Credit Card Fraud Detection

In this project you will predict fraudulent credit card transactions with the help of Machine learning models. Please import the following libraries to get started.

```
import pandas
import numpy
import datetime

import warnings
import missingno as msno

pandas.set_option('display.max_columns', 500)
warnings.simplefilter('ignore')

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import ListedColormap

from sklearn import preprocessing
from sklearn.decomposition import PCA
from sklearn.decomposition import IncrementalPCA

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from statsmodels.stats.outliers_influence import
variance_inflation_factor
from sklearn.metrics import precision_recall_curve
from sklearn.feature_selection import RFE
from sklearn import metrics

import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.linear_model import BayesianRidge
from sklearn.impute import KNNImputer

from collections import Counter
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import ADASYN, RandomOverSampler
```

Utility Functions

```
def format_date(x):
    if str(x) == 'nan':
```

```
        return x
    else:
        return datetime.datetime.strptime(x,"%m/%d/%Y")

def imputation(data, column, value):
    data.loc[data[column].isnull()==True,column] = value
    return data


def missing_value_percentage(data):
    missing_value_summary =
pandas.DataFrame(data.isnull().sum()*100/data.shape[0])
    missing_value_summary.columns = ['Invalid Data %']
    missing_value_summary =
missing_value_summary.loc[missing_value_summary['Invalid Data
%']>0].sort_values('Invalid Data %', ascending=False)
    return missing_value_summary

def create_outlier_df(data, var_list):

outlier_df=pandas.DataFrame(columns=['ColumnName','OutlierCount','Outl
ier%'])
    for var in var_list:
        try:
            Q1 = data[var].quantile(0.25)
            Q3 = data[var].quantile(0.75)
            IQR = Q3 - Q1
            outlier_count=((data[var] < (Q1 - (1.5 * IQR))) |
(data[var] > (Q3 + (1.5 * IQR)))).sum()
            new_row = {
                        'ColumnName':var,
'OutlierCount':outlier_count,
                        'Outlier
%':round(outlier_count*100/data[var].shape[0],2)
                        }
            outlier_df=outlier_df.append(new_row,ignore_index=True)
        except TypeError:
            print('Error with column '+var)
    return outlier_df

def cap_outlier(data, var_list):
    for col in var_list:
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1
        data[col][data[col] <= (Q1 - 1.5 * IQR)] = (Q1 - 1.5 * IQR)
        data[col][data[col] >= (Q3 + 1.5 * IQR)] = (Q3 + 1.5 * IQR)
    return data
```

```
def vif_ranks(data, features, row_count):
    vif = pandas.DataFrame()
    vif['Features'] = data[features].columns
    vif['VIF'] = [variance_inflation_factor(X_train[features].values,
i) for i in range(X_train[features].shape[1])]
    vif['VIF'] = round(vif['VIF'], 2)
    vif = vif.sort_values(by = "VIF", ascending = False)
    return vif.head(row_count)

def create_correlation_df(data,var_list,threshold):
    listi=[]
    listj=[]
    data=data[var_list]
    resultDf=pandas.DataFrame(columns=['Feature 1','Feature
2','Correlation Value'])
    corrDf=data.corr()
    for i in corrDf.columns:
        for j in corrDf.columns:
            if i==j:
                break
            if (corrDf.loc[i,j] >=threshold) and (str(i)!=str(j)):

                new_row = {
                        'Feature 1':str(i), 'Feature 2':str(j),
                        'Correlation Value':round(corrDf.loc[i,j],2)
                        }
                resultDf=resultDf.append(new_row,ignore_index=True)
    return resultDf
```

Performance Metrics Functions

```
def predict_summarize(predicted, actual, threshold, plot_roc_=False):
    y_pred_final = pandas.DataFrame({'Converted':actual,
'Converted_Probability':predicted})
    y_pred_final['CustID'] = range(len(predicted))
    y_pred_final['predicted'] =
y_pred_final['Converted_Probability'].map(lambda x: 1 if x > threshold
else 0)

    # Confusion matrix
    confusion = metrics.confusion_matrix(y_pred_final['Converted'],
y_pred_final['predicted'] )

    TP = confusion[1,1] # true positive
    TN = confusion[0,0] # true negatives
    FP = confusion[0,1] # false positives
    FN = confusion[1,0] # false negatives

    sensitivity = TP / float(TP+FN)
    specificity = TN / float(TN+FP)
```

```
    false_positive_rate = FP/ float(TN+FP)
    precision = TP / float(TP + FP)
    recall = TP / float(TP + FN)

    confusion = pandas.DataFrame(confusion)
    confusion.columns = ['predicted_no','predicted_yes']
    confusion['ind'] = ['actual_no','actual_yes']
    confusion = confusion.set_index('ind')

    print('Accuracy =
',metrics.accuracy_score(y_pred_final['Converted'],
y_pred_final.predicted))
    print('Sensitivity = ',sensitivity)
    print('Specificity = ',specificity)
    print('False Positive Rate = ',false_positive_rate)
    print('\nPrecision = ',precision)
    print('Recall = ',recall)

    if plot_roc_:
        plot_roc(y_pred_final['Converted'],
y_pred_final['Converted_Probability'])


    return confusion

def plot_roc( actual, probs ):
    print('Plotting')
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate =
False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return None

def plot_precision_recall_curve(predicted, actual):
    y_pred_final = pandas.DataFrame({'Converted':actual,
'Converted_Probability':predicted})
    y_pred_final['CustID'] = range(len(actual))

    p, r, thresholds =
```

```
precision_recall_curve(y_pred_final['Converted'],
y_pred_final['Converted_Probability'])
    plt.figure(figsize=(10,5))
    plt.plot(thresholds, p[:-1], "g-",label='precision')
    plt.plot(thresholds, r[:-1], "r-",label='recall')
    plt.xlabel('Thresholds')
    plt.title('Precision Recall Curve')
    plt.legend()
    plt.show()

def plot_feature_importance(features, importances):
    feature_importance = pandas.DataFrame(importances, features)
    feature_importance['abs_value'] = abs(feature_importance[0])
    feature_importance =
feature_importance.sort_values('abs_value',ascending=False)
    plt.figure(figsize=(10,10))

sns.barplot(feature_importance[0],feature_importance.index,palette='hu
sl')
    plt.title('Feature Importances')
```

Data Preparation

```
df = pandas.read_csv('..//data//creditcard.csv')
df.head()
```

```
   Time        V1        V2        V3        V4        V5        V6
V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388
0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -
0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499
0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203
0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921
0.592941

         V8        V9       V10       V11       V12       V13
V14  \
0  0.098698  0.363787  0.090794 -0.551600 -0.617801 -0.991390 -
0.311169
1  0.085102 -0.255425 -0.166974  1.612727  1.065235  0.489095 -
0.143772
2  0.247676 -1.514654  0.207643  0.624501  0.066084  0.717293 -
0.165946
3  0.377436 -1.387024 -0.054952 -0.226487  0.178228  0.507757 -
0.287924
4 -0.270533  0.817739  0.753074 -0.822843  0.538196  1.345852 -
1.119670
```

```
        V15       V16       V17       V18       V19       V20       
V21  \
0  1.468177 -0.470401  0.207971  0.025791  0.403993  0.251412 -
0.018307   
1  0.635558  0.463917 -0.114805 -0.183361 -0.145783 -0.069083 -
0.225775   
2  2.345865 -2.890083  1.109969 -0.121359 -2.261857  0.524980  
0.247998   
3 -0.631418 -1.059647 -0.684093  1.965775 -1.232622 -0.208038 -
0.108300   
4  0.175121 -0.451449 -0.237033 -0.038195  0.803487  0.408542 -
0.009431   

        V22       V23       V24       V25       V26       V27       
V28  \
0  0.277838 -0.110474  0.066928  0.128539 -0.189115  0.133558 -
0.021053   
1 -0.638672  0.101288 -0.339846  0.167170  0.125895 -0.008983  
0.014724   
2  0.771679  0.909412 -0.689281 -0.327642 -0.139097 -0.055353 -
0.059752   
3  0.005274 -0.190321 -1.175575  0.647376 -0.221929  0.062723  
0.061458   
4  0.798278 -0.137458  0.141267 -0.206010  0.502292  0.219422  
0.215153   

   Amount  Class  
0  149.62      0  
1    2.69      0  
2  378.66      0  
3  123.50      0  
4   69.99      0  
```

```
#observe the different feature type present in the data
df.dtypes
```

```
Time      float64
V1        float64
V2        float64
V3        float64
V4        float64
V5        float64
V6        float64
V7        float64
V8        float64
V9        float64
V10       float64
V11       float64
V12       float64
```

```
V13        float64
V14        float64
V15        float64
V16        float64
V17        float64
V18        float64
V19        float64
V20        float64
V21        float64
V22        float64
V23        float64
V24        float64
V25        float64
V26        float64
V27        float64
V28        float64
Amount     float64
Class        int64
dtype: object
```

```
df.describe()
```

```
                Time            V1            V2            V3
V4  \
count  284807.000000  2.848070e+05  2.848070e+05  2.848070e+05
2.848070e+05
mean    94813.859575  3.919560e-15  5.688174e-16 -8.769071e-15
2.782312e-15
std     47488.145955  1.958696e+00  1.651309e+00  1.516255e+00
1.415869e+00
min         0.000000 -5.640751e+01 -7.271573e+01 -4.832559e+01 -
5.683171e+00
25%     54201.500000 -9.203734e-01 -5.985499e-01 -8.903648e-01 -
8.486401e-01
50%     84692.000000  1.810880e-02  6.548556e-02  1.798463e-01 -
1.984653e-02
75%    139320.500000  1.315642e+00  8.037239e-01  1.027196e+00
7.433413e-01
max    172792.000000  2.454930e+00  2.205773e+01  9.382558e+00
1.687534e+01

                 V5            V6            V7            V8
V9  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
2.848070e+05
mean  -1.552563e-15  2.010663e-15 -1.694249e-15 -1.927028e-16 -
3.137024e-15
std    1.380247e+00  1.332271e+00  1.237094e+00  1.194353e+00
1.098632e+00
min   -1.137433e+02 -2.616051e+01 -4.355724e+01 -7.321672e+01 -
```

```
1.343407e+01
25%   -6.915971e-01 -7.682956e-01 -5.540759e-01 -2.086297e-01 -
6.430976e-01
50%   -5.433583e-02 -2.741871e-01  4.010308e-02  2.235804e-02 -
5.142873e-02
75%    6.119264e-01  3.985649e-01  5.704361e-01  3.273459e-01  
5.971390e-01
max    3.480167e+01  7.330163e+01  1.205895e+02  2.000721e+01  
1.559499e+01

                V10           V11           V12           V13  
V14  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  
2.848070e+05
mean   1.768627e-15  9.170318e-16 -1.810658e-15  1.693438e-15  
1.479045e-15
std    1.088850e+00  1.020713e+00  9.992014e-01  9.952742e-01  
9.585956e-01
min   -2.458826e+01 -4.797473e+00 -1.868371e+01 -5.791881e+00 -
1.921433e+01
25%   -5.354257e-01 -7.624942e-01 -4.055715e-01 -6.485393e-01 -
4.255740e-01
50%   -9.291738e-02 -3.275735e-02  1.400326e-01 -1.356806e-02  
5.060132e-02
75%    4.539234e-01  7.395934e-01  6.182380e-01  6.625050e-01  
4.931498e-01
max    2.374514e+01  1.201891e+01  7.848392e+00  7.126883e+00  
1.052677e+01

                V15           V16           V17           V18  
V19  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05  
2.848070e+05
mean   3.482336e-15  1.392007e-15 -7.528491e-16  4.328772e-16  
9.049732e-16
std    9.153160e-01  8.762529e-01  8.493371e-01  8.381762e-01  
8.140405e-01
min   -4.498945e+00 -1.412985e+01 -2.516280e+01 -9.498746e+00 -
7.213527e+00
25%   -5.828843e-01 -4.680368e-01 -4.837483e-01 -4.988498e-01 -
4.562989e-01
50%    4.807155e-02  6.641332e-02 -6.567575e-02 -3.636312e-03  
3.734823e-03
75%    6.488208e-01  5.232963e-01  3.996750e-01  5.008067e-01  
4.589494e-01
max    8.877742e+00  1.731511e+01  9.253526e+00  5.041069e+00  
5.591971e+00

                V20           V21           V22           V23  
V24  \
```

```
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
2.848070e+05
mean   5.085503e-16  1.537294e-16  7.959909e-16  5.367590e-16
4.458112e-15
std    7.709250e-01  7.345240e-01  7.257016e-01  6.244603e-01
6.056471e-01
min   -5.449772e+01 -3.483038e+01 -1.093314e+01 -4.480774e+01 -
2.836627e+00
25%   -2.117214e-01 -2.283949e-01 -5.423504e-01 -1.618463e-01 -
3.545861e-01
50%   -6.248109e-02 -2.945017e-02  6.781943e-03 -1.119293e-02
4.097606e-02
75%    1.330408e-01  1.863772e-01  5.285536e-01  1.476421e-01
4.395266e-01
max    3.942090e+01  2.720284e+01  1.050309e+01  2.252841e+01
4.584549e+00

                V25           V26           V27           V28
Amount  \
count  2.848070e+05  2.848070e+05  2.848070e+05  2.848070e+05
284807.000000
mean   1.453003e-15  1.699104e-15 -3.660161e-16 -1.206049e-16
88.349619
std    5.212781e-01  4.822270e-01  4.036325e-01  3.300833e-01
250.120109
min   -1.029540e+01 -2.604551e+00 -2.256568e+01 -1.543008e+01
0.000000
25%   -3.171451e-01 -3.269839e-01 -7.083953e-02 -5.295979e-02
5.600000
50%    1.659350e-02 -5.213911e-02  1.342146e-03  1.124383e-02
22.000000
75%    3.507156e-01  2.409522e-01  9.104512e-02  7.827995e-02
77.165000
max    7.519589e+00  3.517346e+00  3.161220e+01  3.384781e+01
25691.160000

               Class
count  284807.000000
mean        0.001727
std         0.041527
min         0.000000
25%         0.000000
50%         0.000000
75%         0.000000
max         1.000000
```

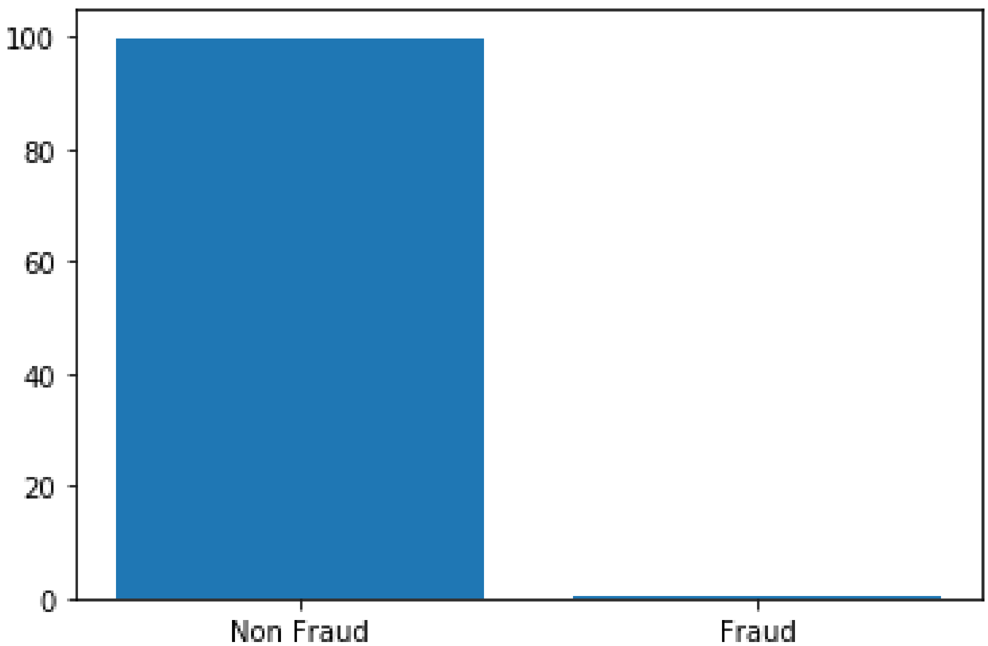
Here we will observe the distribution of our classes.

