CAPSTONE REPORT:
FIRST NAME: Gopalakrishnan
LAST NAME: Kalarikovilagam Subramanian
M12821535

INTRODUCTION:

The profit of credit card companies is directly dependent on the number of customers and the amount spent by the customers on credit card transactions. The main deterrent in customers getting and using a credit card is the fear of fraudulent transactions using their card. The onus falls on the credit card companies to allay this fear by making sure they detect such fraudulent transactions and notify the customers so that they do not lose money.

The goal of this project is to build a model for detecting fraudulent transaction using previous documented data of fraudulent transactions, the available variables and the cash transacted. This model is used for predicting fraudulence in any future transactions so that the customers are not charged for items they did not purchase.

The data set contains 30 variables. Two of the variables are ‘Amount’ of the transaction and ‘Time’ of the transaction which is the time elapsed after the first transaction. The other 28 variables are named V1-V28 and are a result of Principal Component Analysis (PCA) transformation. All the variables are numeric. The output variable is ‘Class’ which is the response variable and can take value of 1 in case of fraud and 0 otherwise. The data set presents the transaction over 2 days and it has 284,807 observations. Out of the 284,807 transactions there are 492 fraud cases. Therefore, the data set is highly unbalanced as the frauds account for 0.172% of all the transactions.

The output variable is binary and hence is a classification problem. Therefore the following methods can be used:

1. Logistic regression
2. Random Forest
3. Gradient Boosting Machine
4. Support Vector Machine (SVM)

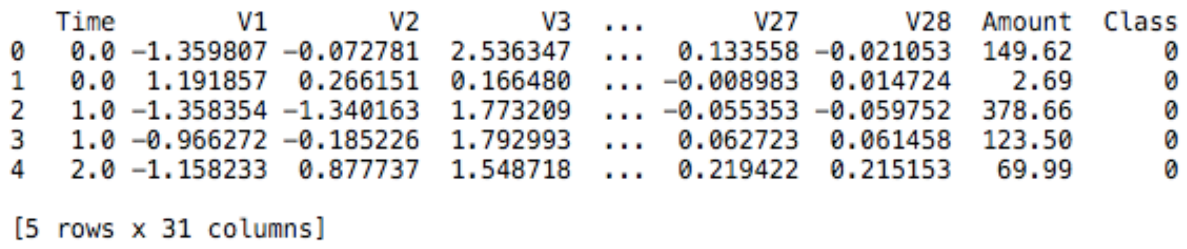
DATA CLEANING AND EXPLORATION:

The following library/modules are used for this work:

```
import pandas as pd                              # Pandas for using data frame
import matplotlib.pyplot as plt                  #for plotting
import numpy as np                               #For using numpy arrays
import seaborn as sns                            # For making interactive plots

import datetime                                        # to detail with date and time
from sklearn.preprocessing import StandardScaler       # for preprocessing the data
from sklearn.ensemble import RandomForestClassifier    # Random forest classifier
from sklearn.tree import DecisionTreeClassifier         # for Decision Tree classifier
from sklearn.svm import SVC                             # for SVM classification
from xgboost import XGBClassifier                       # For XG-Boost Classifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split      # to split the data
from sklearn.model_selection import KFold                  # For cross vbalidation
from sklearn.model_selection import GridSearchCV          # for tunnig hyper parameter it
will use all combination of given parameters
from sklearn.model_selection import RandomizedSearchCV  # same for tunning hyper
parameter but will use random combinations of parameters
from sklearn.metrics import
confusion_matrix,recall_score,precision_recall_curve,auc,roc_curve,roc_auc_score,classifi
cation_report
import warnings
```

The data is imported using the following commands:

```
data = pd.read_csv("creditcard.csv")
data.head()
```



```
   Time        V1        V2        V3  ...       V27       V28  Amount  Class
0   0.0 -1.359807 -0.072781  2.536347  ...  0.133558 -0.021053  149.62      0
1   0.0  1.191857  0.266151  0.166480  ... -0.008983  0.014724    2.69      0
2   1.0 -1.358354 -1.340163  1.773209  ... -0.055353 -0.059752  378.66      0
3   1.0 -0.966272 -0.185226  1.792993  ...  0.062723  0.061458  123.50      0
4   2.0 -1.158233  0.877737  1.548718  ...  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
```

There are 31 variables. The first variable is the time of the transaction. For this problem time is not an important variable. The next 28 are PCA variables and they are unknown. There is the amount variable which signifies the amount of the transaction using the credit card and is a float data type. The Class indicates the classification of the transaction as a fraud of legitimate.

