In [1]:
```
import os
import numpy as np
import pandas as pd
import matplotlib as plt
import matplotlib.pyplot as plt2
import seaborn as sns
import sys

from sklearn.model_selection import train_test_split,cross_val_predict,cross_val_
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,roc_auc_score,roc_curve,classificat:

from inspect import signature
from sklearn.metrics import average_precision_score,precision_recall_curve

from imblearn.over_sampling import SMOTE

from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from mpl_toolkits.mplot3d import Axes3D
```

In [2]:
```
if not sys.warnoptions:
    import warnings
    warnings.simplefilter("ignore")
```

In [3]:
```
os.chdir("C:/Users/Acesocloud/Downloads/Kaggle/Santander Customer Transaction Pre
```

In [4]:
```
df_santander = pd.read_csv("train.csv")
```

In [5]:
```
df_santander_test = pd.read_csv("test.csv")
```

In [6]:
```
print('Shape of our dataset:')
print(df_santander.shape,'\n')
```

```
Shape of our dataset:
(200000, 202)
```

In [7]:
```
pd.options.display.max_columns = None
```

Exploratory Data Analysis

In [8]:
```
print('*'*25,'Exploratory Data Analysis: ','*'*25,'\n')
```

```
************************* Exploratory Data Analysis:  *************************
```