We also see that there are no missing values and hence no missing value treatment is required.

```python
classes=df['Class'].value_counts()
normal_share=classes[0]/df['Class'].count()*100
fraud_share=classes[1]/df['Class'].count()*100
```

```python
# Create a bar plot for the number and percentage of fraudulent vs
non-fraudulent transcations
plt.bar(['Non Fraud', 'Fraud'],
[normal_share/(normal_share+fraud_share)*100,
fraud_share/(normal_share+fraud_share)*100])
```

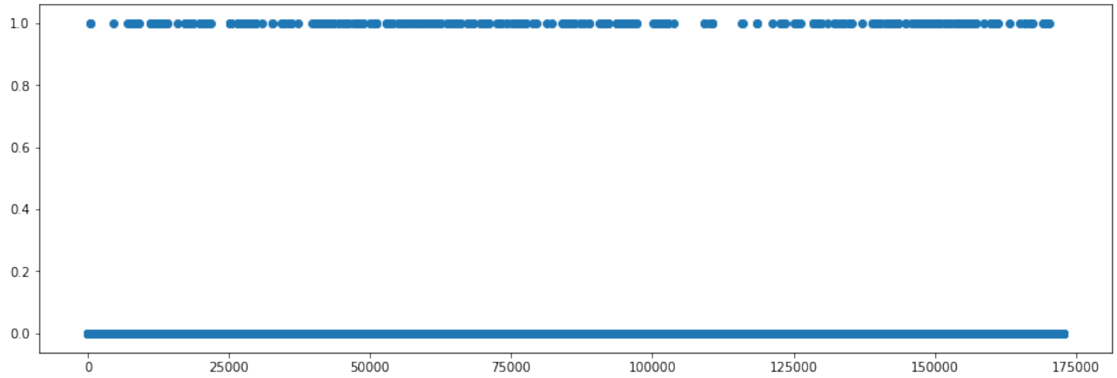
```
<BarContainer object of 2 artists>
```



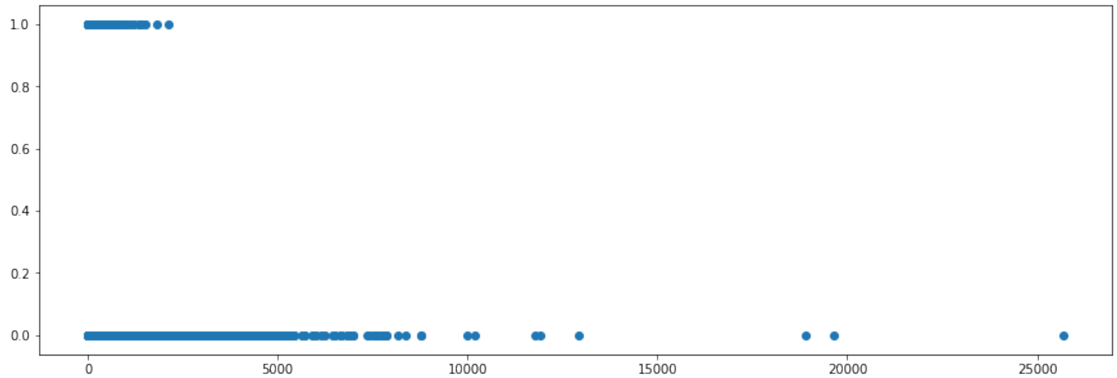
Observation:

The percentage of fraud transactions is as low as .17 %

```python
# Create a scatter plot to observe the distribution of classes with
time
plt.figure(figsize=(15,5))
plt.scatter(df['Time'], df['Class'])
plt.show()
```




```
# Create a scatter plot to observe the distribution of classes with
Amount
plt.figure(figsize=(15,5))
plt.scatter(df['Amount'], df['Class'])
plt.show()
```



We can clearly observe that all the fraudulant transactions are of relatively lower amounts.

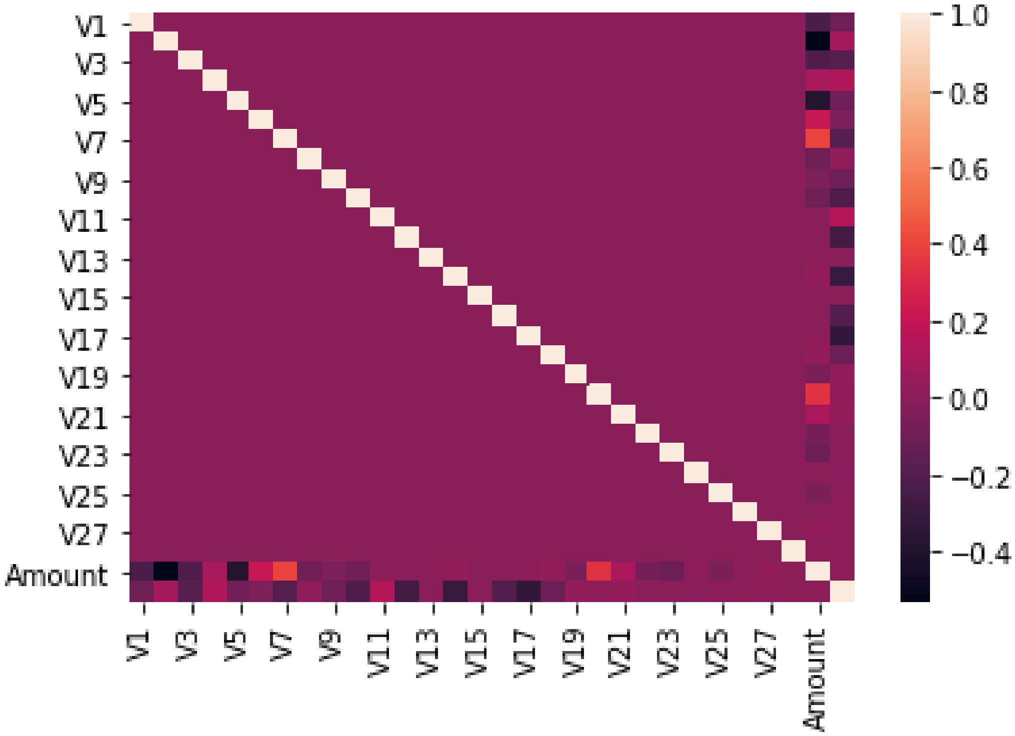
```
# Drop unnecessary columns
df = df.drop('Time', axis=1)
```

Observation:

The time variable can be dropped as it does not have any relavence in the occurance of fraud transactions

```
sns.heatmap(df.corr())
```

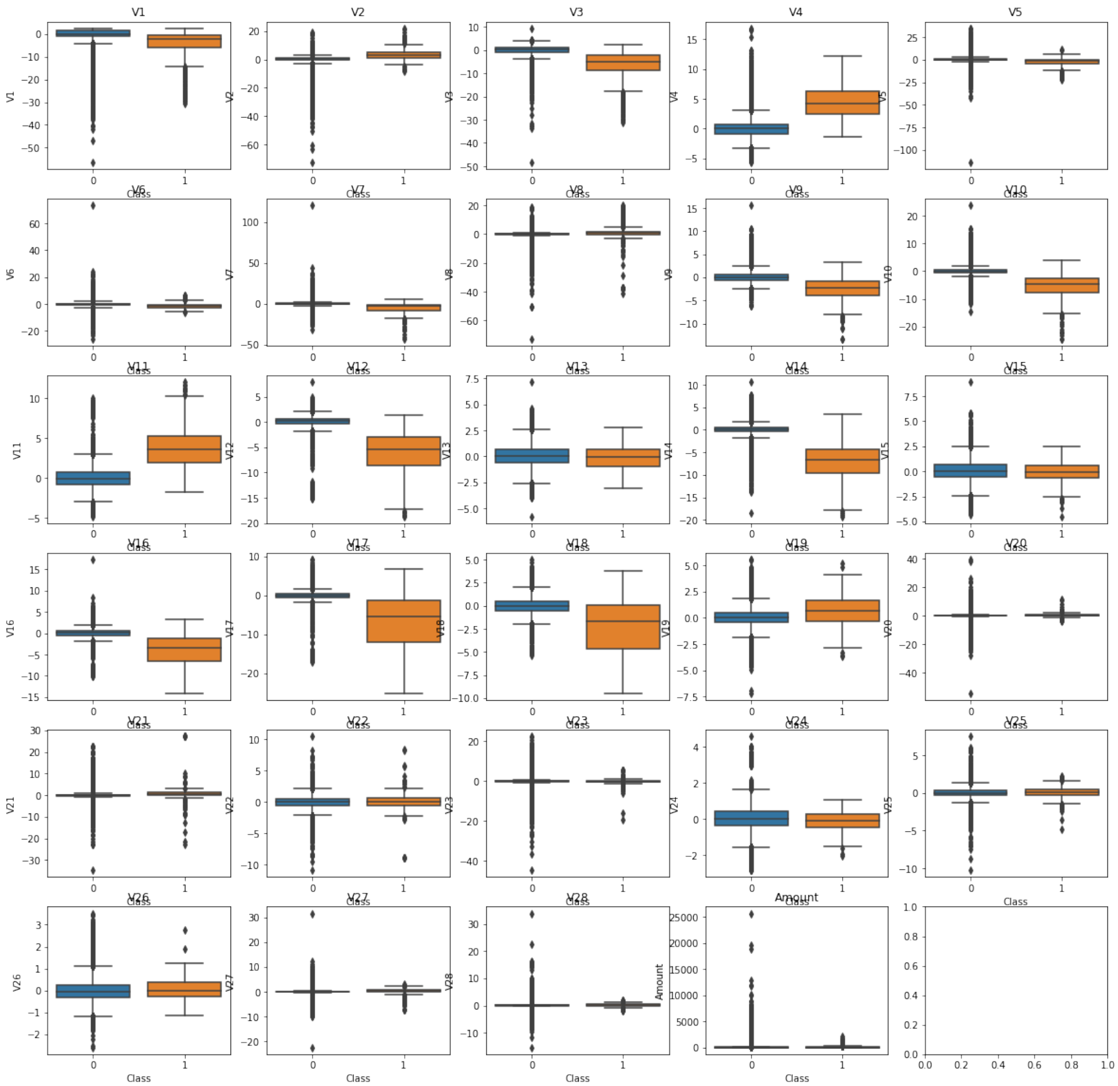
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f57873e4690>
```



Observation:

No columns are correlated to one other, there we need not eliminate the features now.

```
column_list = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9',
'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19',
'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28',
'Amount']

plt.subplots(6,5, figsize=(20,20))
index=1
for column in column_list:
    plt.subplot(6,5, index)
    sns.boxplot(y=df[column], x=df['Class'])
    plt.title(column)
    index += 1
plt.show()
```



Observation:

The columns do have a lot of outliers but in the case of mining out fraud transactions we can retain the outlier values as it.

Splitting the data into train & test data

Feature Analysis - Transformation

```
y = df.pop('Class') #class variable
X = df

X_train,X_test,y_train,y_test = train_test_split(X, y, test_size=0.3,
random_state=100)

print(numpy.sum(y))
print(numpy.sum(y_train))
print(numpy.sum(y_test))
```

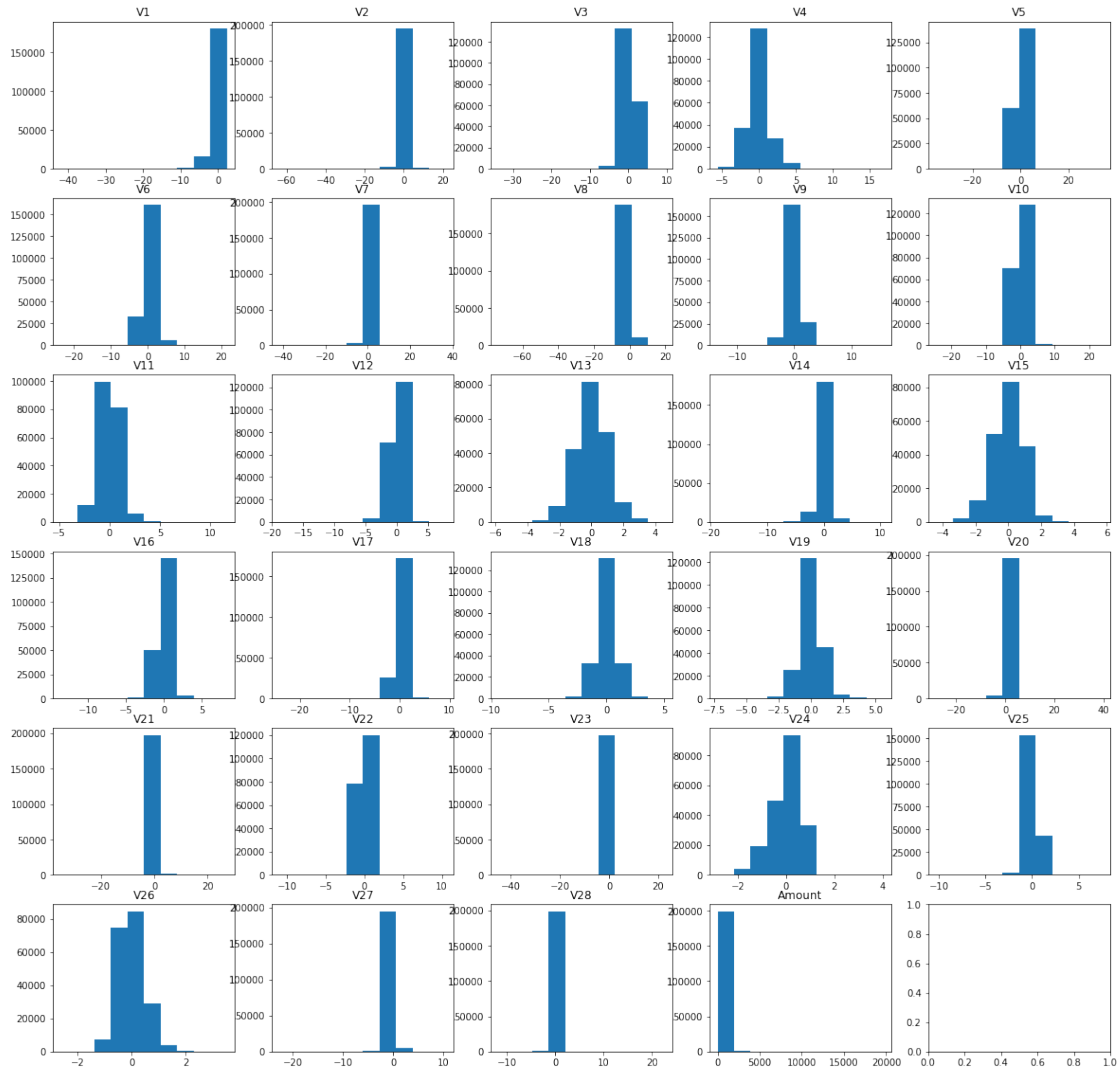
```
print('Train Fraud Rate', numpy.sum(y_train)/len(y_train))
print('Test Fraud Rate', numpy.sum(y_test)/len(y_test))
```