The data is then checked for regular information about the different variables:

```
print(data.info())
```

Output:

```
Data columns (total 31 columns):
Time     284807 non-null float64
V1       284807 non-null float64
V2       284807 non-null float64
V3       284807 non-null float64
V4       284807 non-null float64
V5       284807 non-null float64
V6       284807 non-null float64
V7       284807 non-null float64
V8       284807 non-null float64
V9       284807 non-null float64
V10      284807 non-null float64
```

```
V11      284807 non-null float64
V12      284807 non-null float64
V13      284807 non-null float64
V14      284807 non-null float64
V15      284807 non-null float64
V16      284807 non-null float64
V17      284807 non-null float64
V18      284807 non-null float64
V19      284807 non-null float64
V20      284807 non-null float64
V21      284807 non-null float64
V22      284807 non-null float64
V23      284807 non-null float64
V24      284807 non-null float64
V25      284807 non-null float64
V26      284807 non-null float64
V27      284807 non-null float64
V28      284807 non-null float64
Amount   284807 non-null float64
Class    284807 non-null int64
Dtypes: float64(30), int64(1)
```

The above output shows that there are NA/missing values in any of the variables.

The duplicates are removed:

```
data = data.drop_duplicates()
```

The Class variable is converted into type category and the number of null values are counted as a double check:

```
data.Class = data.Class.astype('category')
data.isnull().sum()
```

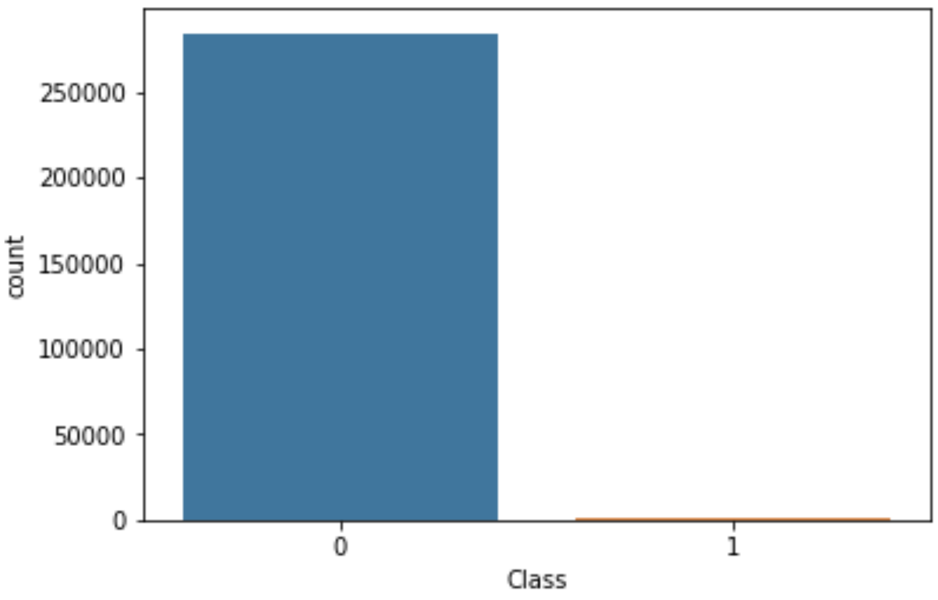
Output:

```
Time    0
V1      0
V2      0
V3      0
V4      0
V5      0
V6      0
V7      0
V8      0
V9      0
```

```
V10      0
V11      0
V12      0
V13      0
V14      0
V15      0
V16      0
V17      0
V18      0
V19      0
V20      0
V21      0
V22      0
V23      0
V24      0
V25      0
V26      0
V27      0
V28      0
Amount    0
Class     0
```

The Class variable is plotted as a bar chart here:

```
sns.countplot("Class",data=data)
```



The above chart shows that the data is highly skewed as expected and there is a lot of normal transactions and very few fraudulent transactions.