In [9]:
```
print('Showing 1st few rows of our dataset: \n')
print(df_santander.head(5))
```

```
Showing 1st few rows of our dataset: 

   ID_code  target    var_0   var_1    var_2   var_3    var_4   var_5   var_6  \
0  train_0       0   8.9255 -6.7863  11.9081  5.0930  11.4607 -9.2834  5.1187   
1  train_1       0  11.5006 -4.1473  13.8588  5.3890  12.3622  7.0433  5.6208   
2  train_2       0   8.6093 -2.7457  12.0805  7.8928  10.5825 -9.0837  6.9427   
3  train_3       0  11.0604 -2.1518   8.9522  7.1957  12.5846 -1.8361  5.8428   
4  train_4       0   9.8369 -1.4834  12.8746  6.6375  12.2772  2.4486  5.9405   

     var_7   var_8   var_9  var_10   var_11   var_12   var_13  var_14  \
0  18.6266 -4.9200  5.7470  2.9252   3.1821  14.0137   0.5745  8.7989   
1  16.5338  3.1468  8.0851 -0.4032   8.0585  14.0239   8.4135  5.4345   
2  14.6155 -4.9193  5.9525 -0.3249 -11.2648  14.1929   7.3124  7.5244   
3  14.9250 -5.8609  8.2450  2.3061   2.8102  13.8463  11.9704  6.4569   
4  19.2514  6.2654  7.6784 -9.4458 -12.1419  13.8481   7.8895  7.7894   

    var_15   var_16   var_17   var_18   var_19   var_20   var_21   var_22  \
0  14.5691   5.7487  -7.2393   4.2840  30.7133  10.5350  16.2191   2.5791   
1  13.7003  13.8275 -15.5849   7.8000  28.5708   3.4287   2.7407   8.5524   
2  14.6472   7.6782  -1.7395   4.7011  20.4775  17.7559  18.1377   1.2145   
3  14.8372  10.7430  -0.4299  15.9426  13.7257  20.3010  12.5579   6.8202   
4  15.0553   8.4871  -3.0680   6.5263  11.3152  21.4246  18.9608  10.1102   

   var_23   var_24   var_25   var_26  var_27  var_28  var_29   var_30  \
0  2.4716  14.3831  13.4325  -5.1488 -0.4073  4.9306  5.9965  -0.3085   
1  3.3716   6.9779  13.8910 -11.7684 -2.5586  5.0464  0.5481  -9.2987   
2  3.5137   5.6777  13.2177  -7.9940 -2.9029  5.8463  6.1439 -11.1025   
3  2.7229  12.1354  13.7367   0.8135 -0.9059  5.9070  2.8407 -15.2398   
4  2.7142  14.2080  13.5433   3.1736 -3.3423  5.9015  7.9352  -3.1582   

    var_31  var_32   var_33   var_34   var_35  var_36  var_37   var_38  \
0  12.9041 -3.8766  16.8911  11.1920  10.5785  0.6764  7.8871   4.6667   
1   7.8755  1.2859  19.3710  11.3702   0.7399  2.7995  5.8434  10.8160   
2  12.4858 -2.2871  19.0422  11.0449   4.1087  4.6974  6.9346  10.8917   
3  10.4407 -2.5731   6.1796  10.6093  -5.9158  8.1723  2.8521   9.1738   
4   9.4668 -0.0083  19.3239  12.4057   0.6329  2.7922  5.8184  19.3038   

   var_39   var_40   var_41   var_42   var_43   var_44   var_45   var_46  \
0  3.8743  -5.2387   7.3746  11.5767  12.0446  11.6418  -7.0170   5.9226   
1  3.6783 -11.1147   1.8730   9.8775  11.7842   1.2444 -47.3797   7.3718   
2  0.9003 -13.5174   2.2439  11.5283  12.0406   4.1006  -7.9078  11.1405   
3  0.6665  -3.8294  -1.0370  11.7770  11.2834   8.0485 -24.6840  12.7404   
4  1.4450  -5.5963  14.0685  11.9171  11.5111   6.9087 -65.4863  13.8657   

    var_47   var_48   var_49   var_50   var_51  var_52  var_53  var_54  \
0 -14.2136  16.0283   5.3253  12.9194  29.0460 -0.6940  5.1736 -0.7474   
1   0.1948  34.4014  25.7037  11.8343  13.2256 -4.1083  6.6885 -8.0946   
2  -5.7864  20.7477   6.8874  12.9143  19.5856  0.7268  6.4059  9.3124   
3 -35.1659   0.7613   8.3838  12.6832   9.5503  1.7895  5.2091  8.0913   
4   0.0444  -0.1346  14.4268  13.3273  10.4857 -1.4367  5.7555 -8.5414   

    var_55   var_56  var_57  var_58   var_59   var_60   var_61  var_62  \
```

```
0  14.8322  11.2668  5.3822  2.0183  10.1166  16.1828   4.9590  2.0771
1  18.5995  19.3219  7.0118  1.9210   8.8682   8.0109  -7.2417  1.7944
2   6.2846  15.6372  5.8200  1.1000   9.1854  12.5963 -10.3734  0.8748
3  12.3972  14.4698  6.5850  3.3164   9.4638  15.7820 -25.0222  3.4418
4  14.1482  16.9840  6.1812  1.9548   9.2048   8.6591 -27.7439 -0.4952