```
492
350
142
Train Fraud Rate 0.001755582753155033
Test Fraud Rate 0.001661926664559999
```

Observation:

The train and test sets both have similar event rates.

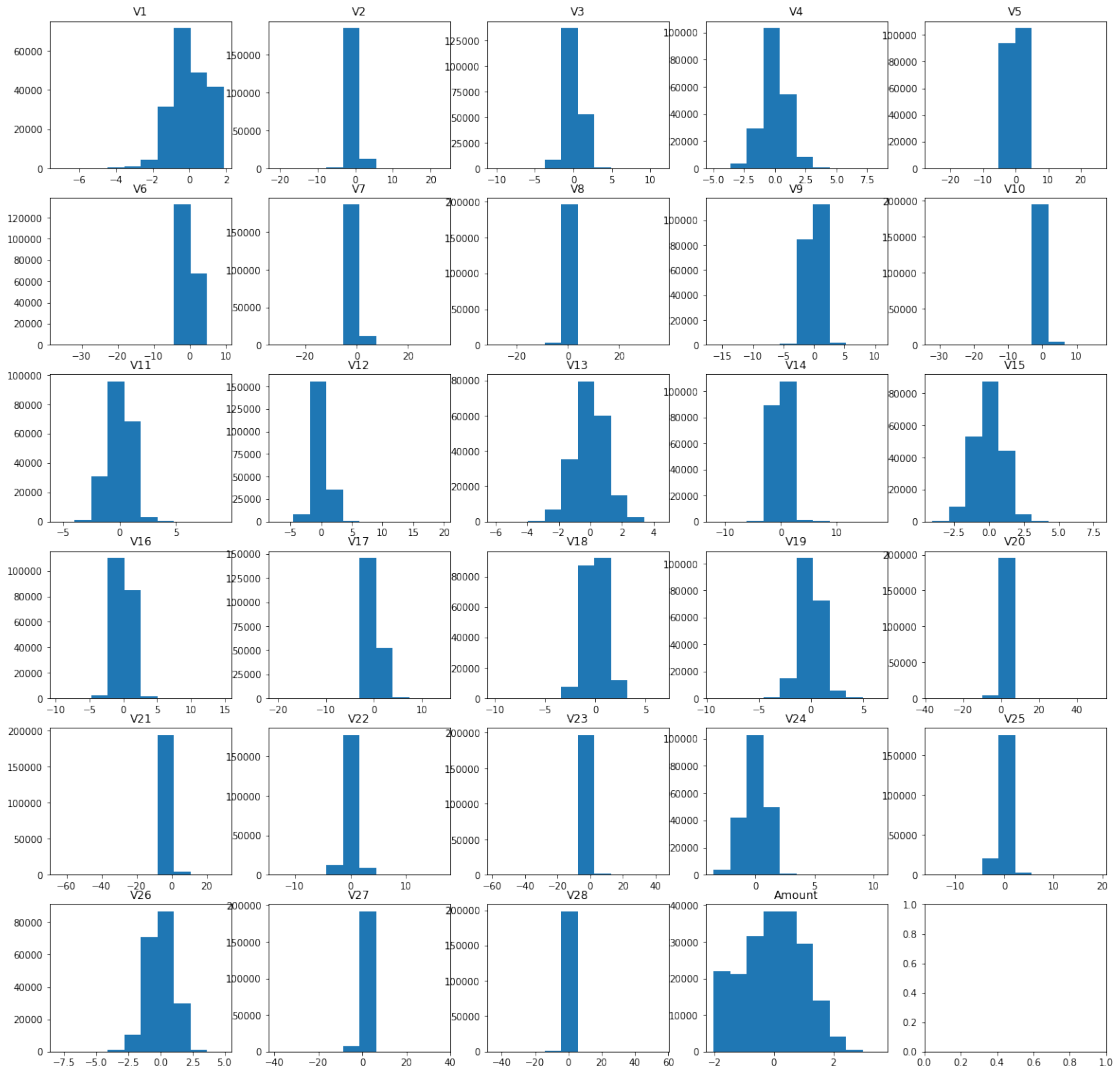
```
# plot the histogram of a variable from the dataset to see the
skewness
plt.subplots(6,5, figsize=(20,20))
index=1
for column in column_list:
    plt.subplot(6,5, index)
    plt.hist(X_train[column])
    plt.title(column)
    index += 1
plt.show()
```



Most of the above variables are skewed. So we can transform them to make them gaussian

```
# - Apply : preprocessing.PowerTransformer(copy=False) to fit &
transform the train & test data
pt = preprocessing.PowerTransformer()
X_train[column_list] = pt.fit_transform(X_train[column_list])
X_test[column_list] = pt.transform(X_test[column_list])

scaler = preprocessing.StandardScaler()
X_train[X_train.columns] = scaler.fit_transform(X_train)
X_test[X_train.columns] = scaler.transform(X_test[X_train.columns])

# plot the histogram of a variable from the dataset again to see the
result
plt.subplots(6,5, figsize=(20,20))
index=1
for column in column_list:
  plt.subplot(6,5, index)
  plt.hist(X_train[column])
```

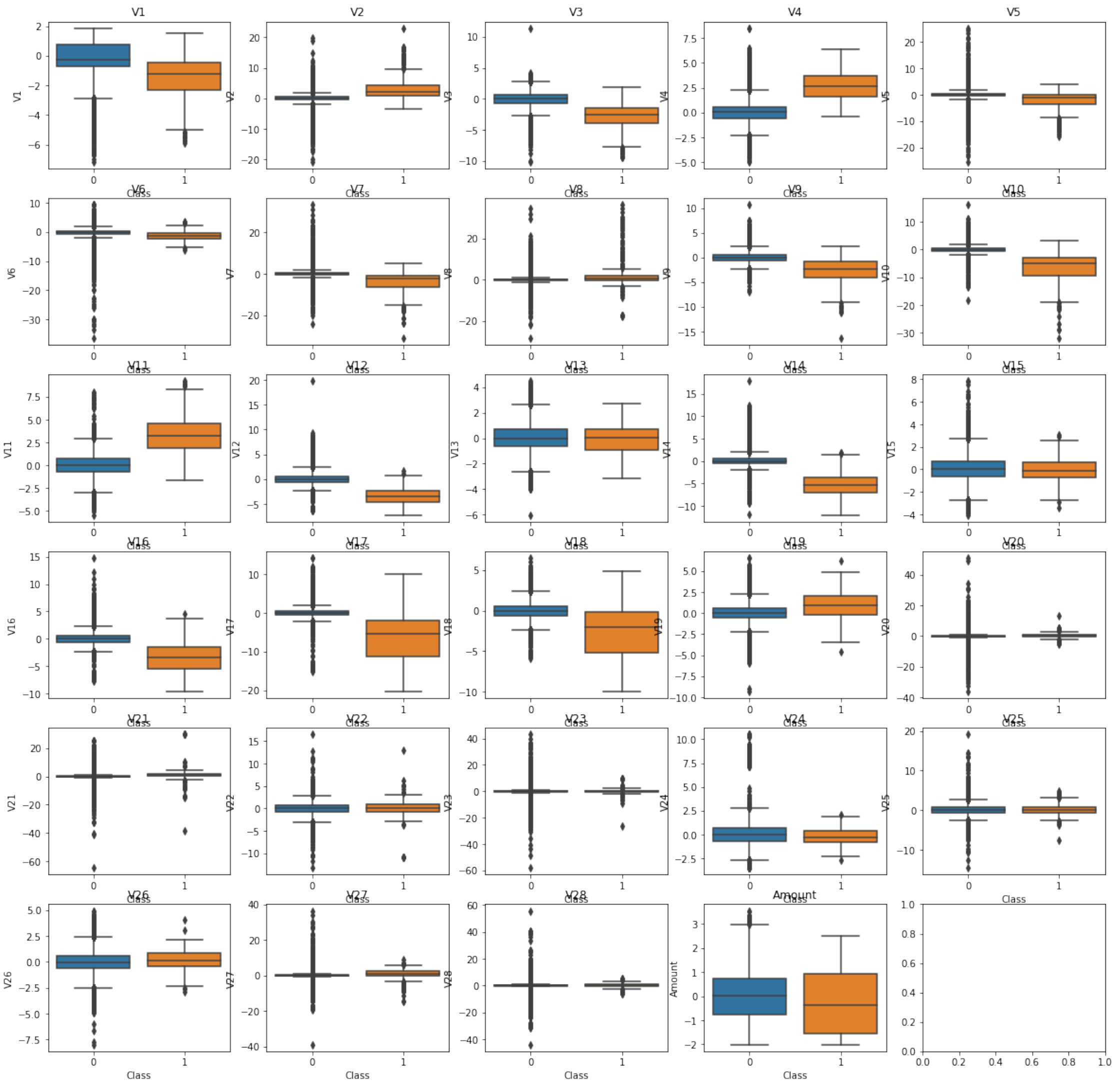
```
    plt.title(column)
    index += 1
plt.show()
```



Observation:

Few columns like amount column have undergone the transformation and have a different distribution now. The transformation was fit on train data and the test data was only transformed.

```
plt.subplots(6,5, figsize=(20,20))
index=1
for column in column_list:
    plt.subplot(6,5, index)
    sns.boxplot(y=X_train[column], x=y_train)
    plt.title(column)
    index += 1
plt.show()
```



Eliminate Insignificant Variables Using VIF

Though we have PCA transformed data, the objective still remains to be able to extract the important features impacting the churn. We will thus work towards this by eliminating the redundant variables.

```
features_set_1 = X_train.columns
vif_ranks(X_train, features_set_1, 10)
```

```
   Features   VIF
28   Amount  1.72
1        V2  1.60
0        V1  1.51
2        V3  1.28
4        V5  1.21
6        V7  1.10
7        V8  1.10
11      V12  1.09
```

```
5        V6  1.06
3        V4  1.04
```

The variables have VIF < 3. Therefore, we need not eliminate the variables at this stage.

Model Building

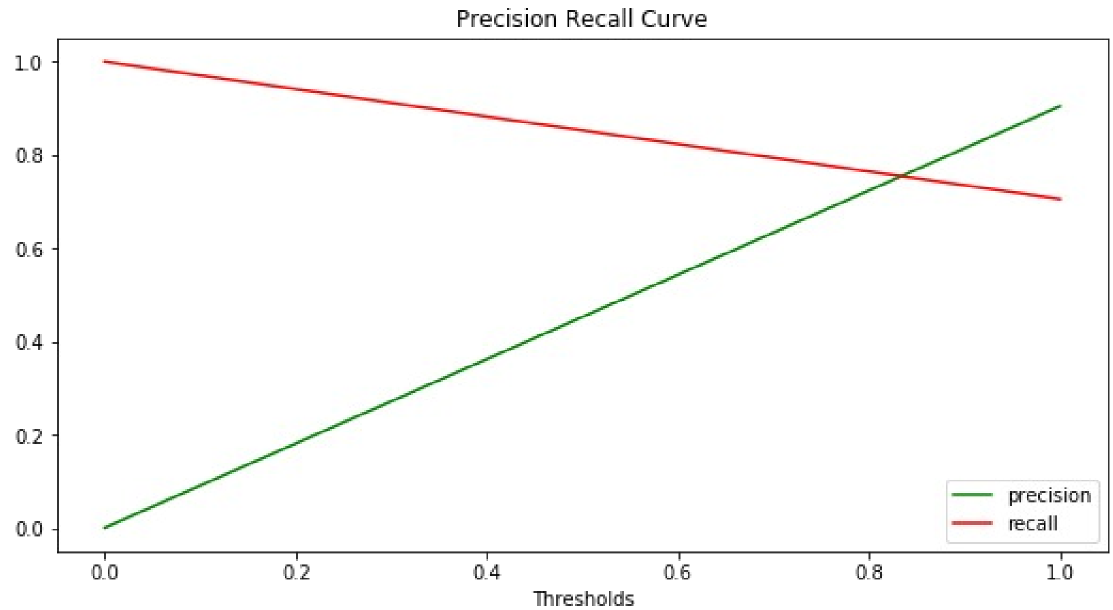
- Build different models on the imbalanced dataset and see the result

```
kf = KFold(n_splits=5, shuffle=False)
```

We can use kfold cross validation

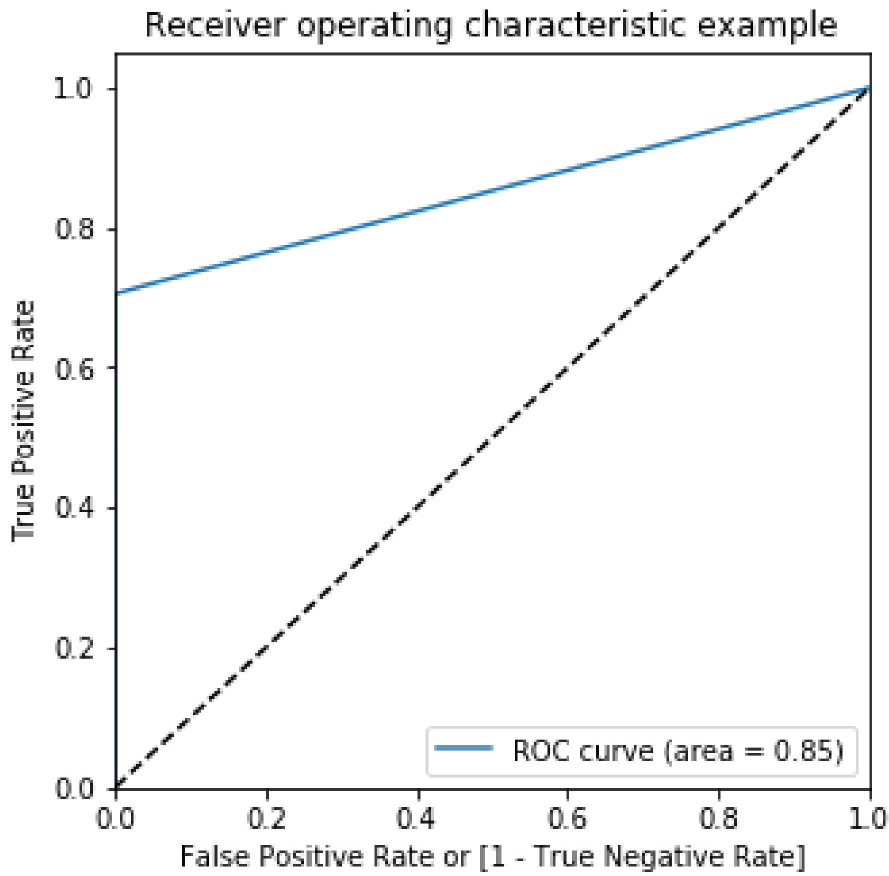
Logistic Regression

```
'''
grid_params = {
    "C":numpy.logspace(-3, 3,7),
    "penalty":["l1", "l2"]
}
logreg = LogisticRegression()
logreg_cv = GridSearchCV(logreg, grid_params, cv=kf)
logreg_cv.fit(X_train, y_train)
print("tuned hpyerparameters :(best parameters) ",
logreg_cv.best_params_)
print("accuracy :", logreg_cv.best_score_)
'''

'\ngrid_params = {\n    "C":numpy.logspace(-3, 3,7), \n    "penalty":
["l1", "l2"]\n}\nlogreg = LogisticRegression()\nlogreg_cv =
GridSearchCV(logreg, grid_params, cv=kf)\nlogreg_cv.fit(X_train,
y_train)\nprint("tuned hpyerparameters :(best parameters) ",
logreg_cv.best_params_)\nprint("accuracy :", logreg_cv.best_score_)\n'

logreg = LogisticRegression(C=1, penalty='l2')

logreg.fit(X_train, y_train)

