine the best parameters is $F1$.

```
0.9389936062007237
{'max_features': 'auto', 'n_estimators': 600}
```

0.93 is the F1 score for the model tuned with under-sampled data, with the best parameters 600 trees and auto select for the number of features to consider when looking for the best split.

b) Using the best parameters, trained the model again with the original training set and also the SMOTE training set.

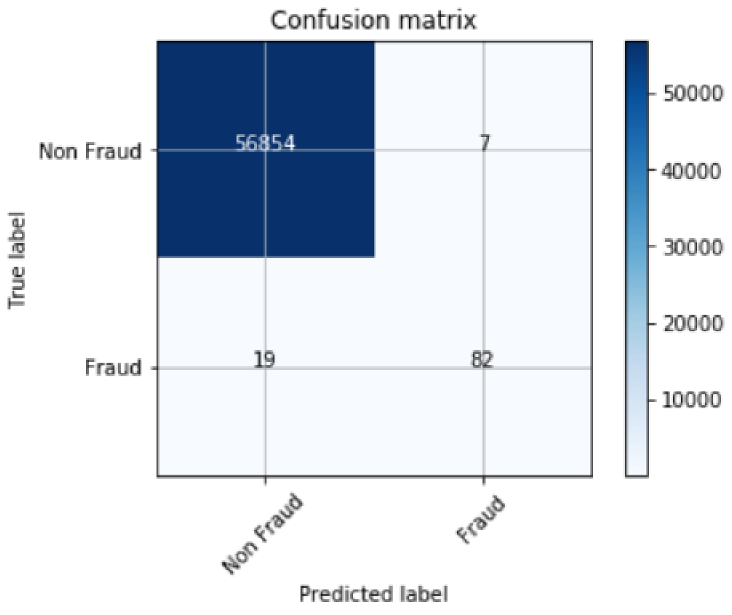
c) Predicted the split of test data with the above trained model.



The Area Under Curve - ROC is 0.86. While the other error measures and confusion matrix are below:

```
Accuracy is 0.9995435553526912
Precision is 0.9213483146067416
Recall is 0.8118811881188119
F1_score is 0.8631578947368421
```

```
[[56854     7]
 [   19    82]]
```



Issues that were faced using Random – Forest Classification:

a) Using SMOTE data for tuning:

We attempted to train the Random Forest classification model using the oversampled data from SMOTE. There were scalability problems due to higher volume of data to be trained and optimized using GridSearchCV. Since GridSearchCv selects the best parameters by running all the combinations of all parameters given, it couldn't process the huge training data.

b) GridSearchCV parameter tuning:

Tuning some of the parameters in Random Forest took up a lot of time even with the undersampled data. Adding the below parameters(even just one of them) made the processing very slow and could not complete the grid search.

- Criterion : To measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

- Max depth: The maximum depth of the tree (4-8)
- Min Samples split: The minimum number of samples required to split an internal node (2-5)

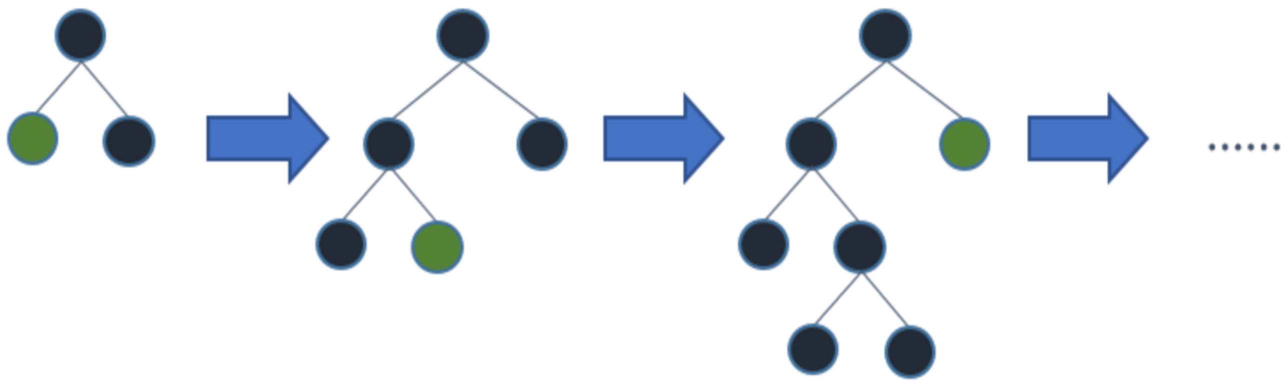
c) Processing time:

The weakness of the model is the processing time. As we extend the number of trees, it runs slower.