    var_63  var_64  var_65  var_66   var_67  var_68  var_69   var_70  var_71  \
0 -0.2154  8.6748  9.5319  5.8056  22.4321  5.0109 -4.7010  21.6374  0.5663
1 -1.3147  8.1042  1.5365  5.4007   7.9344  5.0220  2.2302  40.5632  0.5134
2  5.8042  3.7163 -1.1016  7.3667   9.8565  5.0228 -5.7828   2.3612  0.8520
3 -4.3923  8.6464  6.3072  5.6221  23.6143  5.0220 -3.9989   4.0462  0.2500
4 -1.7839  5.2670 -4.3205  6.9860   1.6184  5.0301 -3.2431  40.1236  0.7737

    var_72   var_73   var_74   var_75   var_76   var_77  var_78   var_79  \
0  5.1999   8.8600  43.1127  18.3816  -2.3440  23.4104  6.5199  12.1983
1  3.1701  20.1068   7.7841   7.0529   3.2709  23.4822  5.5075  13.7814
2  6.3577  12.1719  19.7312  19.4465   4.5048  23.2378  6.3191  12.8046
3  1.2516  24.4187   4.5290  15.4235  11.6875  23.6273  4.0806  15.2733
4 -0.7264   4.5886  -4.5346  23.3521   1.0273  19.1600  7.1734  14.3937

     var_80   var_81   var_82   var_83  var_84   var_85   var_86   var_87  \
0  13.6468  13.8372   1.3675   2.9423 -4.5213  21.4669   9.3225  16.4597
1   2.5462  18.1782   0.3683  -4.8210 -5.4850  13.7867 -13.5901  11.0993
2   7.4729  15.7811  13.3529  10.1852  5.4604  19.0773  -4.4577   9.5413
3   0.7839  10.5404   1.6212  -5.2896  1.6027  17.9762  -2.3174  15.6298
4   2.9598  13.3317  -9.2587  -6.7075  7.8984  14.5265   7.0799  20.1670

     var_88   var_89   var_90  var_91   var_92   var_93   var_94  var_95  \
0   7.9984  -1.7069 -21.4494  6.7806  11.0924   9.9913  14.8421  0.1812
1   7.9022  12.2301   0.4768  6.8852   8.0905  10.9631  11.7569 -1.2722
2  11.9052   2.1447 -22.4038  7.0883  14.1613  10.5080  14.2621  0.2647
3   4.5474   7.5509  -7.5866  7.0364  14.4027  10.7795   7.2887 -1.0930
4   8.0053   3.7954 -39.7997  7.0065   9.3627  10.4316  14.0553  0.0213

     var_96   var_97  var_98  var_99  var_100  var_101  var_102  var_103  \
0   8.9642  16.2572  2.1743 -3.4132   9.4763  13.3102  26.5376   1.4403
1  24.7876  26.6881  1.8944  0.6939 -13.6950   8.4068  35.4734   1.7093
2  20.4031  17.0360  1.6981 -0.0269  -0.3939  12.6317  14.8863   1.3854
3  11.3596  18.1486  2.8344  1.9480 -19.8592  22.5316  18.6129   1.3512
4  14.7246  35.2988  1.6844  0.6715 -22.9264  12.3562  17.3410   1.6940

    var_104  var_105  var_106  var_107  var_108  var_109  var_110  var_111  \
0  14.7100   6.0454   9.5426  17.1554  14.1104  24.3627   2.0323   6.7602
1  15.1866   2.6227   7.3412  32.0888  13.9550  13.0858   6.6203   7.1051
2  15.0284   3.9995   5.3683   8.6273  14.1963  20.3882   3.2304   5.7033
3   9.3291   4.2835  10.3907   7.0874  14.3256  14.4135   4.2827   6.9750
4   7.1179   5.1934   8.8230  10.6617  14.0837  28.2749  -0.1937   5.9654

    var_112  var_113  var_114  var_115  var_116  var_117  var_118  var_119  \
0   3.9141  -0.4851   2.5240   1.5093   2.5516  15.5752 -13.4221   7.2739
1   5.3523   8.5426   3.6159   4.1569   3.0454   7.8522 -11.5100   7.5109
2   4.5255   2.1929   3.1290   2.9044   1.1696  28.7632 -17.2738   2.1056
3   1.6480  11.6896   2.5762  -2.5459   5.3446  38.1015   3.5732   5.0988
4   1.0719   7.9923   2.9138  -3.6135   1.4684  25.6795  13.8224   4.7478

    var_120  var_121  var_122  var_123  var_124  var_125  var_126  var_127  \
0  16.0094   9.7268   0.8897   0.7754   4.2218  12.0039  13.8571  -0.7338
```

```
1  31.5899   9.5018   8.2736  10.1633   0.1225  12.5942  14.5697   2.4354
2  21.1613   8.9573   2.7768  -2.1746   3.6932  12.4653  14.1978  -2.5511
3  30.5644  11.3025   3.9618  -8.2464   2.7038  12.3441  12.5431  -1.3683
4  41.1037  12.7140   5.2964   9.7289   3.9370  12.1316  12.5815   7.0642

   var_128  var_129  var_130  var_131  var_132  var_133  var_134  var_135  \
0  -1.9245  15.4462  12.8287   0.3587   9.6508   6.5674   5.1726   3.1345
1   0.8194  16.5346  12.4205  -0.1780   5.7582   7.0513   1.9568  -8.9921