A count of the normal and fraudulent transactions are calculated as follows:

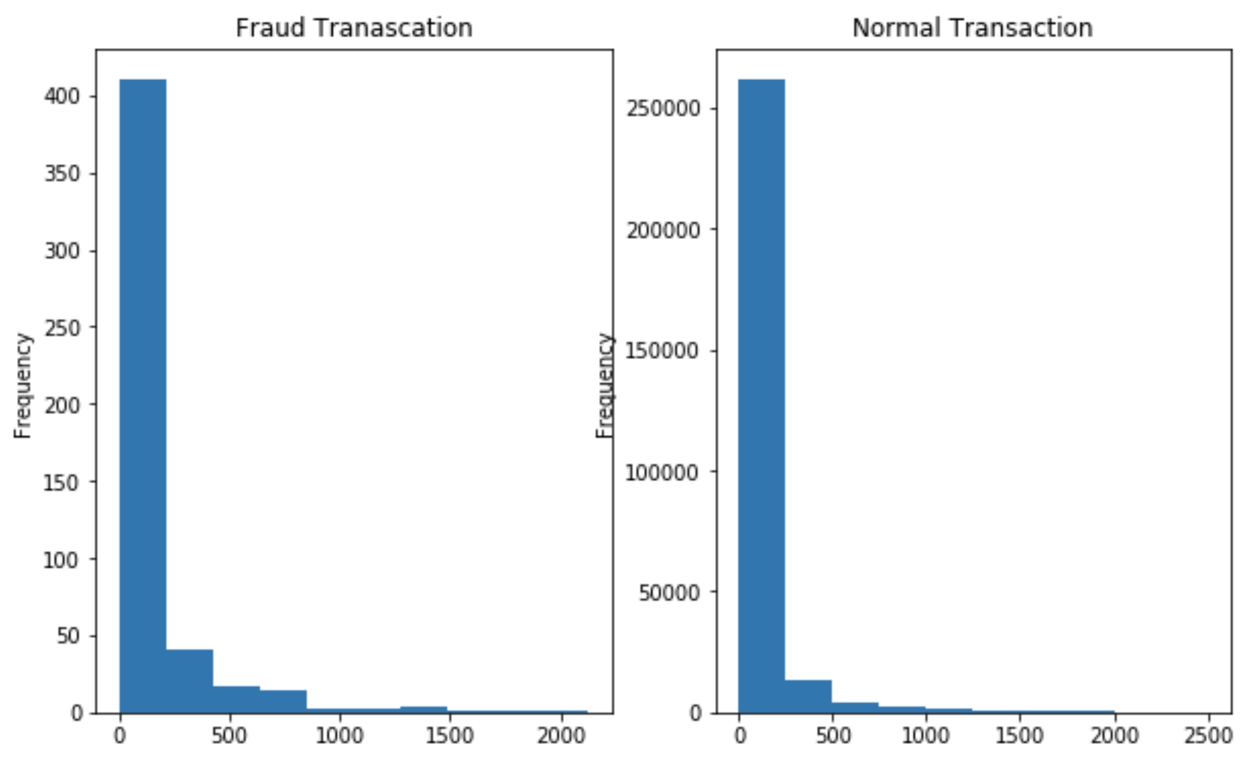
```
Normal_trans = len(data[data["Class"]==0])
Fraud_trans = len(data[data["Class"]==1])
print("Number of normal transactions", Normal_trans)
print("Number of Fraudulent transactions",Fraud_trans)
Perc_Normal_trans = Normal_trans/(Normal_trans+Fraud_trans)
print("percentage of normal transacation is",Perc_Normal_trans*100)
Perc_Fraud_trans = Fraud_trans/(Normal_trans+Fraud_trans)
print("percentage of fraud transacation",Perc_Fraud_trans*100)
```

Output:

```
Number of normal transactions 284315
Number of Fraudulent transactions 492
percentage of normal transacation is 99.82725143693798
percentage of fraud transacation 0.1727485630620034
```

The distribution of amounts for normal and transactions is plotted to see of there is a pattern:

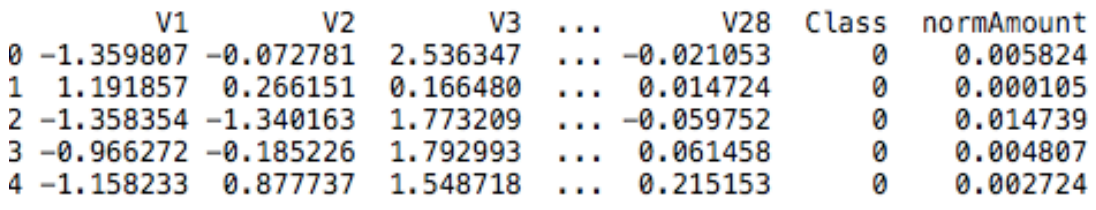
```
Fraud_trans_df = data[data["Class"]==1]
Normal_trans_df = data[data["Class"]==0]
plt.figure(figsize=(10,6))
plt.subplot(121)
Fraud_trans_df[Fraud_trans_df["Amount"]<= 2500].Amount.plot.hist(title="Fraud
Tranascation")
plt.subplot(122)
Normal_trans_df[Normal_trans_df["Amount"]<=2500].Amount.plot.hist(title="Normal
Transaction")
```



The normal transaction has most of the amount in 0-250 range as is the case for fraudulent transactions. There is no clear differentiating pattern between the two types of transactions.