LogisticRegression(C=1, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                   warm_start=False)

plot_precision_recall_curve(logreg.predict(X_train), y_train)
```




```
predict_summarize(logreg.predict(X_train), y_train, 0.8, True)
```

```
Accuracy =  0.9993529423566943
Sensitivity =  0.7057142857142857
Specificity =  0.9998693559247088
False Positive Rate =  0.00013064407529118555

Precision =  0.9047619047619048
Recall =  0.7057142857142857
Plotting
```




```
              predicted_no  predicted_yes
ind
actual_no           198988             26
actual_yes             103            247
```

Observation:

This has a low recall value. We can continue with other models & recursive feature eliminations.

Logistic Regression - Iterations

```
rfe_feature_set_1=X_train.columns

features=rfe_feature_set_1

X_train_sm = sm.add_constant(X_train[features])
X_test_sm = sm.add_constant(X_test[features])
logm2 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
model = logm2.fit()
model.summary()

<class 'statsmodels.iolib.summary.Summary'>
"""
```

```
                 Generalized Linear Model Regression Results

======================================================================
========
Dep. Variable:                  Class   No. Observations:
199364
Model:                            GLM   Df Residuals:
199334
Model Family:                Binomial   Df Model:
29
Link Function:                  logit   Scale:
1.0000
Method:                          IRLS   Log-Likelihood:
-672.87
Date:                Sun, 20 Sep 2020   Deviance:
1345.7
Time:                        20:21:48   Pearson chi2:
4.99e+05
No. Iterations:                    12

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
const         -8.9200      0.183    -48.667      0.000      -9.279
-8.561
V1            -0.0667      0.098     -0.683      0.494      -0.258
0.125
V2            -0.2710      0.083     -3.248      0.001      -0.435
-0.107
V3            -0.4474      0.099     -4.499      0.000      -0.642
-0.253
V4             0.8344      0.096      8.660      0.000       0.646
1.023
V5            -0.0743      0.081     -0.912      0.362      -0.234
0.085
V6             0.1365      0.094      1.447      0.148      -0.048
0.321
V7            -0.1907      0.070     -2.719      0.007      -0.328
-0.053
V8            -0.2512      0.059     -4.279      0.000      -0.366
-0.136
V9            -0.2690      0.099     -2.723      0.006      -0.463
-0.075
V10           -0.1099      0.094     -1.174      0.241      -0.294
0.074
```

```
V11            0.0054      0.097      0.056      0.955      -0.185
0.196
V12           -0.7616      0.121     -6.318      0.000      -0.998
-0.525
V13           -0.2510      0.096     -2.617      0.009      -0.439
-0.063
V14           -0.8387      0.083    -10.045      0.000      -1.002
-0.675
V15           -0.2010      0.096     -2.093      0.036      -0.389
-0.013
V16           -0.3660      0.096     -3.827      0.000      -0.553
-0.179
V17            0.0084      0.061      0.138      0.890      -0.111
0.128
V18            0.1215      0.101      1.200      0.230      -0.077
0.320
V19           -0.1316      0.087     -1.520      0.128      -0.301
0.038
V20           -0.0761      0.057     -1.327      0.184      -0.188
0.036
V21            0.2008      0.070      2.879      0.004       0.064
0.338
V22            0.3241      0.112      2.889      0.004       0.104
0.544
V23           -0.0717      0.043     -1.658      0.097      -0.157
0.013
V24            0.0445      0.105      0.422      0.673      -0.162
0.251
V25            0.1090      0.084      1.297      0.195      -0.056
0.274
V26           -0.0680      0.117     -0.582      0.561      -0.297
0.161
V27           -0.0711      0.055     -1.301      0.193      -0.178
0.036
V28           -0.0580      0.033     -1.759      0.079      -0.123
0.007
Amount        -0.0197      0.106     -0.187      0.852      -0.227
0.188
======================================================================
========
"""
```

Observation:

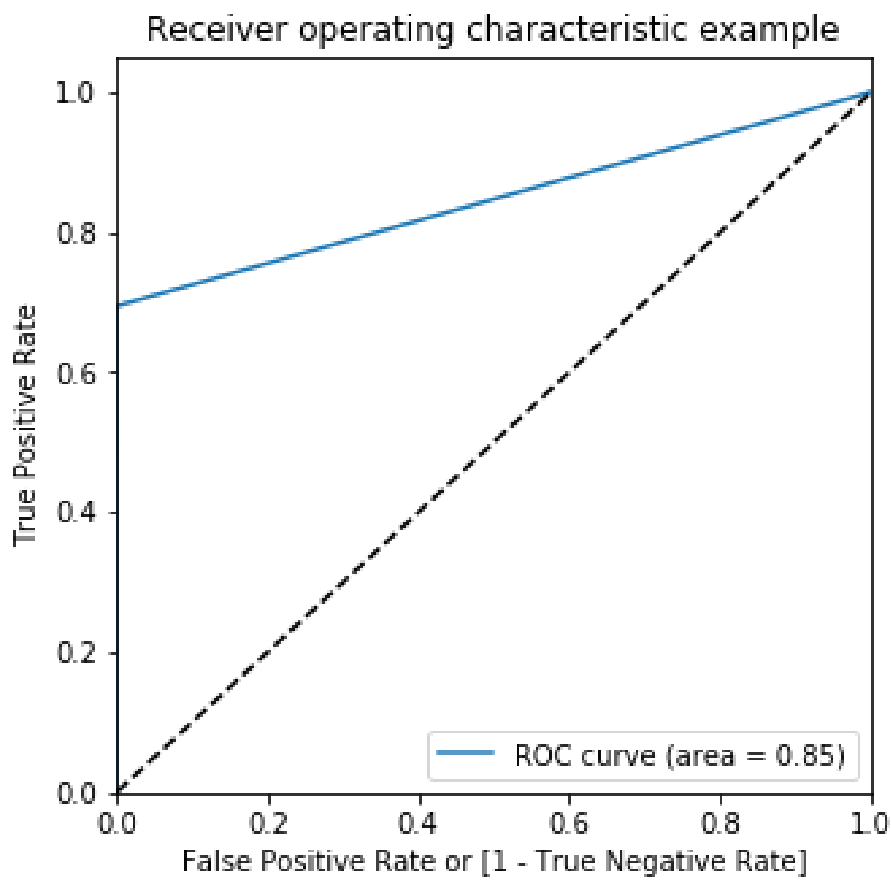
We can iteratively eliminate the features with p > 0.05. The cell below was not executed at once, the set of features to be eliminated was done repeated basis until all vairables had p < 0.05

```
# removing the features with p-value >0.05, iteratively, one at a time
features=list(set(rfe_feature_set_1)-
set(['V11','V17','Amount','V24','V26','V1','V5','V18',
```

```
'V23','V25','V20','V28','V10','V27','V6','V19']))

X_train_sm = sm.add_constant(X_train[features])
X_test_sm = sm.add_constant(X_test[features])
logm2 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
model = logm2.fit()
model.summary()
```

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                 Generalized Linear Model Regression Results

======================================================================
========
Dep. Variable:                  Class   No. Observations:
199364
Model:                            GLM   Df Residuals:
199350
Model Family:                Binomial   Df Model:
13
Link Function:                  logit   Scale:
1.0000
Method:                          IRLS   Log-Likelihood:
-680.66
Date:                Sun, 20 Sep 2020   Deviance:
1361.3
Time:                        20:21:49   Pearson chi2:
5.79e+05
No. Iterations:                    12

Covariance Type:            nonrobust

======================================================================
========
                 coef    std err          z      P>|z|      [0.025
0.975]
----------------------------------------------------------------------
--------
const         -8.8567      0.162    -54.691      0.000      -9.174
-8.539
V15           -0.2422      0.092     -2.629      0.009      -0.423
-0.062
V2            -0.1791      0.062     -2.910      0.004      -0.300
-0.058
V16           -0.3186      0.065     -4.891      0.000      -0.446
-0.191
V3            -0.4383      0.086     -5.097      0.000      -0.607
-0.270
V8            -0.2271      0.045     -5.068      0.000      -0.315
```

```
-0.139
V7            -0.1045      0.055     -1.888      0.059      -0.213
0.004
V4             0.7933      0.088      8.965      0.000       0.620
0.967
V13           -0.2437      0.093     -2.613      0.009      -0.427
-0.061
V14           -0.8932      0.067    -13.258      0.000      -1.025
-0.761
V22            0.2372      0.102      2.326      0.020       0.037
0.437
V12           -0.7873      0.114     -6.891      0.000      -1.011
-0.563
V21            0.1911      0.065      2.962      0.003       0.065
0.318
V9            -0.2117      0.082     -2.592      0.010      -0.372
-0.052
======================================================================
========
"""

logreg = LogisticRegression(C=1, penalty='l2')

logreg.fit(X_train[features], y_train)

LogisticRegression(C=1, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                   warm_start=False)

predict_summarize(logreg.predict(X_train[features]), y_train, 0.8,
True)

Accuracy =  0.999332878553801
Sensitivity =  0.6942857142857143
Specificity =  0.9998693559247088
False Positive Rate =  0.00013064407529118555

Precision =  0.9033457249070632
Recall =  0.6942857142857143
Plotting
```




```
             predicted_no  predicted_yes
ind
actual_no          198988             26
actual_yes            107            243
```

Observation:

This has a low recall value. We can continue with other models & recursive feature eliminations.

Random Forest

```
'''
n_estimators = [50, 100]
max_features = ['auto','sqrt']
criterion = ["gini", "entropy"]
max_depth = [5,10,15]
min_samples_split = [30, 50]
min_impurity_decrease = [0.1, 0.2]
param_grid = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
              'min_samples_split': min_samples_split,
              'criterion':criterion,
```

```
                'bootstrap':[True],
                'oob_score':[True]}

grid_search = GridSearchCV(estimator = RandomForestClassifier(),
                           param_grid = param_grid,
                        cv = kf, n_jobs = 8, verbose = 2)
grid_search.fit(X_train, y_train)
print("tuned hpyerparameters :(best parameters) ",
grid_search.best_params_)
print("accuracy :", grid_search.best_score_)
'''
```

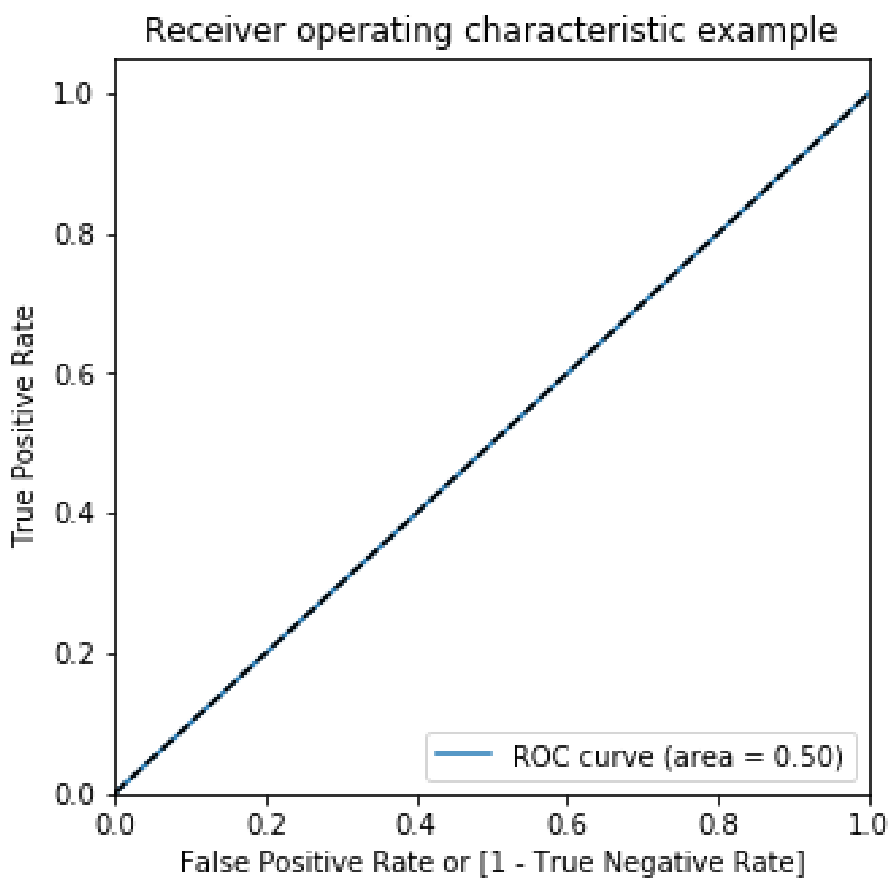
```
'\nn_estimators = [50, 100]\nmax_features = [\'auto\',\'sqrt\']\
ncriterion = ["gini", "entropy"]\nmax_depth = [5,10,15]\
nmin_samples_split = [30, 50]\nmin_impurity_decrease = [0.1, 0.2]\
nparam_grid = {\'n_estimators\': n_estimators,\
n                \'max_features\': max_features,\
n                \'max_depth\': max_depth,\
n                \'min_samples_split\': min_samples_split,\n
\'criterion\':criterion,\n                \'bootstrap\':[True],\n
\'oob_score\':[True]}\n\ngrid_search = GridSearchCV(estimator =
RandomForestClassifier(), \n                           param_grid =
param_grid, \n                        cv = kf, n_jobs = 8, verbose =
2)\ngrid_search.fit(X_train, y_train)\nprint("tuned hpyerparameters :
(best parameters) ", grid_search.best_params_)\nprint("accuracy :",
grid_search.best_score_)\n'
```

```
model_rf = RandomForestClassifier(bootstrap=True,
                                  criterion = 'gini',
                                  max_depth=5,
                                  max_features='auto',
                                  min_samples_split=50,
                                  n_estimators=50,
                                  min_impurity_decrease=0.1,
                                  random_state = 42,
                                  oob_score=True
                                 )
model_rf.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
class_weight=None,
                       criterion='gini', max_depth=5,
max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.1,
min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=50,
                       min_weight_fraction_leaf=0.0, n_estimators=50,
                       n_jobs=None, oob_score=True, random_state=42,
verbose=0,
                       warm_start=False)
```

```
predict_summarize([x[1] for x in model_rf.predict_proba(X_train)],
y_train, 0.32, True)
```

```
Accuracy =  0.998244417246845
Sensitivity =  0.0
Specificity =  1.0
False Positive Rate =  0.0

Precision =  nan
Recall =  0.0
Plotting
```




```
            predicted_no  predicted_yes
ind
actual_no         199014              0
actual_yes           350              0
```

Observation:

This model seems to be performing very poor by only predicting no.