2  -0.9479  17.1092  11.5419   0.0975   8.8186   6.6231   3.9358 -11.7218
3   3.5974  13.9761  14.3003   1.0486   8.9500   7.1954  -1.1984   1.9586
4   5.6518  10.9346  11.4266   0.9442   7.7532   6.6173  -6.8304   6.4730

   var_136  var_137  var_138  var_139  var_140  var_141  var_142  var_143  \
0  29.4547  31.4045   2.8279  15.6599   8.3307  -5.6011  19.0614  11.2663
1   9.7797  18.1577  -1.9721  16.1622   3.6937   6.6803  -0.3243  12.2806
2  24.5437  15.5827   3.8212   8.6674   7.3834  -2.4438  10.2158   7.4844
3  27.5609  24.6065  -2.8233   8.9821   3.8873  15.9638  10.0142   7.8388
4  17.1728  25.8128   2.6791  13.9547   6.6289  -4.3965  11.7159  16.1080

   var_144  var_145  var_146  var_147  var_148  var_149  var_150  var_151  \
0   8.6989   8.3694  11.5659 -16.4727   4.0288  17.9244  18.5177  10.7800
1   8.6086  11.0738   8.9231  11.7700   4.2578  -4.4223  20.6294  14.8743
2   9.1104   4.3649  11.4934   1.7624   4.0714  -1.2681  14.3330   8.0088
3   9.9718   2.9253  10.4994   4.1622   3.7613   2.3701  18.0984  17.1765
4   7.6874   9.1570  11.5670 -12.7047   3.7574   9.9110  20.1461   1.2995

   var_152  var_153  var_154  var_155  var_156  var_157  var_158  var_159  \
0   9.0056  16.6964  10.4838   1.6573  12.1749 -13.1324  17.6054  11.5423
1   9.4317  16.7242  -0.5687   0.1898  12.2419  -9.6953  22.3949  10.6261
2   4.4015  14.1479  -5.1747   0.5778  14.5362  -1.7624  33.8820  11.6041
3   7.6508  18.2452  17.0336 -10.9370  12.0500  -1.2155  19.9750  12.3892
4   5.8493  19.8234   4.7022  10.6101  13.0021 -12.6068  27.0846   8.0913

   var_160  var_161  var_162  var_163  var_164  var_165  var_166  var_167  \
0  15.4576   5.3133   3.6159   5.0384   6.6760  12.6644   2.7004  -0.6975
1  29.4846   5.8683   3.8208  15.8348  -5.0121  15.1345   3.2003   9.3192
2  13.2070   5.8442   4.7086   5.7141  -1.0410  20.5092   3.2790  -5.5952
3  31.8833   5.9684   7.2084   3.8899 -11.0882  17.2502   2.5881  -2.7018
4  33.5107   5.6953   5.4663  18.2201   6.5769  21.2607   3.2304  -1.7759

   var_168  var_169  var_170  var_171  var_172  var_173  var_174  var_175  \
0   9.5981   5.4879  -4.7645  -8.4254  20.8773   3.1531  18.5618   7.7423
1   3.8821   5.7999   5.5378   5.0988  22.0330   5.5134  30.2645  10.4968
2   7.3176   5.7690  -7.0927  -3.9116   7.2569  -5.8234  25.6820  10.9202
3   0.5641   5.3430  -7.1541  -6.1920  18.2366  11.7134  14.7483   8.1013
4   3.1283   5.5518   1.4493  -2.6627  19.8056   2.3705  18.4685  16.3309

   var_176  var_177  var_178  var_179  var_180  var_181  var_182  var_183  \
0 -10.1245  13.7241  -3.5189   1.7202  -8.4051   9.0164   3.0657  14.3691
1  -7.2352  16.5721  -7.3477  11.0752  -5.5937   9.4878 -14.9100   9.4245
2  -0.3104   8.8438  -9.7009   2.4013  -4.2935   9.3908 -13.2648   3.1545
3  11.8771  13.9552 -10.4701   5.6961  -3.7546   8.4117   1.8986   7.2601
4  -3.3456  13.5261   1.7189   5.1743  -7.6938   9.7685   4.8910  12.2198

   var_184  var_185  var_186  var_187  var_188  var_189  var_190  var_191  \
0  25.8398   5.8764  11.8411 -19.7159  17.5743   0.5857   4.4354   3.9642
1  22.5441  -4.8622   7.6543 -15.9319  13.3175  -0.3566   7.6421   7.7214
```

```
2  23.0866  -5.3000   5.3745  -6.2660  10.1934  -0.8417   2.9057   9.7905
3  -0.4639  -0.0498   7.9336 -12.8279  12.4124   1.8489   4.4666   4.7433
4  11.8503  -7.8931   6.4209   5.9270  16.0201  -0.2829  -1.4905   9.5214

   var_192  var_193  var_194  var_195  var_196  var_197  var_198  var_199
0   3.1364   1.6910  18.5227  -2.3978   7.8784   8.5635  12.7803  -1.0914
1   2.5837  10.9516  15.4305   2.0339   8.1267   8.7889  18.3560   1.9518
2   1.6704   1.6858  21.6042   3.1417  -6.5213   8.2675  14.7222   0.3965
3   0.7178   1.4214  23.0347  -1.2706  -2.9275  10.2922  17.9697  -8.9996
4  -0.1508   9.1942  13.2876  -1.5121   3.9267   9.5031  17.9974  -8.8104
```