The amount column values are varying a lot and could affect the models built using the data. Hence. The values in this column needs to be normalized. The time and old Amount column have been removed as they are useful in modeling the data.

data['normAmount'] = (data['Amount']-min(data['Amount']))/(max(data['Amount'])-min(data['Amount']))
data = data.drop(['Time','Amount'],axis=1)
data.head()



```
         V1        V2        V3  ...       V28  Class  normAmount
0 -1.359807 -0.072781  2.536347  ... -0.021053      0    0.005824
1  1.191857  0.266151  0.166480  ...  0.014724      0    0.000105
2 -1.358354 -1.340163  1.773209  ... -0.059752      0    0.014739
3 -0.966272 -0.185226  1.792993  ...  0.061458      0    0.004807
4 -1.158233  0.877737  1.548718  ...  0.215153      0    0.002724
```

DATA ANALYSIS:

The techniques of ‘under-sampling’ and ‘oversampling’ are used here for handling the imbalanced data. In under-sampling the majority class is under-sampled in the final data and in oversampling the minority class is replicated many times. By following the above two techniques some balance is restored in the data.

The data is separated into two data frames, one containing all the variables except the output and the other containing only the output:

```
X = data.ix[:, data.columns != 'Class']
y = data.ix[:, data.columns == 'Class']
```

A function for getting under-sampled data is created as follows:

```
def undersample(i):
    # Number of data points in the minority class
    no_fraud_trans = len(data[data.Class == 1])
    fraud_index = np.array(data[data.Class == 1].index)
    # Picking the indices of the normal classes
    normal_index = data[data.Class == 0].index
    # Out of the indices we picked, randomly select "x" number (number_records_fraud)
    rand_normal_index = np.random.choice(normal_index, no_fraud_trans*i, replace =
False)
    rand_normal_index = np.array(rand_normal_index)
    # Appending the 2 indices
    under_sample_index = np.concatenate([fraud_index,rand_normal_index])
    # Under sample dataset
    under_sample_data = data.iloc[under_sample_index,:]
    return(under_sample_data)
```

In this function the normal rows are selected as equal to or some multiples of number of fraud transactions. The number can be manipulated to change the ratios of normal and fraudulent transactions.

To predict the accuracy of a model a confusion matrix needs to be created. The various metrics of a confusion matrix are:

Accuracy = TP+TN/Total

Precision = TP/(TP+FP)

Recall = TP/(TP+FN)

Where,

TP = True positive means no of positive cases which are predicted positive

TN = True negative means no of negative cases which are predicted negative

FP = False positive means no of negative cases which are predicted positive

FN= False Negative means no of positive cases which are predicted negative

The recall is the best sense of measure as it will give us a sense of only the fraud transactions. This is because in this case the TP of finding fraud transactions will be high and FN of rejecting a true transactions will be low.

A function is made for preparing the confusion matrix as follows:

```
def model(model,train_data,test_data,train_output,test_output):
    clf= model
    clf.fit(train_data,train_output.values.ravel())
    pred=clf.predict(test_data)
    cnf_matrix=confusion_matrix(test_output,pred)
    print("the recall for this model is :",cnf_matrix[1,1]/(cnf_matrix[1,1]+cnf_matrix[1,0]))
    fig= plt.figure(figsize=(6,3))# to plot the graph
    print("TP",cnf_matrix[1,1,]) # no of fraud transaction which are predicted fraud
    print("TN",cnf_matrix[0,0]) # no. of normal transaction which are predited normal
    print("FP",cnf_matrix[0,1]) # no of normal transaction which are predicted fraud
    print("FN",cnf_matrix[1,0]) # no of fraud Transaction which are predicted normal
    sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
    plt.title("Confusion_matrix")
    plt.xlabel("Predicted_class")
    plt.ylabel("Real class")
    plt.show()
    print("\n----------Classification Report------------------------------------")
    print(classification_report(test_output,pred))
```

The original undersampled/oversampled data sets needs to be split into training and testing data sets. A function is created for doing the same:

```
## Creating the training and testing data
def data_prepration(x): # preparing data for training and testing as we are going to use
different data
    #again and again so make a function
    x_data = x.ix[:,x.columns != "Class"]
    x_output =x.ix[:,x.columns=="Class"]
    x_data_train,x_data_test,x_output_train,x_output_test =
train_test_split(x_data,x_output,test_size=0.3)
    print("length of training data")
    print(len(x_data_train))
    print("length of test data")
    print(len(x_data_test))
    return(x_data_train,x_data_test,x_output_train,x_output_test)
```

The test-train split library is used for doing this function. The ratio of the training to test data is taken in the ratio 0.7:0.3.