3.3 LightGBM

LightGBM is a highly efficient gradient boosting framework that uses tree based learning algorithms. It is designed to have faster training speed and better accuracy. However, it also might have the problem of overfitting to training set. We can improve the situation of overfitting by tuning some of the parameters. The following report is going to implement this method.

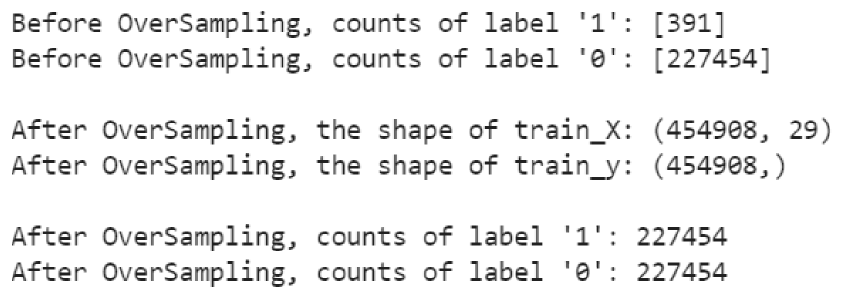
Advantages and disadvantages of lightGBM :
LightGBM is extremely efficient when dealing with large data. So, the calculation time is much shorter than Random Forest and XGBoost, especially training on the oversampling dataset.LightGBM uses leaf-wise generating strategy, which increases its potential of overfitting. But we can improve this by tuning parameters.



Leaf-wise tree growth

Steps followed in this approach:
a) Sample the data using SMOTE

SMOTE is short for 'Synthetic Minority Over-sampling Technique'. It is an approach of oversampling that is widely used to deal with imbalanced dataset. Since lightGBM is an efficient way of handling large dataset, we did the classification by oversampling instead of undersampling. After OverSampling, counts of label '1' and '0' are both 227454.

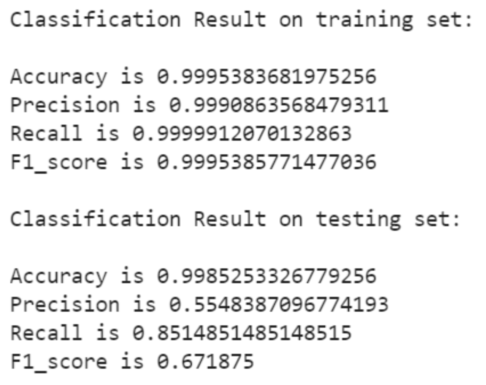


```
Before OverSampling, counts of label '1': [391]
Before OverSampling, counts of label '0': [227454]

After OverSampling, the shape of train_X: (454908, 29)
After OverSampling, the shape of train_y: (454908,)

After OverSampling, counts of label '1': 227454
After OverSampling, counts of label '0': 227454
```

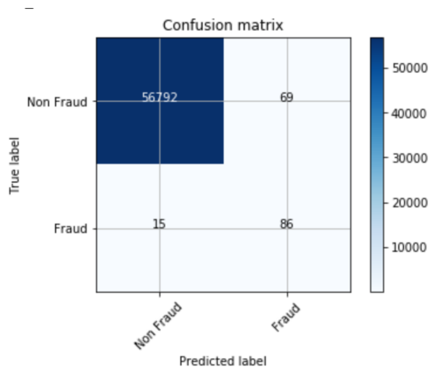
b) Tuning the parameters step by step using RandomizedSearchCV

RandomizedSearchCv does a randomized search overall all parameters. Different from GridSearchCV, RandomizedSerachCV is more efficient. So we first try tuning the parameters using RandomizedSearchCV to get a narrow range of reasonable choices in a large parameter space. Then we move to GridSearchCV to settle down the best parameters combination.

If we use just use the default value of lightGBM to do the classification, the problem of overfitting will be extremely severe. Following screenshot is the score of classification before tuning the parameters.