In [10]:

```
print("Basic info about dataset:\n")
print(df_santander.info())
```

```
Basic info about dataset:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Columns: 202 entries, ID_code to var_199
dtypes: float64(200), int64(1), object(1)
memory usage: 308.2+ MB
None
```

In [11]:

```
print("Data Description:\n")
```

```
Data Description:
```

Target Class Count

In [12]:

```
target_count = df_santander['target'].value_counts()
```

In [13]:

```
print("Count of categories of the target variable:\n", target_count)
```

```
Count of categories of the target variable:
 0    179902
1     20098
Name: target, dtype: int64
```

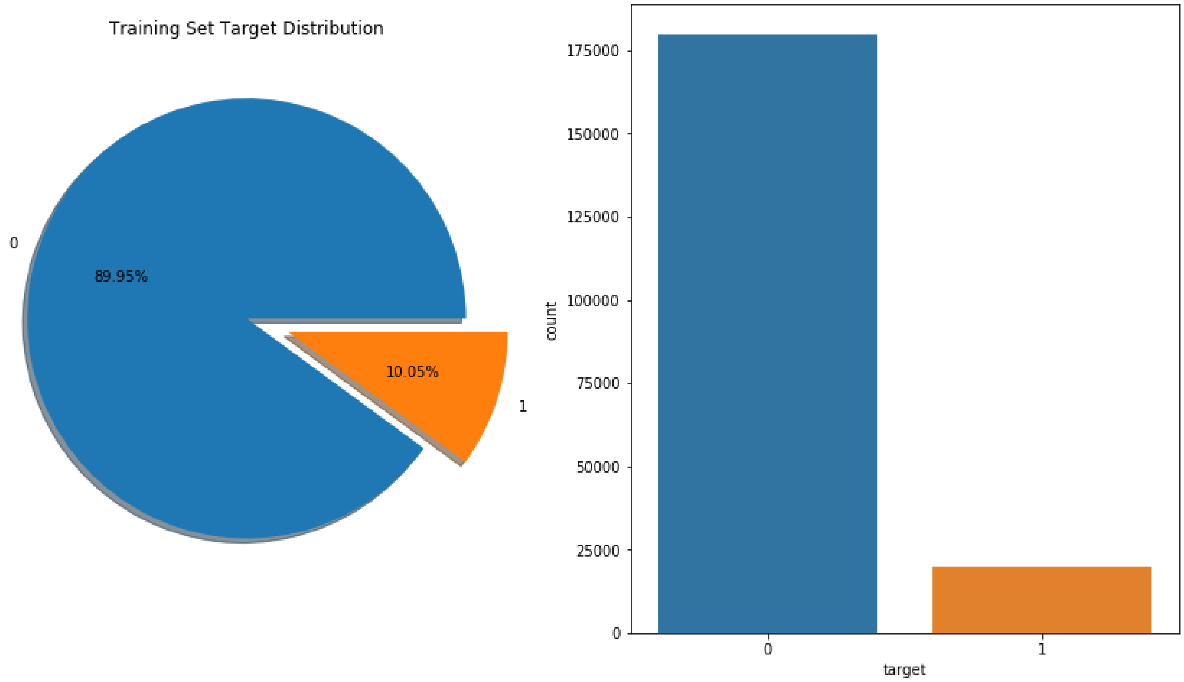
In [14]:

```
print("Percentage of each category of the target variable:\n", ((target_count/df_
```

```
Percentage of each category of the target variable:
 0    89.951
1    10.049
Name: target, dtype: float64
```

Data Visualization

In [15]:
```
f, ax = plt2.subplots(1,2,figsize=(15,8))
pie_data = df_santander['target'].value_counts()
pie_data.plot.pie(explode=[0,0.2], autopct='%1.2f%%', ax = ax[0], shadow = True)
ax[0].set_title('Training Set Target Distribution')
ax[0].set_ylabel('')

sns.countplot('target', data = df_santander, ax = ax[1])
plt2.show()
```



Missing Value Analysis

In [16]:
```
train_missing = df_santander.isnull().sum()
```

In [17]:
```
print("No. of rows having missing values in train data:")
print(train_missing.loc[train_missing > 0].shape[0])
```

```
No. of rows having missing values in train data:
0
```

In [18]:
```
test_missing = df_santander_test.isnull().sum()
print("No. of rows having missing values in test data:")
print(test_missing.loc[test_missing > 0].shape[0])
```

```
No. of rows having missing values in test data:
0
```

Outlier Analysis

```
Can not perform as we have imbalance dataset
```

Distribution of training data

In [19]:
```
def plot_train_data_dist(cat_0,cat_1, label1, label2, columns):
    i = 0
    sns.set_style('darkgrid')

    fig = plt2.figure()
    ax = plt2.subplots(10,10,figsize=(22,18))

    for col in columns:
        i += 1
        plt2.subplot(10,10,i)
        sns.distplot(cat_0[col], hist=False, label=label1)
        sns.distplot(cat_1[col], hist=False, label=label2)
        plt2.legend()
        plt2.xlabel('Attribute',)
    plt2.show()
```