Logistic Regression:

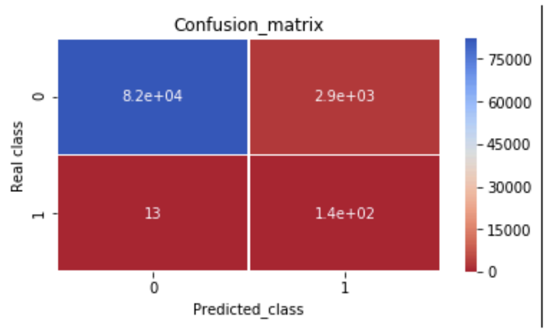
Logistic regression is first used to model the data. The following code illustrates the procedure. The undersampled training data is used for modeling and the entire test data is used for checking the accuracy of the model. The data-preparation function is used for generating the training and testing data using both the under-sampled and original data sets. There are four ratios which are considered for this example. This creates the following ratios of normal and fraud transactions : (0.5:0.5, 0.66:0.33, 0.75:0.25)

```
for i in range(1,4):
   Undersample_data = undersample(i)

US_train_input,US_test_input,US_train_output,US_test_output=data_prepration(Undersa
mple_data)
   train_input,test_input,train_output,test_output=data_prepration(data)
   #the partion for whole data
   print()
   clf=LogisticRegression()
   model(clf,US_train_input,test_input,US_train_output,test_output)
```

OUTPUT:

For the ratio 0.5:0.5

```
the recall for this model is : 0.9171974522292994
TP 144
TN 82400
FP 2886
FN 13
/Users/gk/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

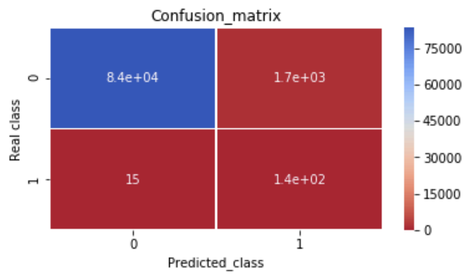


```
----------Classification Report------------------------------------
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     85286
           1       0.05      0.92      0.09       157

   micro avg       0.97      0.97      0.97     85443
   macro avg       0.52      0.94      0.54     85443
weighted avg       1.00      0.97      0.98     85443
```

The recall for this model is pretty good at 0.92. However, the precision is very low at 0.05

For the ratio 0.66:0.33

```
the recall for this model is : 0.9032258064516129
TP 140
TN 83581
FP 1707
FN 15
/Users/gk/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

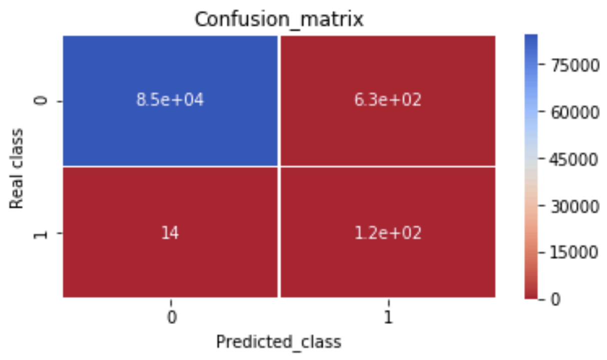


```
----------Classification Report------------------------------------
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     85288
           1       0.08      0.90      0.14       155

   micro avg       0.98      0.98      0.98     85443
   macro avg       0.54      0.94      0.56     85443
weighted avg       1.00      0.98      0.99     85443
```

When the ratio of fraudulent transactions in the data-set reduces the recall reduces to 0.9 and the precision improves to 0.08.

For the ratio 0.75:0.25

```
the recall for this model is : 0.8931297709923665
TP 117
TN 84683
FP 629
FN 14
```



```
----------Classification Report------------------------------------
              precision    recall  f1-score   support

           0       1.00      0.99      1.00     85312
           1       0.16      0.89      0.27       131

   micro avg       0.99      0.99      0.99     85443
   macro avg       0.58      0.94      0.63     85443
weighted avg       1.00      0.99      1.00     85443
```

The recall for this model further reduces to 0.89 while the precision increases to 0.16.