RFE with Random Forest

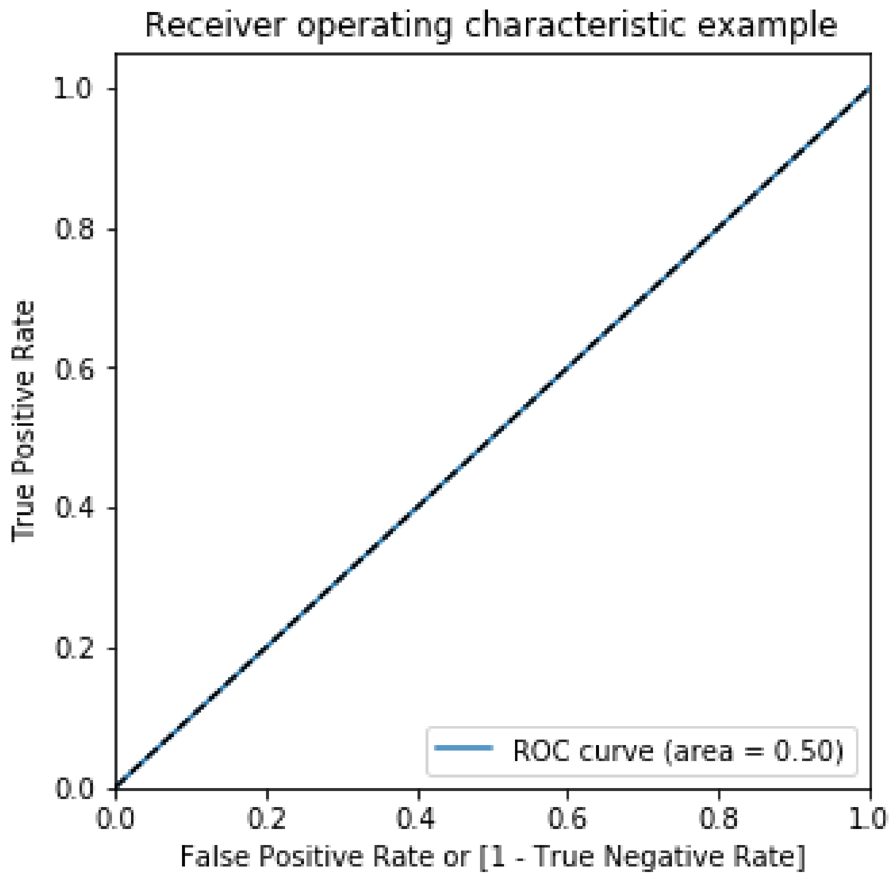
```
model_rf = RandomForestClassifier(bootstrap=True,
                                  criterion = 'gini',
```

```
                                  max_depth=5,
                                  max_features='auto',
                                  min_samples_split=50,
                                  n_estimators=50,
                                  min_impurity_decrease=0.1,
                                  random_state = 42,
                                  oob_score=True
                                 )

model_rf.fit(X_train[features], y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
class_weight=None,
                       criterion='gini', max_depth=5,
max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.1,
min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=50,
                       min_weight_fraction_leaf=0.0, n_estimators=50,
                       n_jobs=None, oob_score=True, random_state=42,
verbose=0,
                       warm_start=False)
```

```
predict_summarize([x[1] for x in
model_rf.predict_proba(X_train[features])], y_train, 0.32, True)
```

```
Accuracy =  0.998244417246845
Sensitivity =  0.0
Specificity =  1.0
False Positive Rate =  0.0

Precision =  nan
Recall =  0.0
Plotting
```




```
            predicted_no  predicted_yes
ind
actual_no         199014              0
actual_yes           350              0
```

Notice that the metrics is same even after using RFE features, hence not deleting features for final variable selection

Model building with balancing Classes

Perform class balancing with :

- Random Oversampling
- SMOTE
- ADASYN

This can be done on the train data to help the model recognise the fraudulant transactions better.

Class Imbalance

```
print('Fraud flag 1 count: ',y.value_counts()[1])
print('Fraud flag 0 count: ',y.value_counts()[0])
ClassImbRatio=y.value_counts()[1]/len(y) * 100
```

```
print('Class imbalance ratio: ',round(ClassImbRatio,3))
```

```
Fraud flag 1 count:  492
Fraud flag 0 count:  284315
Class imbalance ratio:  0.173
```

SMOTE

```
# Implementing SMOTE
smote=SMOTE(sampling_strategy=0.3, random_state=42, k_neighbors=3)

X_train_smote,y_train_smote=smote.fit_sample(X_train,y_train)

print('Before SMOTE',Counter(y_train))
print('After SMOTE',Counter(y_train_smote))
```

```
Before SMOTE Counter({0: 199014, 1: 350})
After SMOTE Counter({0: 199014, 1: 59704})
```

ADASYN

```
X_train_adasyn, y_train_adasyn = ADASYN(sampling_strategy=0.3,
random_state=42).fit_sample(X_train,y_train)

print('Before ADASYN',Counter(y_train))
print('After ADASYN',Counter(y_train_adasyn))
```

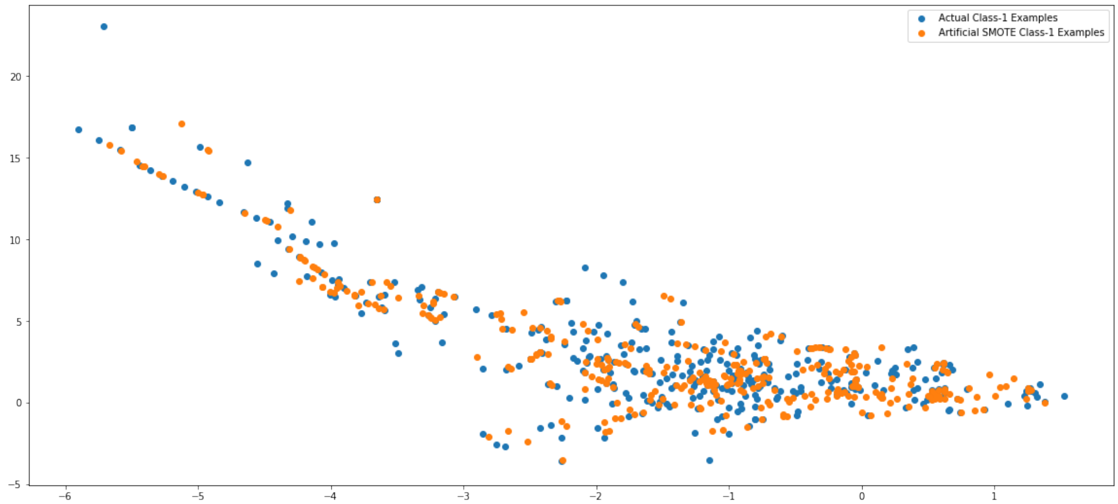
```
Before ADASYN Counter({0: 199014, 1: 350})
After ADASYN Counter({0: 199014, 1: 59666})
```

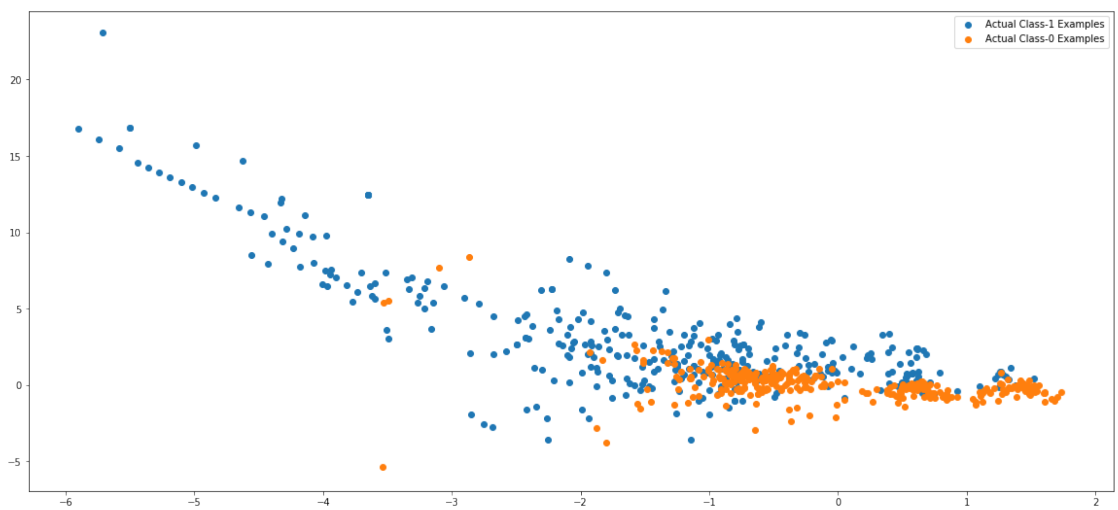
Print the class distribution after applying SMOTE

```
X_train_smote_1 = X_train_smote[X_train.shape[0]:]

X_train_1 = X_train.to_numpy()[numpy.where(y_train==1.0)]
X_train_0 = X_train.to_numpy()[numpy.where(y_train==0.0)]


plt.rcParams['figure.figsize'] = [20, 20]
fig = plt.figure()

plt.subplot(2, 1, 1)
plt.scatter(X_train_1[:, 0], X_train_1[:, 1], label='Actual Class-1
Examples')
plt.scatter(X_train_smote_1.iloc[:X_train_1.shape[0], 0],
X_train_smote_1.iloc[:X_train_1.shape[0], 1],
            label='Artificial SMOTE Class-1 Examples')
plt.legend()

plt.subplot(2, 1, 2)
plt.scatter(X_train_1[:, 0], X_train_1[:, 1], label='Actual Class-1
Examples')
plt.scatter(X_train_0[:X_train_1.shape[0], 0],
```

```
X_train_0[:X_train_1.shape[0], 1], label='Actual Class-0 Examples')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f5783f47890>
```



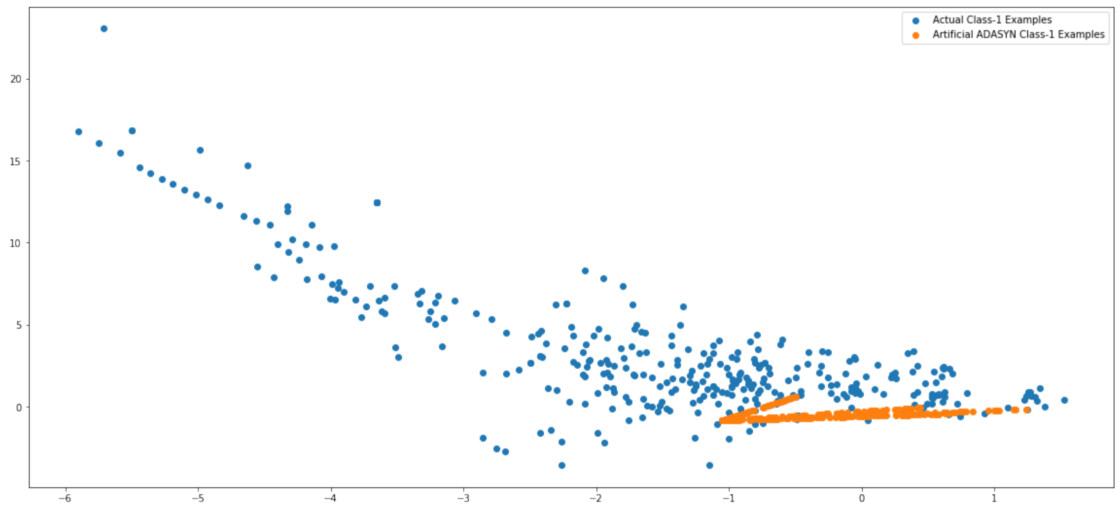


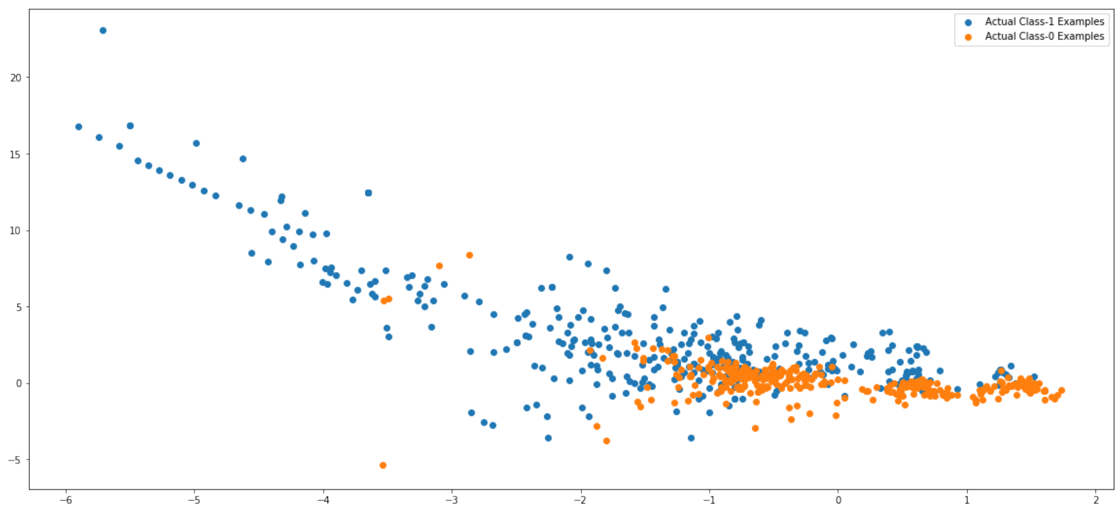
Print the class distribution after applying ADASYN

```
import warnings
warnings.filterwarnings("ignore")

# Artificial minority samples and corresponding minority labels from
ADASYN are appended
# below X_train and y_train respectively
# So to exclusively get the artificial minority samples from ADASYN,
we do
X_train_adasyn_1 = X_train_adasyn[X_train.shape[0]:]

X_train_1 = X_train.to_numpy()[numpy.where(y_train==1.0)]
X_train_0 = X_train.to_numpy()[numpy.where(y_train==0.0)]
```

```
import matplotlib.pyplot as plt
%matplotlib inline
plt.rcParams['figure.figsize'] = [20, 20]
fig = plt.figure()

plt.subplot(2, 1, 1)
plt.scatter(X_train_1[:, 0], X_train_1[:, 1], label='Actual Class-1
Examples')
plt.scatter(X_train_adasyn_1.iloc[:X_train_1.shape[0], 0],
X_train_adasyn_1.iloc[:X_train_1.shape[0], 1],
            label='Artificial ADASYN Class-1 Examples')
plt.legend()

plt.subplot(2, 1, 2)
plt.scatter(X_train_1[:, 0], X_train_1[:, 1], label='Actual Class-1
Examples')
plt.scatter(X_train_0[:X_train_1.shape[0], 0],
X_train_0[:X_train_1.shape[0], 1], label='Actual Class-0 Examples')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7f5783d24390>
```





Looking at the distribution of the classes, it seems like ADASYN has created more similar samples as compared to SMOTE

Models on Oversampled Data

Logistic Regression - SMOTE

```
'''
grid_params = {
    "C":numpy.logspace(-3, 3,7),
    "penalty":["l1", "l2"]
}
logreg = LogisticRegression()
logreg_cv = GridSearchCV(logreg, grid_params, cv=kf)
logreg_cv.fit(X_train_smote, y_train_smote)
print("tuned hpyerparameters :(best parameters) ",
logreg_cv.best_params_)
```

```
print("accuracy :", logreg_cv.best_score_)
'''
```