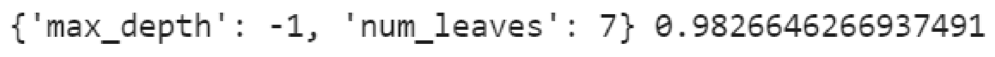


```
Classification Result on training set:

Accuracy is 0.9995383681975256
Precision is 0.9990863568479311
Recall is 0.9999912070132863
F1_score is 0.9995385771477036

Classification Result on testing set:

Accuracy is 0.9985253326779256
Precision is 0.5548387096774193
Recall is 0.8514851485148515
F1_score is 0.671875
```



i) Important parameters related to accuracy

max_depth: Set the depth of tree. Overfitting might appear it is set to be too deep.
num_leaves: Tunning the complexity of tree

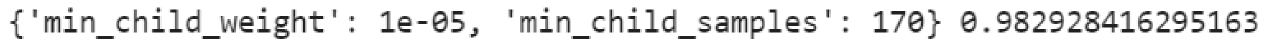


```
{'max_depth': -1, 'num_leaves': 7} 0.9826646266937491
```

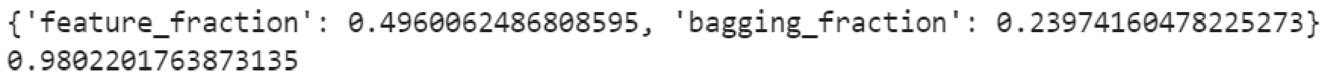
We will then continue tune other parameters to alleviate the problem.

ii) Important parameters to avoid overfitting

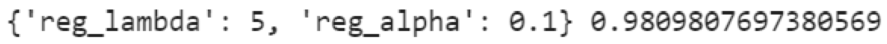
min_child_samples: It is also named as min_data_in_leaf. Setting this parameter to a large number will avoid generate a over-deep tree, but might also lead to underfitting.
min_child_weight: It is also named as min_sum_hessian_in_leaf, standing for Minimum sum of hessians in one leaf to allow a split. Higher values will potentially decrease overfitting.
feature_fraction: Sampling the features. This parameter can deal with overfitting and at the same time increase training speed.
bagging_fraction: It will randomly select part of data without resampling. It can deal with overdfitting and at the same time increase training speed.
reg_alpha & reg_lambda: Regularizer(L1-NORM and L2-NORM)



```
{'min_child_weight': 1e-05, 'min_child_samples': 170} 0.982928416295163
```



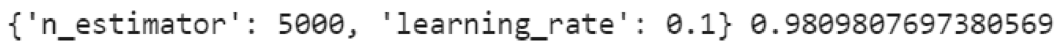
```
{'feature_fraction': 0.4960062486808595, 'bagging_fraction': 0.23974160478225273}
0.9802201763873135
```



```
{'reg_lambda': 5, 'reg_alpha': 0.1} 0.9809807697380569
```

iv) Learning rate and number of iteration

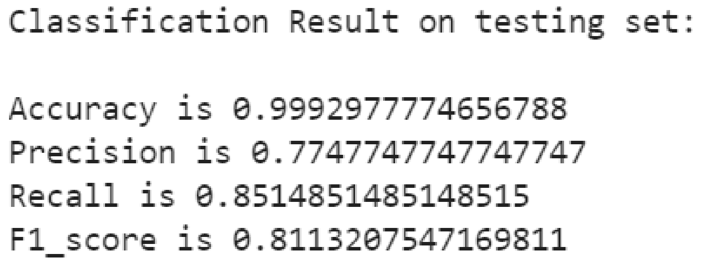
learning_rate: Tuning learning_rate can help improve the accuracy of the model.
n_estimators: Number of boosting iterations



```
{'n_estimator': 5000, 'learning_rate': 0.1} 0.9809807697380569
```

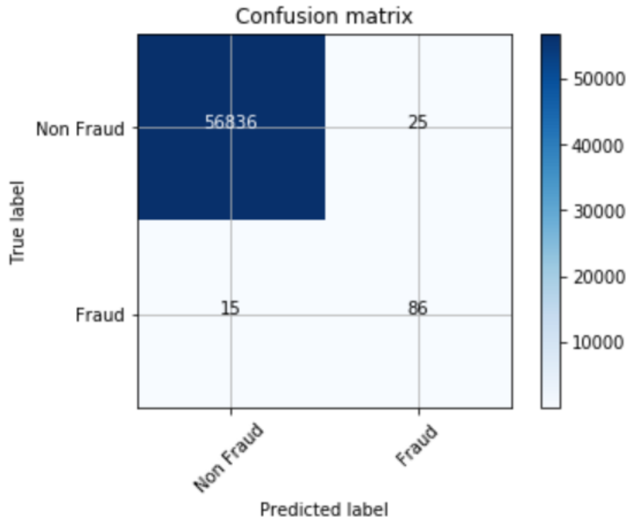
c) Predict the test set with the above trained model.