RANDOM FOREST:

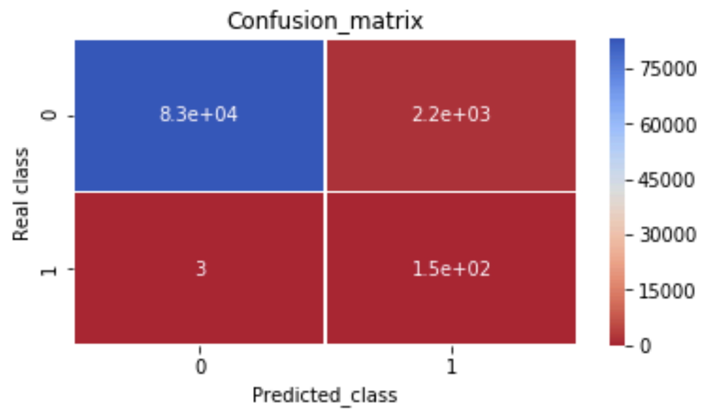
All the data-preparation step remains the same as above. The modeling algorithm is changed to random forest as follows:

```
for i in range(1,4):
   Undersample_data = undersample(i)

US_train_input,US_test_input,US_train_output,US_test_output=data_prepration(Unders
ample_data)
   train_input,test_input,train_output,test_output=data_prepration(data)
   #the partion for whole data
   print()
   clf=RandomForestClassifier(n_estimators=100)
   model(clf,US_train_input,test_input,US_train_output,test_output)
```

OUTPUT:

Ratio is 0.5:0.5

```
the recall for this model is : 0.9803921568627451
TP 150
TN 83119
FP 2171
FN 3
```

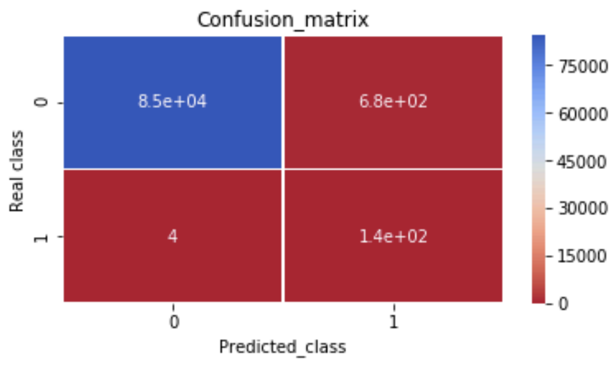


```
----------Classification Report-----------------------------------
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     85290
           1       0.06      0.98      0.12       153

   micro avg       0.97      0.97      0.97     85443
   macro avg       0.53      0.98      0.55     85443
weighted avg       1.00      0.97      0.99     85443
```

The recall and precision values are much better compared to the same data using logistic regression. The recall value is 0.98 and precision is 0.06.

Ratio of 0.66:0.33

```
the recall for this model is : 0.9722222222222222
TP 140
TN 84615
FP 684
FN 4
```

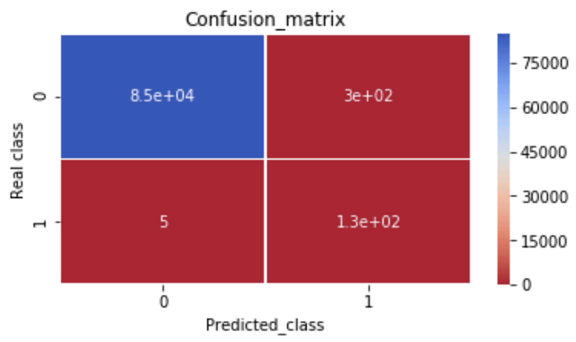


```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

           0       1.00      0.99      1.00     85299
           1       0.17      0.97      0.29       144

   micro avg       0.99      0.99      0.99     85443
   macro avg       0.58      0.98      0.64     85443
weighted avg       1.00      0.99      0.99     85443
```

The recall-value reduces to 0.97 and precision value increases to 0.17. The pattern followed is similar to logistic regression.

Ratio of 0.75:0.25

```
the recall for this model is : 0.9637681159420289
TP 133
TN 85005
FP 300
FN 5
```



```
----------Classification Report-----------------------------------
             precision    recall  f1-score   support

           0       1.00      1.00      1.00     85305
           1       0.31      0.96      0.47       138

   micro avg       1.00      1.00      1.00     85443
   macro avg       0.65      0.98      0.73     85443
weighted avg       1.00      1.00      1.00     85443
```

The recall has reduced to 0.96 and precision has increased to 0.31.