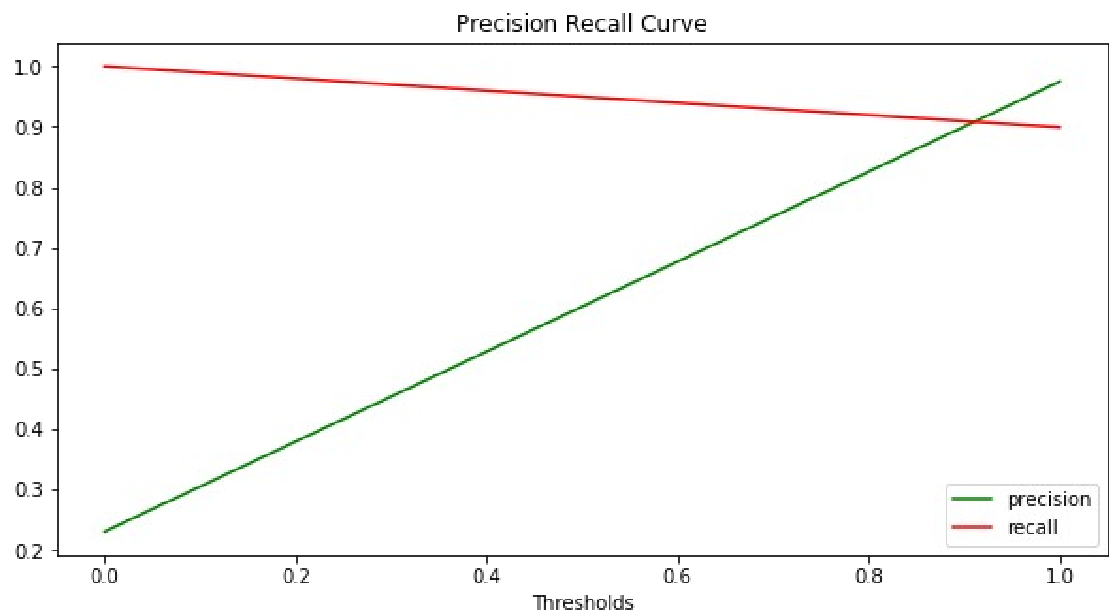
```
'\ngrid_params = {\n    "C":numpy.logspace(-3, 3,7), \n    "penalty":
["l1", "l2"]\n}\nlogreg = LogisticRegression()\nlogreg_cv =
GridSearchCV(logreg, grid_params, cv=kf)\nlogreg_cv.fit(X_train_smote,
y_train_smote)\nprint("tuned hpyerparameters :(best parameters) ",
logreg_cv.best_params_)\nprint("accuracy :", logreg_cv.best_score_)\n'
```

```
logreg = LogisticRegression(C=10, penalty='l2')
```

```
logreg.fit(X_train_smote, y_train_smote)
```

```
LogisticRegression(C=10, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                   warm_start=False)
```

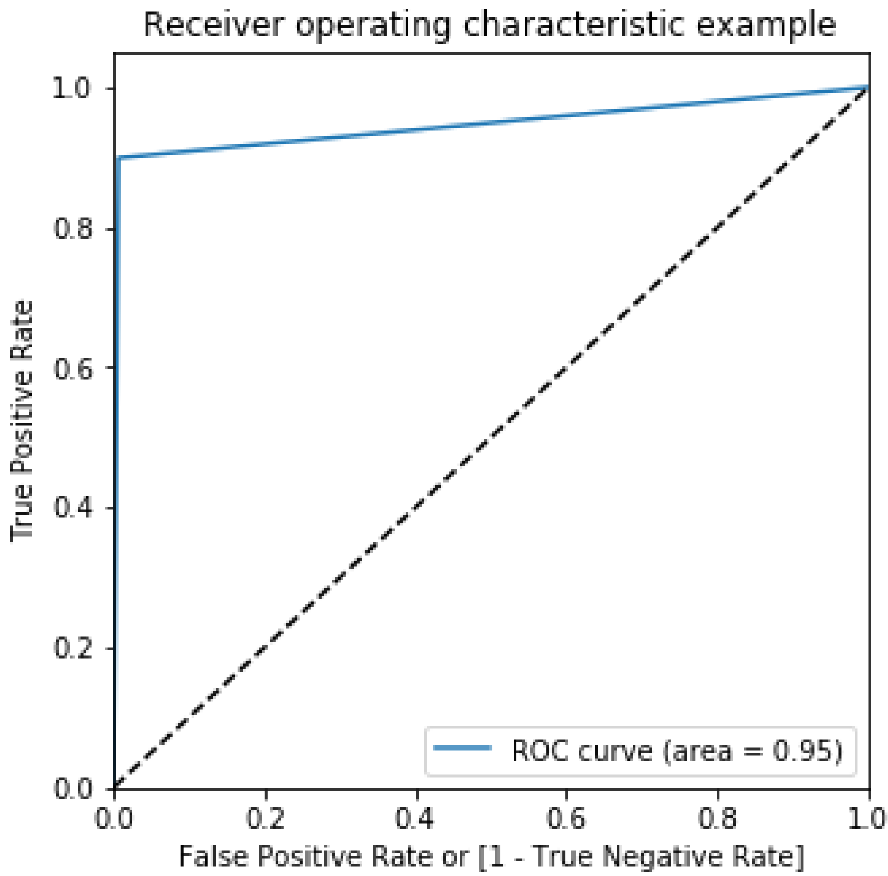
```
plot_precision_recall_curve(logreg.predict(X_train_smote),
y_train_smote)
```




```
predict_summarize(logreg.predict(X_train_smote), y_train_smote, 0.9,
True)
```

```
Accuracy =  0.9714128897100318
Sensitivity =  0.8992697306713118
Specificity =  0.9930557649210608
False Positive Rate =  0.00694423507893917
```

```
Precision =  0.9749055781522371
Recall =  0.8992697306713118
Plotting
```




```
            predicted_no  predicted_yes
ind
actual_no         197632           1382
actual_yes          6014          53690
```

Observation:

The recall value has improved upon balancing of data, we can try the same with adasyn & random forest.

Logistic Regression - ADASYN

```
'''
grid_params = {
    "C":numpy.logspace(-3, 3,7),
    "penalty":["l1", "l2"]
}
logreg = LogisticRegression()
logreg_cv = GridSearchCV(logreg, grid_params, cv=kf)
logreg_cv.fit(X_train_adasyn, y_train_adasyn)
print("tuned hpyerparameters :(best parameters) ",
```

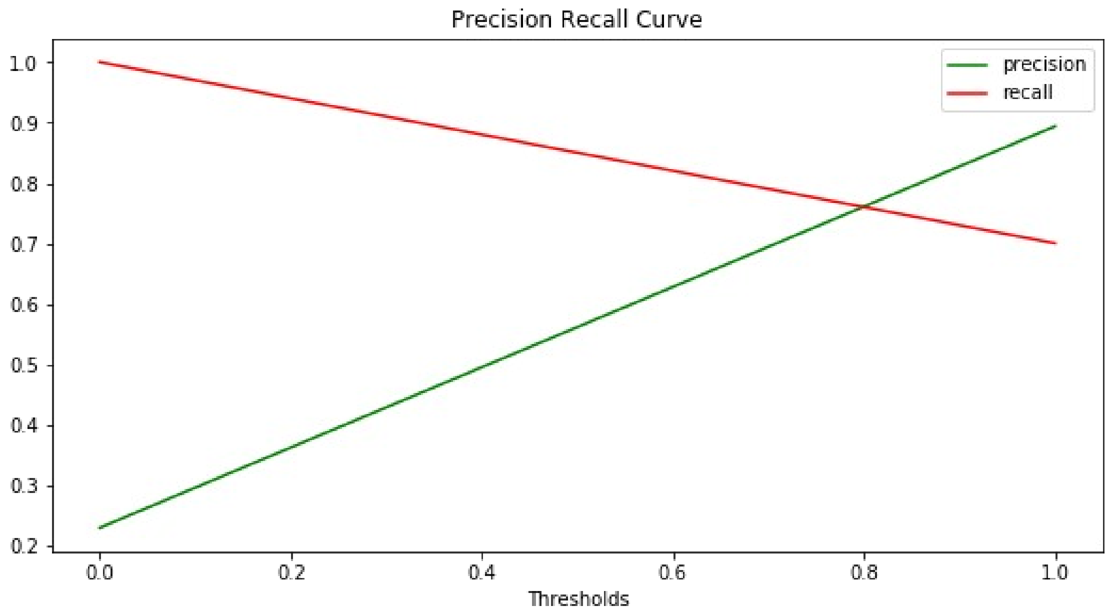
```
logreg_cv.best_params_)
print("accuracy :", logreg_cv.best_score_)
'''
```

```
'\ngrid_params = {\n    "C":numpy.logspace(-3, 3,7), \n    "penalty":
["l1", "l2"]\n}\nlogreg = LogisticRegression()\nlogreg_cv =
GridSearchCV(logreg, grid_params, cv=kf)\
nlogreg_cv.fit(X_train_adasyn, y_train_adasyn)\nprint("tuned
hpyerparameters :(best parameters) ", logreg_cv.best_params_)\
nprint("accuracy :", logreg_cv.best_score_)\n'
```

```
logreg = LogisticRegression(C=0.001, penalty='l2')

logreg.fit(X_train_adasyn, y_train_adasyn)
```

```
LogisticRegression(C=0.001, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                   warm_start=False)
```

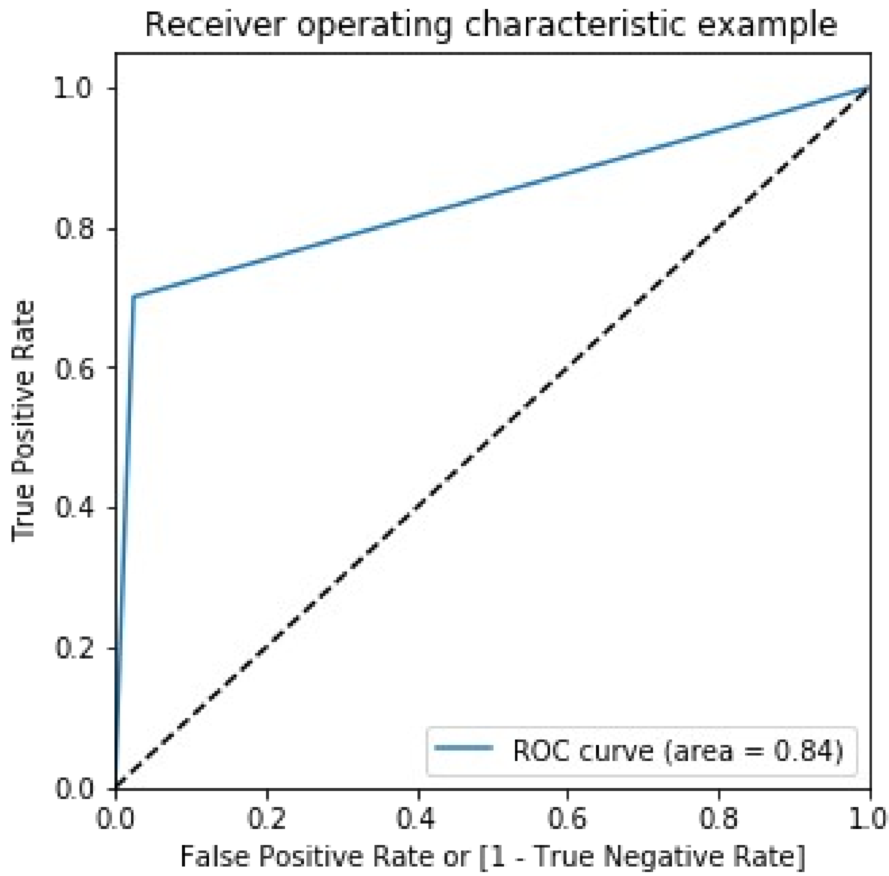
```
plot_precision_recall_curve(logreg.predict(X_train_adasyn),
y_train_adasyn)
```




```
predict_summarize(logreg.predict(X_train_adasyn), y_train_adasyn, 0.8,
True)
```

```
Accuracy =  0.9117867635688882
Sensitivity =  0.7005664867763886
Specificity =  0.9751123036570292
```

```
False Positive Rate =  0.024887696342970847

Precision =  0.8940602742070027
Recall =  0.7005664867763886
Plotting
```




```
            predicted_no  predicted_yes
ind
actual_no         194061           4953
actual_yes         17866          41800
```

Observation:

The recall value was relatively better in case of smote + logistic.

RFE with SMOTE data for Logistics Regression

```
'''
features=X_train_smote.columns

X_train_sm = sm.add_constant(X_train_smote[features])
logm2 = sm.GLM(y_train_smote,X_train_sm, family =
sm.families.Binomial())
model = logm2.fit()
```

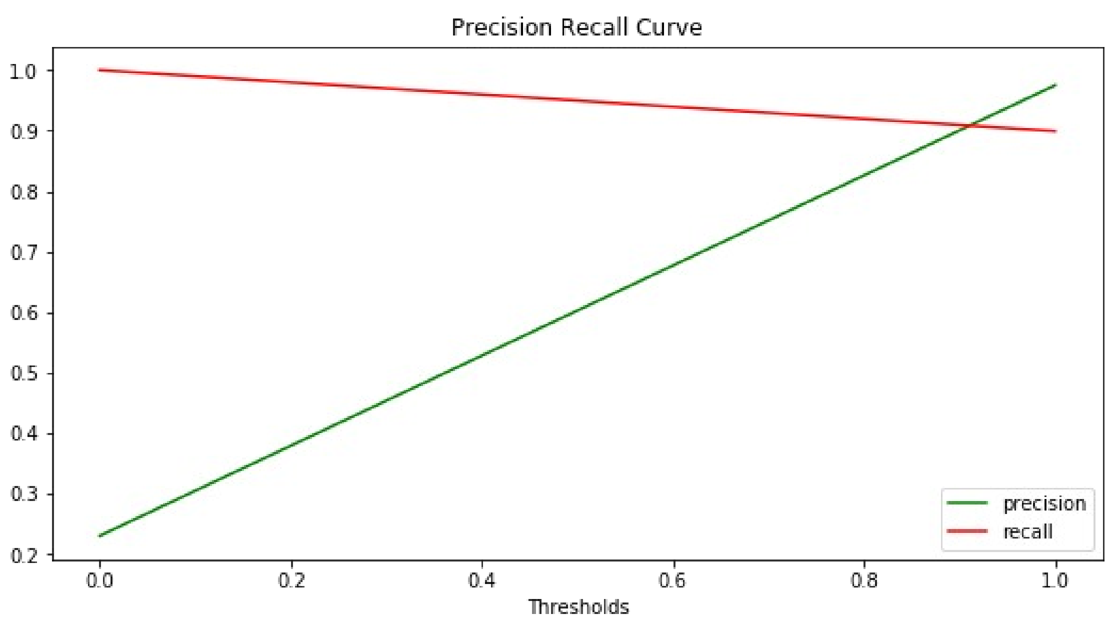
```
model.summary()
'''
```

```
'\nfeatures=X_train_smote.columns\n\nX_train_sm =
sm.add_constant(X_train_smote[features])\nlogm2 =
sm.GLM(y_train_smote,X_train_sm, family = sm.families.Binomial())\
nmodel = logm2.fit()\nmodel.summary()\n'
```

```
# Removing features with p-value >0.05 & not removing Amount as it is
an Important feature

features=list(set(X_train_smote.columns)-set(['V23','V27']))

logreg = LogisticRegression(C=1, penalty='l2')

logreg.fit(X_train_smote[features], y_train_smote)
```