In [20]:
```
cat_0 = df_santander.loc[df_santander['target'] == 0]
cat_1 = df_santander.loc[df_santander['target'] == 1]
```

In [21]:
```
label1 = '0'
label2 = '1'
```

In [22]:
```
columns = df_santander.columns.values[2:102]
plot_train_data_dist(cat_0, cat_1, label1, label2, columns)
```

```
<Figure size 432x288 with 0 Axes>
```

In [23]:
```
columns = df_santander.columns.values[102:202]
plot_train_data_dist(cat_0, cat_1, label1, label2, columns)
```

```
<Figure size 432x288 with 0 Axes>
```

Distribution of test data

In [24]:
```
def plot_test_data_dist(test_attributes):
    i=0
    sns.set_style('whitegrid')

    fig=plt2.figure()
    ax=plt2.subplots(10,10,figsize=(22,18))

    for attribute in test_attributes:
        i+=1
        plt2.subplot(10,10,i)
        sns.distplot(df_santander_test[attribute],hist=False)
        plt2.xlabel('Attribute',)
        sns.set_style("ticks", {"xtick.major.size": 8, "ytick.major.size": 8})
    plt2.show()
```

In [25]:
```
test_attributes=df_santander_test.columns.values[1:101]
plot_test_data_dist(test_attributes)
```

```
<Figure size 432x288 with 0 Axes>
```

In [26]:
```
test_attributes=df_santander_test.columns.values[102:202]
plot_test_data_dist(test_attributes)
```

```
<Figure size 432x288 with 0 Axes>
```

Check for duplicate rows

In [27]:
```
duplicateRowsDF = df_santander[df_santander.duplicated()]

print("No. of duplicate rows based on all columns are :")
print(duplicateRowsDF.shape[0])
```

```
No. of duplicate rows based on all columns are :
0
```

In [28]:
```
duplicateRowsDF = df_santander_test[df_santander_test.duplicated()]

print("No. of duplicate rows based on all columns are :")
print(duplicateRowsDF.shape[0])
```

```
No. of duplicate rows based on all columns are :
0
```

Correlation Analysis

In [29]:
```
num_train = df_santander.columns.values[2:202]
num_test = df_santander_test.columns.values[1:201]
```

Correlation between train data

In [30]:
```
train_corr = df_santander[num_train].corr().abs()
```

In [31]:
```
train_corr = train_corr.unstack()
train_corr
```

Out[31]:
```
var_0    var_0      1.000000
         var_1      0.000544
         var_2      0.006573
         var_3      0.003801
         var_4      0.001326
                      ...
var_199  var_195    0.002042
         var_196    0.000607
         var_197    0.004991
         var_198    0.004731
         var_199    1.000000
Length: 40000, dtype: float64
```

In [32]:
```
train_corr = train_corr.sort_values(kind="quicksort")
train_corr
```

Out[32]:
```
var_75   var_191    2.703975e-08
var_191  var_75     2.703975e-08
var_173  var_6      5.942735e-08
var_6    var_173    5.942735e-08
var_126  var_109    1.313947e-07
                        ...
var_128  var_128    1.000000e+00
var_127  var_127    1.000000e+00
var_126  var_126    1.000000e+00
var_124  var_124    1.000000e+00
var_199  var_199    1.000000e+00
Length: 40000, dtype: float64
```

In [33]:
```
train_corr = train_corr.reset_index()
train_corr
```

Out[33]:

	level_0	level_1	0
0	var_75	var_191	2.703975e-08
1	var_191	var_75	2.703975e-08
2	var_173	var_6	5.942735e-08
3	var_6	var_173	5.942735e-08
4	var_126	var_109	1.313947e-07
...	...	...	...
39995	var_128	var_128	1.000000e+00
39996	var_127	var_127	1.000000e+00
39997	var_126	var_126	1.000000e+00
39998	var_124	var_124	1.000000e+00
39999	var_199	var_199	1.000000e+00

40000 rows × 3 columns

In [34]:
```
train_corr
```

Out[34]:

	level_0	level_1	0
0	var_75	var_191	2.703975e-08
1	var_191	var_75	2.703975e-08
2	var_173	var_6	5.942735e-08
3	var_6	var_173	5.942735e-08
4	var_126	var_109	1.313947e-07
...	...	...	...
39995	var_128	var_128	1.000000e+00
39996	var_127	var_127	1.000000e+00
39997	var_126	var_126	1.000000e+00
39998	var_124	var_124	1.000000e+00
39999	var_199	var_199	1.000000e+00