SUPPORT VECTOR MACHINE:

The data remains the same as above and the modeling algorithm is changed to support-vector machine. The following table illustrates the main results:

Ratio	Precision	Recall
0.5:0.5	0.03	0.90
0.66:0.33	0.07	0.91
0.75:0.25	0.08	0.87

GRADIENT BOOSTING MACHINE:

The data remains the same as above and the modeling algorithm is changed to gradient-boosting machine. The following table illustrates the main results:

Ratio	Precision	Recall
0.5:0.5	0.05	0.96
0.66:0.33	0.13	0.95
0.75:0.25	0.22	0.94

OVERSAMPLING:

A function was created for oversampling the data as follows:

```
 def oversample(k):
    train_input,test_input,train_output,test_output=data_prepration(data1)
    train_input["Class"]= train_output["Class"] # combining class with original data
    traindata = train_input.copy()
    normal_data = traindata[traindata["Class"]==0]
    print("length of normal data",len(normal_data))
    fraud_data = traindata[traindata["Class"]==1]
    print("length of fraud data",len(fraud_data))
    test_input['normAmount'] = (test_input['Amount']-
min(test_input['Amount']))/(max(test_input['Amount'])-min(test_input['Amount']))
    test_input.drop(["Time","Amount"],axis=1,inplace=True)
    #Over Sampling
    for i in range (100*k):
        normal_data= normal_data.append(fraud_data)
    os_data = normal_data.copy()
    print("Proportion of Normal data in oversampled data is
",len(os_data[os_data["Class"]==0])/len(os_data))
    print("Proportion of fraud data in oversampled data is
",len(os_data[os_data["Class"]==1])/len(os_data))
```

```
    os_data['normAmount'] = (os_data['Amount']-
min(os_data['Amount']))/(max(os_data['Amount'])-min(os_data['Amount']))
    os_data.drop(["Time","Amount"],axis=1,inplace=True)
    # now take all over sampled data as training and test it for test data
    os_data_X = os_data.ix[:,os_data.columns != "Class"]
    os_data_y = os_data.ix[:,os_data.columns == "Class"]
    return(os_data_X,os_data_y,test_input,test_output)
```

Here the training and test data are created. The training data is split into normal and fraudulent transactions. The fraudulent transactions are added to the normal transactions as many times as the number specified by 100*k, where k is an integer. Three data split ratios of (0.5:0.5, 0.75:0.25, 0.66:0.33) are being checked for.

LOGISTIC REGRESSION:

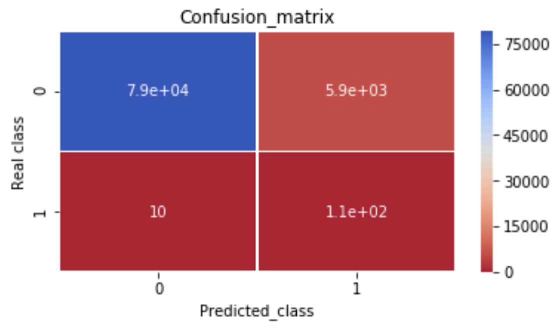
Logistic regression is first used to model the data. The following code illustrates the procedure. The oversampled training data is used for modeling and the entire test data is used for checking the accuracy of the model. The data-preparation function is used for generating the training and testing data using both the over-sampled and original data sets.

```
   os_input, os_output, test_input, test_output = oversample(k)
   clf=LogisticRegression()
   model(clf,os_input,test_input,os_output,test_output)
```

OUTPUT:

Ratio of 0.5:0.5

```
the recall for this model is : 0.9180327868852459
TP 112
TN 79375
FP 5946
FN 10
```

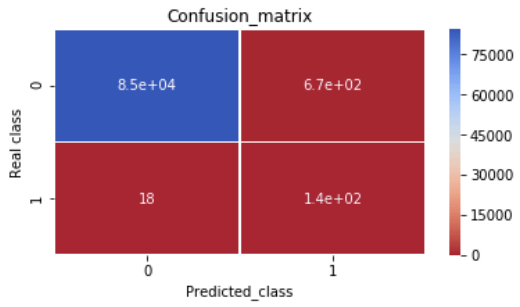


```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

           0       1.00      0.93      0.96     85321
           1       0.02      0.92      0.04       122

   micro avg       0.93      0.93      0.93     85443
   macro avg       0.51      0.92      0.50     85443
weighted avg       1.00      0.93      0.96     85443
```

The recall is this case is 0.92 and precision is 0.02.