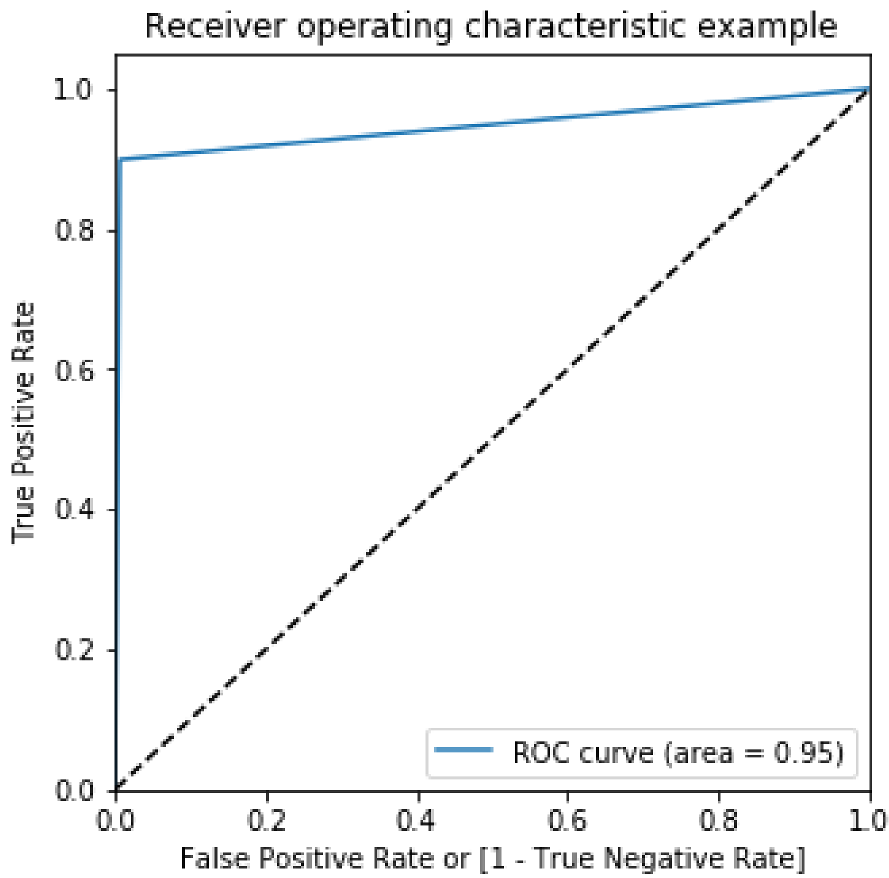
```
LogisticRegression(C=1, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                   warm_start=False)
```

```
plot_precision_recall_curve(logreg.predict(X_train_smote[features]),
y_train_smote)
```




```
predict_summarize(logreg.predict(X_train_smote[features]),
y_train_smote, 0.9, True)
```

```
Accuracy =  0.9714360809839284
Sensitivity =  0.8992697306713118
Specificity =  0.9930859135538204
False Positive Rate =  0.006914086446179666

Precision =  0.9750118040169978
Recall =  0.8992697306713118
Plotting
```




```
            predicted_no  predicted_yes
ind
actual_no         197638           1376
actual_yes          6014          53690
```

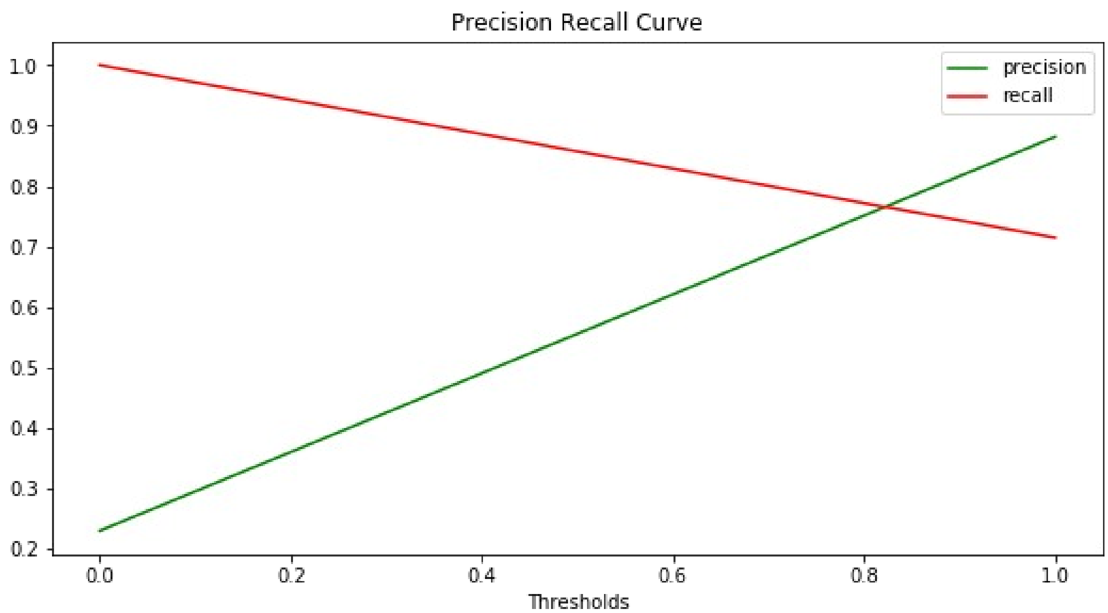
Observation:

The recall value has not changed much as compared to the metric prior to rfe

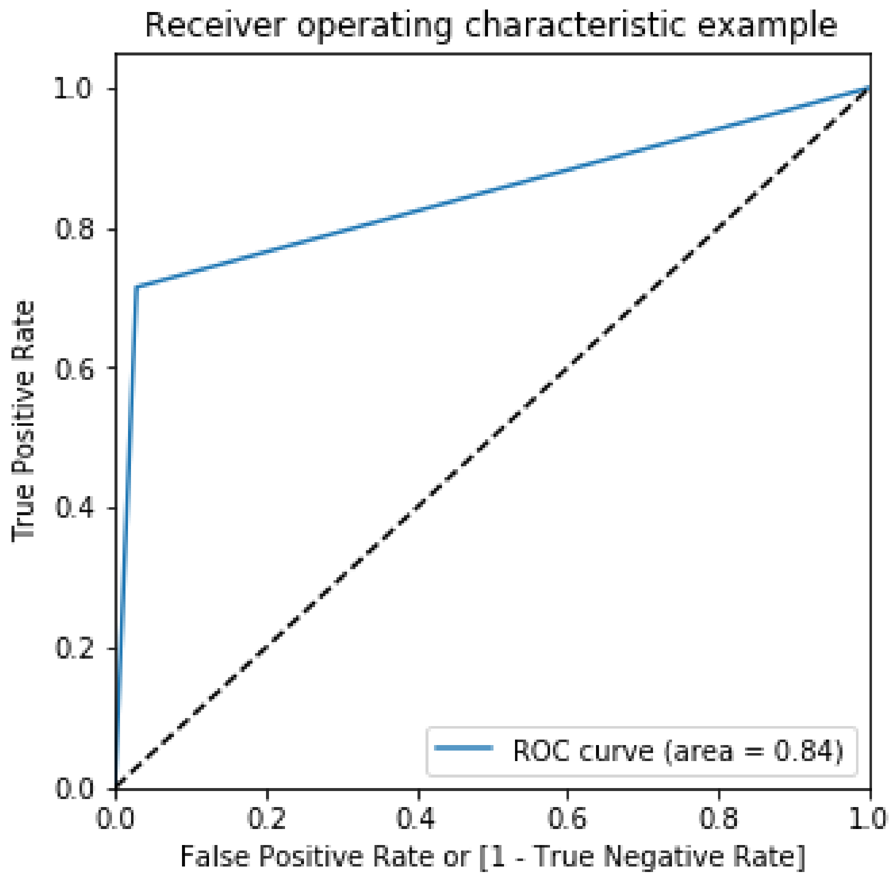
RFE with ADASYN data

```
# Removing features with p-value >0.05 & not removing Amount as it is an Important feature

features=list(set(X_train_adasyn.columns)-set(['V23','V27']))
```

```
logreg = LogisticRegression(C=1, penalty='l2')

logreg.fit(X_train_adasyn[features], y_train_adasyn)

LogisticRegression(C=1, class_weight=None, dual=False,
fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001,
verbose=0,
                   warm_start=False)

plot_precision_recall_curve(logreg.predict(X_train_adasyn[features]),
y_train_adasyn)
```




```
predict_summarize(logreg.predict(X_train_adasyn[features]),
y_train_adasyn, 0.9, True)

Accuracy =  0.9121037575382712
Sensitivity =  0.7149465357154828
Specificity =  0.9712130804867999
False Positive Rate =  0.028786919513200077

Precision =  0.8816004298675264
Recall =  0.7149465357154828
Plotting
```




```
             predicted_no  predicted_yes
ind
actual_no          193285           5729
actual_yes          17008          42658
```

Observation:

The recall value has not changed much as compared to the metric prior to rfe

Random Forest - SMOTE

```
'''
n_estimators = [50, 100]
max_features = ['auto','sqrt']
criterion = ["gini", "entropy"]
max_depth = [5,10,15]
min_samples_split = [30, 50]
min_impurity_decrease = [0.1, 0.2]
param_grid = {'n_estimators': n_estimators,
                'max_features': max_features,
                'max_depth': max_depth,
                'min_samples_split': min_samples_split,
                'criterion':criterion,
                'bootstrap':[True],
```

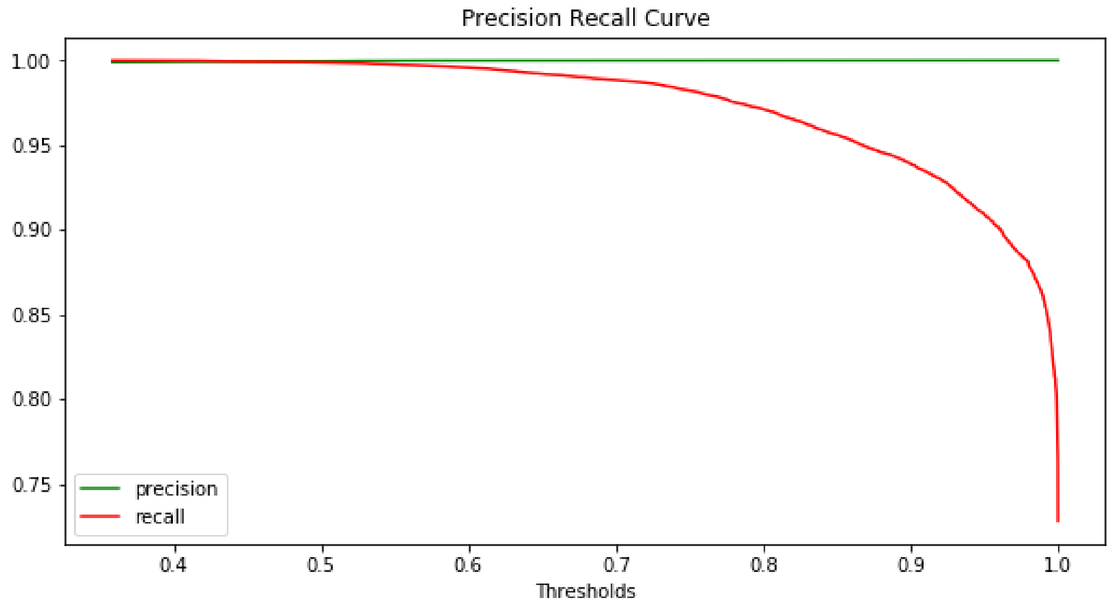
```
                'oob_score':[True]}

grid_search = GridSearchCV(estimator = RandomForestClassifier(),
                           param_grid = param_grid,
                        cv = kf, n_jobs = 8, verbose = 2)
grid_search.fit(X_train_smote, y_train_smote)
print("tuned hpyerparameters :(best parameters) ",
grid_search.best_params_)
print("accuracy :", grid_search.best_score_)
'''
```

```
'\nn_estimators = [50, 100]\nmax_features = [\'auto\',\'sqrt\']\
ncriterion = ["gini", "entropy"]\nmax_depth = [5,10,15]\
nmin_samples_split = [30, 50]\nmin_impurity_decrease = [0.1, 0.2]\
nparam_grid = {\'n_estimators\': n_estimators,\
n                \'max_features\': max_features,\
n                \'max_depth\': max_depth,\
n                \'min_samples_split\': min_samples_split,\n
\'criterion\':criterion,\n                \'bootstrap\':[True],\n
\'oob_score\':[True]}\n\ngrid_search = GridSearchCV(estimator =
RandomForestClassifier(), \n                           param_grid =
param_grid, \n                        cv = kf, n_jobs = 8, verbose =
2)\ngrid_search.fit(X_train_smote, y_train_smote)\nprint("tuned
hpyerparameters :(best parameters) ", grid_search.best_params_)\
nprint("accuracy :", grid_search.best_score_)\n'
```

```
model_rf_smote = RandomForestClassifier(bootstrap=True,
                                  criterion = 'entropy',
                                  max_depth=15,
                                  max_features='sqrt',
                                  min_samples_split=30,
                                  n_estimators=50,
                                  random_state = 42,
                                  oob_score=True
                                 )
model_rf_smote.fit(X_train_smote, y_train_smote)
```

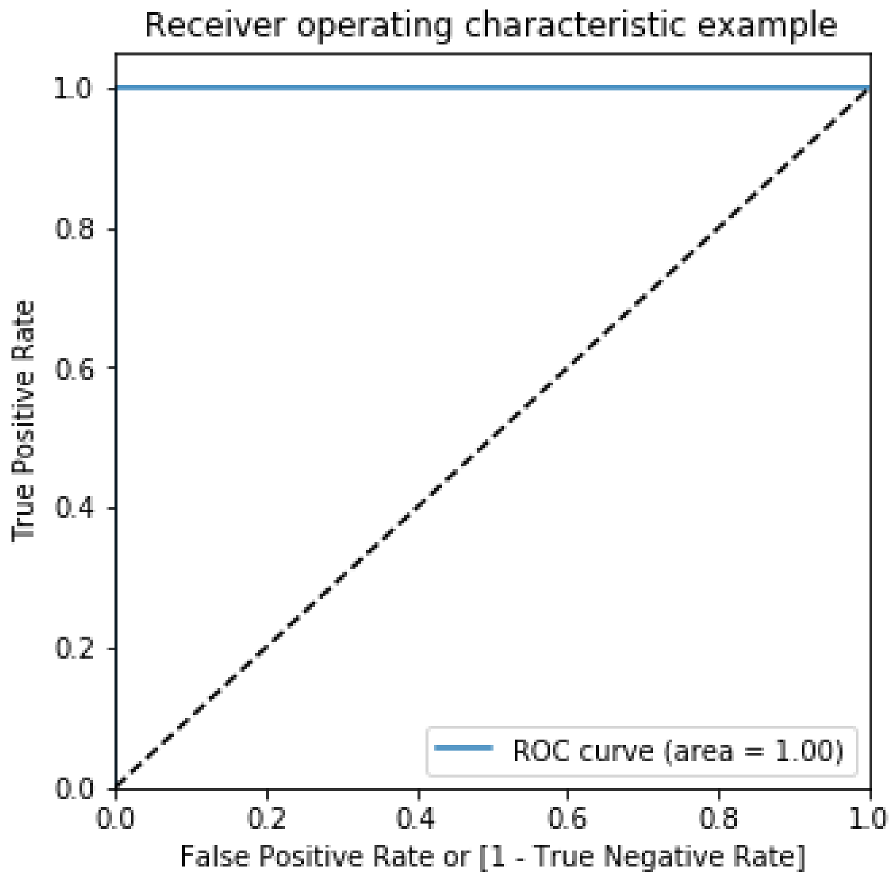
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
class_weight=None,
                       criterion='entropy', max_depth=15,
max_features='sqrt',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0,
min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=30,
                       min_weight_fraction_leaf=0.0, n_estimators=50,
                       n_jobs=None, oob_score=True, random_state=42,
verbose=0,
                       warm_start=False)
```

```
plot_precision_recall_curve([x[1] for x in
model_rf_smote.predict_proba(X_train_smote)], y_train_smote.values)
```




```
predict_summarize([x[1] for x in
model_rf_smote.predict_proba(X_train_smote)], y_train_smote, 0.5,
True)
```

```
Accuracy =  0.9996444004669176
Sensitivity =  0.9988610478359908
Specificity =  0.999879405468962
False Positive Rate =  0.00012059453103801742