40000 rows × 3 columns

Correlation between test data

In [35]:
```
test_corr = df_santander_test[num_test].corr().abs()
```

In [36]:
```
test_corr = test_corr.unstack()
```

In [37]:
```
test_corr = test_corr.sort_values(kind="quicksort")
```

In [38]:
```
test_corr = test_corr.reset_index()
```

Excluding correlation between same variables as that will be 1 always

In [39]:
```
train_corr = train_corr[train_corr['level_0']!=train_corr['level_1']]
train_corr
```

Out[39]:

	level_0	level_1	0
0	var_75	var_191	2.703975e-08
1	var_191	var_75	2.703975e-08
2	var_173	var_6	5.942735e-08
3	var_6	var_173	5.942735e-08
4	var_126	var_109	1.313947e-07
...	...	...	...
39795	var_165	var_81	9.713658e-03
39796	var_53	var_148	9.787532e-03
39797	var_148	var_53	9.787532e-03
39798	var_26	var_139	9.844361e-03
39799	var_139	var_26	9.844361e-03

39800 rows × 3 columns

In [40]:
```
test_corr = test_corr[test_corr['level_0']!=test_corr['level_1']]
```

In [41]:
```
test_corr.iloc[:,2].describe()
```

Out[41]:
```
count    3.980000e+04
mean     1.853484e-03
std      1.399296e-03
min      1.477268e-07
25%      7.349334e-04
50%      1.560695e-03
75%      2.689444e-03
max      9.867773e-03
Name: 0, dtype: float64
```

In [42]:
```
train_corr.iloc[:,2].describe()
```

Out[42]:
```
count    3.980000e+04
mean     1.986439e-03
std      1.506084e-03
min      2.703975e-08
25%      7.903091e-04
50%      1.679507e-03
75%      2.874466e-03
max      9.844361e-03
Name: 0, dtype: float64
```

In [43]:
```
train_corr=df_santander[num_train].corr()
train_corr
```

Out[43]:

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	
var_0	1.000000	-0.000544	0.006573	0.003801	0.001326	0.003046	0.006983	0.002429	0.0
var_1	-0.000544	1.000000	0.003980	0.000010	0.000303	-0.000902	0.003258	0.001511	0.0
var_2	0.006573	0.003980	1.000000	0.001001	0.000723	0.001569	0.000883	-0.000991	0.0
var_3	0.003801	0.000010	0.001001	1.000000	-0.000322	0.003253	-0.000774	0.002500	0.0
var_4	0.001326	0.000303	0.000723	-0.000322	1.000000	-0.001368	0.000049	0.004549	0.0
...	...	...	...	...	...	...	...	...	
var_195	0.002073	-0.000785	-0.001070	0.001206	0.003706	-0.001274	0.001244	0.001854	0.0
var_196	0.004386	-0.000377	0.003952	-0.002800	0.000513	0.002880	0.005378	0.001045	-0.0
var_197	-0.000753	-0.004157	0.001078	0.001164	-0.000046	-0.000535	-0.003565	0.003466	-0.0
var_198	-0.005776	-0.004861	-0.000877	-0.001651	-0.001821	-0.000953	-0.003025	0.000650	0.0
var_199	0.003850	0.002287	0.003855	0.000506	-0.000786	0.002767	0.006096	-0.001457	0.0

200 rows × 200 columns

In [44]:
```
train_corr=train_corr.values.flatten()
train_corr
```

Out[44]:
```
array([ 1.00000000e+00, -5.43699242e-04,  6.57283380e-03, ...,
        4.99055495e-03, -4.73055989e-03,  1.00000000e+00])
```