Ratio of 0.66:0.33

```
TP 142
TN 84614
FP 669
FN 18
```

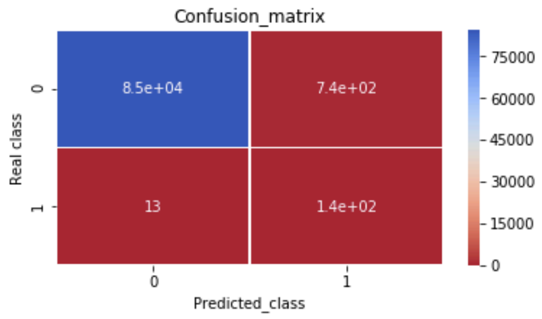


```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

           0       1.00      0.99      1.00     85283
           1       0.18      0.89      0.29       160

   micro avg       0.99      0.99      0.99     85443
   macro avg       0.59      0.94      0.64     85443
weighted avg       1.00      0.99      0.99     85443
```

The recall is 0.89 and precision is 0.18.

Ratio of 0.75:0.25

```
TP 143
TN 84550
FP 737
FN 13
```



```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

           0       1.00      0.99      1.00     85287
           1       0.16      0.92      0.28       156

   micro avg       0.99      0.99      0.99     85443
   macro avg       0.58      0.95      0.64     85443
weighted avg       1.00      0.99      0.99     85443
```

The precision is 0.16 and recall is 0.92.

RANDOM FOREST:

All the data-preparation step remains the same as above. The modeling algorithm is changed to random forest as follows:

```
for i in range(1,4):
  os_input, os_output,test_input, test_output = oversample(i)
  clf=RandomForestClassifier(n_estimators=100)
  model(clf,os_input,test_input,os_output,test_output)
```

OUTPUT:

Ratio	Precision	Recall
0.5:0.5	0.96	0.76
0.66:0.33	0.96	0.76
0.75:0.25	0.93	0.76

This is the highest precision, recall achieved so far. The recall though has come down a bit from the under-sampled result. However, the precision has improved a lot and almost reached 1.

The most important features are checked:

```
V14           0.180343
V10           0.155397
V12           0.097605
V17           0.088358
V4            0.083979
V16           0.070600
V11           0.064976
V3            0.047045
V7            0.021615
V27           0.018615
V18           0.018219
V2            0.016857
V6            0.014118
V9            0.013045
V1            0.012912
V21           0.010496
V19           0.009654
V8            0.009065
V13           0.008510
normAmount    0.007914
V15           0.007713
V23           0.007414
V20           0.006318
```

V25	0.005470
V26	0.005390
V5	0.005251
V28	0.005071
V22	0.004209
V24	0.003841

The variables V14, V10, V12 and V17, V4 are the most important and the normalized amount does not seem to be an important feature here.

The table having all the four modeling algorithms for the three ratios is given below:

Under-Sampling	**Logistic Regression**		**Random Forest**		**Support Vector Machine**		**Gradient Boosting Machine**	
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
0.5:0.5	**0.05**	**0.92**	**0.06**	**0.98**	**0.03**	**0.90**	**0.05**	**0.96**
0.66:0.33	**0.08**	**0.9**	**0.17**	**0.97**	**0.07**	**0.91**	**0.13**	**0.95**
0.75:0.25	**0.16**	**0.89**	**0.31**	**0.96**	**0.08**	**0.87**	**0.22**	**0.94**

Over-Sampling	**Logistic Regression**		**Random Forest**		**Gradient Boosting Machine**	
	Precision	**Recall**	**Precision**	**Recall**	**Precision**	**Recall**
0.5:0.5	**0.02**	**0.92**	**0.96**	**0.76**	**0.19**	**0.87**
0.66:0.33	**0.18**	**0.89**	**0.96**	**0.76**	**0.40**	**0.86**
0.75:0.25	**0.16**	**0.92**	**0.93**	**0.76**	**0.51**	**0.84**

CONCLUSION:

From the above tables, it is clear that Over-sampling is leading to a good-mixture of precision and recall values. Under-sampling is resulting is very good recall values but very poor precision values. Hence, it is better to go with over-sampling for overall better performance.

Among the modeling algorithms, Random Forest is performing best and giving a very good precision and recall values. When the sampling ratio of normal and fraud transaction is changed with more presence of fraudulent transactions the precision is reducing while there is not much change in the recall value.

The variable importance using the random-forest model is studied and it is found that V14, V10, V12, V17 and V4 are the top five important variables.