The final classification score is as the following:



```
Classification Result on testing set:

Accuracy is 0.9992977774656788
Precision is 0.7747747747747747
Recall is 0.8514851485148515
F1_score is 0.8113207547169811
```

```
[[56836    25]
 [   15    86]]
```



Section 4: Literature Survey

The dataset was collected and analyzed during a research collaboration of Worldline and the Machine Learning Group [1] of ULB (Université Libre de Bruxelles) on big data mining and fraud detection. It contains the credit card transactions of European Cardholder in September 2013 with class fraud and non-fraud.

This is also a Kaggle competition focusing on classification of imbalanced dataset. Two most widely used method is Under-sampling and SMOTE, Synthetic Minority Oversampling Technique[2].

There are three main supervised methods to do the classification tasks: Logistic Regression, Gradient Boosting and Random Forest. We tried all of this three models in this report. Actually, experiments suggest that data characteristics considerably impact the classification performance of the methods[3]. There is no single method that clearly outperforms all methods in all problem situations.

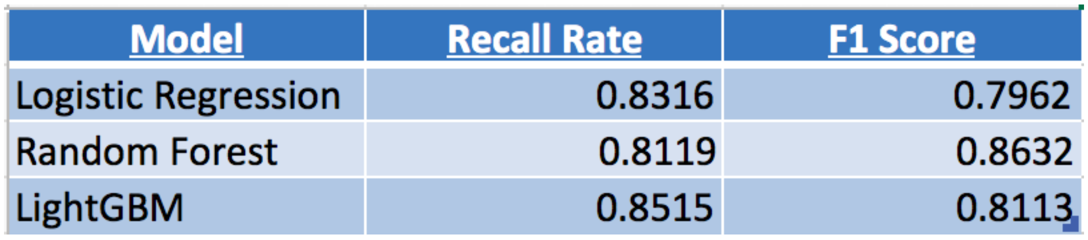
Performance metrics in classification are fundamental in assessing the quality of learning methods and learned models. Analysis shows that most of the measures used in machine learning and pattern recognition for evaluating classifiers measure different things. Especially for problems with imbalanced class distribution, the correlations of different measurements are worse[4]. In our report, we used confusion matrix, accuracy, precision rate, recall rate as well as F1 score to evaluate our model.

Banking sector traditionally detected fraud through manual reviews. Now with the advancement of technology, high scale banking organizations are slowly moving towards self learning algorithms from supervised learning. There have been huge regulations imposed on the methodologies[5], but the industry is trying to get past the scrutiny.

Section 5: Results and Conclusions

In this predictive task, we mainly used 3 models to detect the fraudulent transaction, logistic regression, random forest and gradient boosting. Since the data is highly imbalanced, we tried cost function adjustment, oversample and undersample techniques to reach a balanced training set.

The 3 optimal models have achieved similar results. LightGBM has the highest recall rate while Random Forest has the highest F1 score. Due to scalability issues, we could not tune some of the models to the best of our abilities. If we tune them appropriately, we believe we can reach higher F1 score and corresponding precision and recall rates.



Model	Recall Rate	F1 Score
Logistic Regression	0.8316	0.7962
Random Forest	0.8119	0.8632
LightGBM	0.8515	0.8113

We also tried support vector machine and XGBoosting during our analysis, but the training process is too slow so we had to turn it off before reaching any results. Comparatively speaking, logistic regression is straightforward and easy for interpretation, though in our case the PCA transformed features made it difficult to make economic interpretations. Random forest is relatively slower to train, but it's based on multiple trees and yields more stable results. LightGBM is efficient to train and we have many parameters at our disposal. The detailed parameter interpretation and model performance are listed at the end of each model.

Section 6: Reference

[**1**] Datasource: Worldline and the Machine Learning Group http://mlg.ulb.ac.be
[**2**] Ajinkya More,(Aug 2013). Survey of resampling techniques for improving classification performance in unbalanced datasets
[**3**] Melody Y. Kiang, (May 2002). A comparative assessment of classification methods
[**4**] C. Ferri, J. Hernández-Orallo, R. Modroiu(2009). An experimental comparison of performance measures for classification
[**5**] Article on trends in banking fraud: https://www.marutitech.com/machine-learning-fraud-detection/