Precision =  0.999597720415689
Recall =  0.9988610478359908
Plotting
```




```
            predicted_no  predicted_yes
ind
actual_no         198990             24
actual_yes            68          59636
```

Observation:

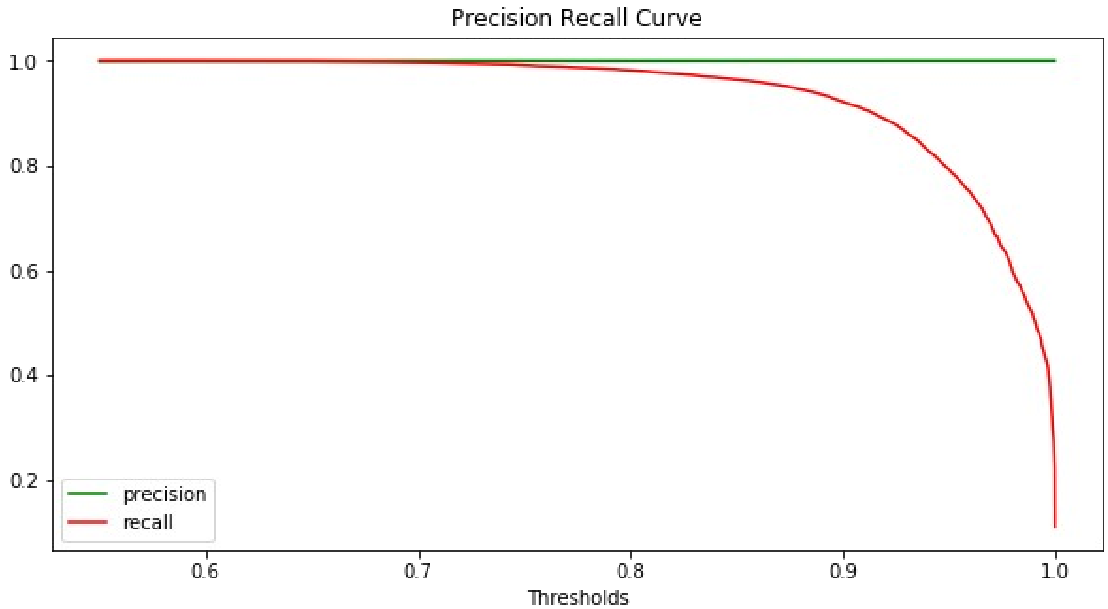
There is a drastic improvement in the recall value as per the balanced train data is concerned, we can try another iteration with adasyn sampled data.

Random Forest - ADASYN

```
'''
n_estimators = [50, 100]
max_features = ['auto','sqrt']
criterion = ["gini", "entropy"]
max_depth = [5,10,15]
min_samples_split = [30, 50]
min_impurity_decrease = [0.1, 0.2]
param_grid = {'n_estimators': n_estimators,
                'max_features': max_features,
                'max_depth': max_depth,
                'min_samples_split': min_samples_split,
                'criterion':criterion,
```

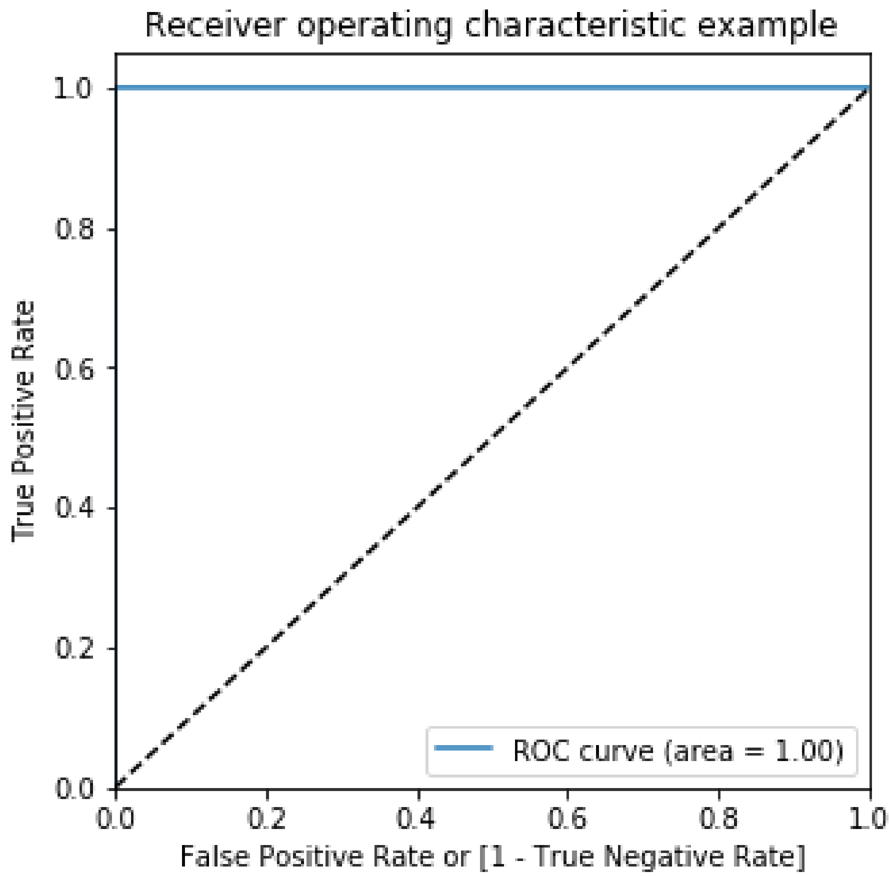
```
                 'bootstrap':[True],
                 'oob_score':[True]}

grid_search = GridSearchCV(estimator = RandomForestClassifier(),
                           param_grid = param_grid,
                        cv = kf, n_jobs = 8, verbose = 2)
grid_search.fit(X_train_adasyn, y_train_adasyn)
print("tuned hpyerparameters :(best parameters) ",
grid_search.best_params_)
print("accuracy :", grid_search.best_score_)
'''

'\nn_estimators = [50, 100]\nmax_features = [\'auto\',\'sqrt\']\
ncriterion = ["gini", "entropy"]\nmax_depth = [5,10,15]\
nmin_samples_split = [30, 50]\nmin_impurity_decrease = [0.1, 0.2]\
nparam_grid = {\'n_estimators\': n_estimators,\
n                 \'max_features\': max_features,\
n                 \'max_depth\': max_depth,\
n                 \'min_samples_split\': min_samples_split,\n
\'criterion\':criterion,\n                \'bootstrap\':[True],\n
\'oob_score\':[True]}\n\ngrid_search = GridSearchCV(estimator =
RandomForestClassifier(), \n                           param_grid =
param_grid, \n                       cv = kf, n_jobs = 8, verbose =
2)\ngrid_search.fit(X_train_adasyn, y_train_adasyn)\nprint("tuned
hpyerparameters :(best parameters) ", grid_search.best_params_)\
nprint("accuracy :", grid_search.best_score_)\n'

model_rf_adasyn = RandomForestClassifier(bootstrap=True,
                                  criterion = 'entropy',
                                  max_depth=15,
                                  max_features='sqrt',
                                  min_samples_split=30,
                                  n_estimators=50,
                                  random_state = 42,
                                  oob_score=True
                                 )
model_rf_adasyn.fit(X_train_adasyn, y_train_adasyn)

RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
class_weight=None,
                       criterion='entropy', max_depth=15,
max_features='sqrt',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0,
min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=30,
                       min_weight_fraction_leaf=0.0, n_estimators=50,
                       n_jobs=None, oob_score=True, random_state=42,
verbose=0,
                       warm_start=False)
```

```
plot_precision_recall_curve([x[1] for x in
model_rf_adasyn.predict_proba(X_train_adasyn)], y_train_adasyn.values)
```




```
predict_summarize([x[1] for x in
model_rf_adasyn.predict_proba(X_train_adasyn)], y_train_adasyn, 0.6,
True)
```

```
Accuracy =  0.999845368795423
Sensitivity =  0.9999162001810076
Specificity =  0.9998241329755696
False Positive Rate =  0.0001758670244304421

Precision =  0.9994136960600375
Recall =  0.9999162001810076
Plotting
```



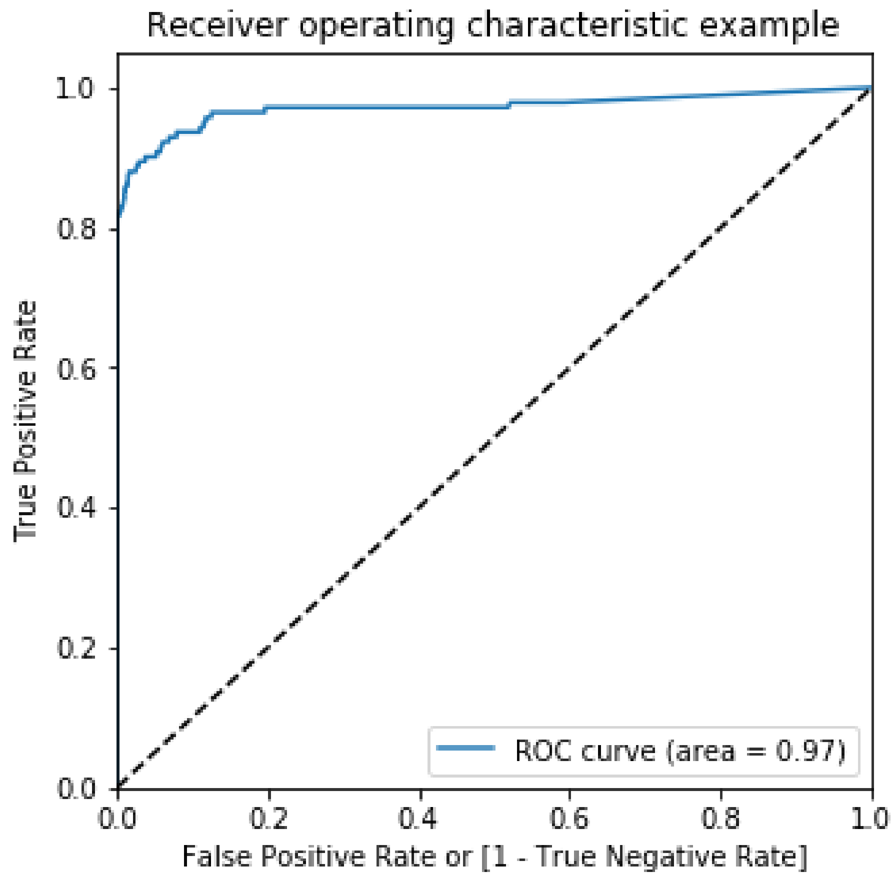

```
            predicted_no  predicted_yes
ind
actual_no         198979             35
actual_yes             5          59661
```

Observation:

The recall value has further improved and by far is the best in case of adasyn + random forest. We can retain this to be the final model & check the recall as per the test data.

```
predict_summarize([x[1] for x in
model_rf_adasyn.predict_proba(X_test)], y_test, 0.6, True)
```

```
Accuracy =  0.9992392589211521
Sensitivity =  0.795774647887324
Specificity =  0.9995779650883343
False Positive Rate =  0.0004220349116657483

Precision =  0.7583892617449665
Recall =  0.795774647887324
Plotting
```

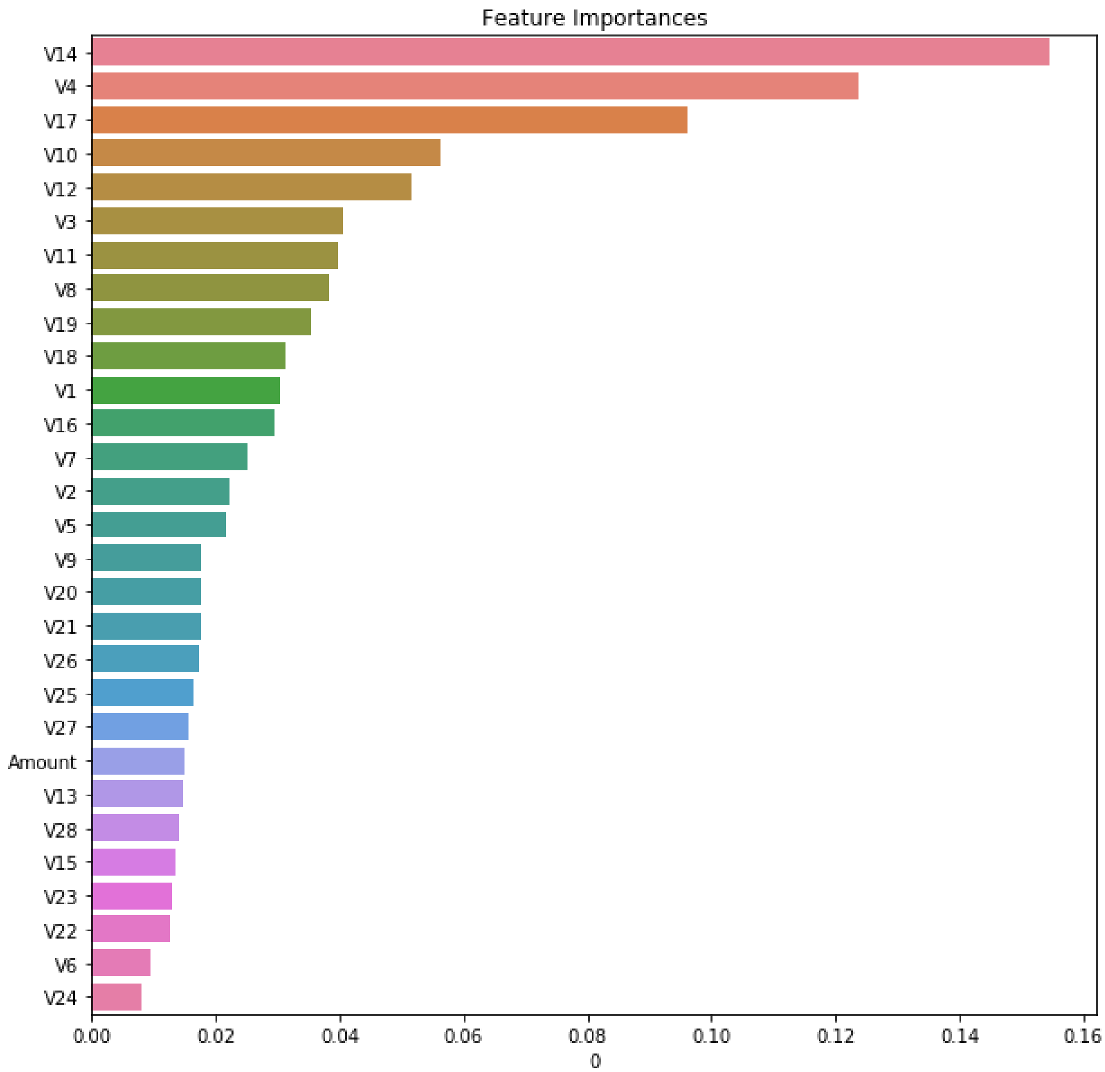



```
            predicted_no  predicted_yes
ind
actual_no          85265             36
actual_yes            29            113
```

Observation:

The recall value on the test data is quite different from the train data, but for a data with imbalance as low as 0.17%, the recall value of 80% is quite good.

```
plot_feature_importance(X_train_adasyn.columns,
model_rf_adasyn.feature_importances_)
```



The top influencing parameters are

- V14
- V4
- V17
- V10
- V12

This can help reduce the fraudulant transaction significantly as it reduces the number of transactions to be underwritten or verified, thus reducing the effort for the same process while increasing the safety. Only the transactions which are flagged by the model can be verified.