In [45]:
```
train_corr=train_corr[train_corr!=1]
```

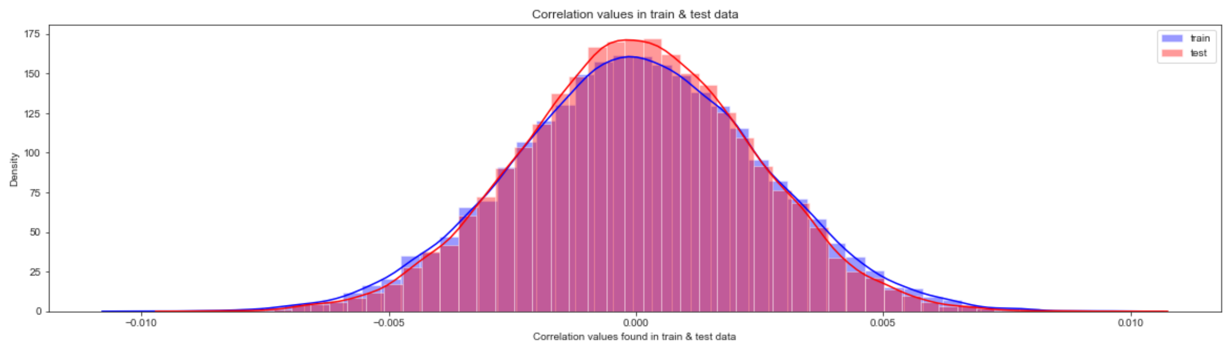
In [46]:
```
test_corr=df_santander_test[num_test].corr()
```

In [47]:
```
test_corr = test_corr.values.flatten()
```

In [48]:
```
test_corr=test_corr[test_corr!=1]
```

In [49]:
```
plt2.figure(figsize=(20,5))
sns.distplot(train_corr,color="blue",label="train")
sns.distplot(test_corr,color="red",label="test")
plt2.xlabel("Correlation values found in train & test data")
plt2.ylabel("Density")
plt2.title ("Correlation values in train & test data")
plt2.legend()
```

Out[49]: <matplotlib.legend.Legend at 0x19f97791148>



Feature Importance

In [50]:
```
X = df_santander.drop(columns=['ID_code', 'target'], axis=1)
y = df_santander['target']
```

In [51]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,random_state=42)
```

In [52]:
```
rf_model=RandomForestClassifier(n_estimators=10,random_state=42)
rf_model.fit(X_test,y_test)
```

Out[52]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=42, verbose=
0,
                       warm_start=False)
```

In [53]:
```
importance = pd.DataFrame(rf_model.feature_importances_, columns = ['Feature Impo
```

In [54]:
```
columns = pd.DataFrame(data=X.columns.values);
```

In [55]:
```
columns['imporatance'] = importance
```

In [56]:
```
columns = columns.rename(columns={0: "Variable"})
```

In [57]:
```
columns = columns.rename(columns={'imporatance':'importance'})
```

In [58]:
```
columns.sort_values(by=['importance'], inplace=True)
```

In [59]:
```
columns
```

Out[59]:

	Variable	importance
30	var_30	0.003018
27	var_27	0.003184
72	var_72	0.003242
38	var_38	0.003261
3	var_3	0.003287
...	...	...
109	var_109	0.009395
80	var_80	0.009492
53	var_53	0.009571
139	var_139	0.010192
81	var_81	0.013020

200 rows × 2 columns

In [60]:
```
# Var_81 most important
```

In [61]:
```
X=df_santander.drop(['ID_code','target'],axis=1)
y=df_santander['target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, rando
```

In [62]:
```
sm = SMOTE(random_state=42)
X_smote,y_smote=sm.fit_sample(X_train,y_train)
X_smote_v,y_smote_v=sm.fit_sample(X_test,y_test)
```

In [63]:
```
x = pd.concat([X_smote,y_smote],axis=1)
```

In [64]:
```
y = pd.concat([X_smote_v,y_smote_v], axis=1)
```

In [65]:
```
xy = pd.concat([x,y],axis=0)
```

In [66]:
```
X_train.head()
```

Out[66]:

	var_0	var_1	var_2	var_3	var_4	var_5	var_6	var_7	var_8	var_9	va
23595	8.6599	2.3606	8.9668	4.7867	7.1358	0.1176	6.5557	12.4706	3.3068	6.5320	-4.:
83339	12.6858	-5.0178	8.4828	6.1694	9.7005	-16.9539	6.1838	21.1163	3.2714	8.8544	1.(
158960	15.5542	-7.6605	12.9832	5.3324	7.5846	0.2431	4.0529	21.6316	4.0790	6.1170	-1.:
94374	7.9124	-0.1867	12.5261	10.8331	10.5677	-15.0974	5.4738	20.2226	-1.2281	6.7459	6.:
16546	17.2725	-8.2606	8.1404	7.9506	8.7911	2.8503	4.2706	15.2856	3.2672	7.3236	1.!

Feature Scaling

In [67]:
```
from sklearn.preprocessing import StandardScaler
```

In [68]:
```
X_train = StandardScaler().fit_transform(X_train)
```

In [69]:
```
X_test = StandardScaler().fit_transform(X_test)
```

PCA

In [70]:
```
from sklearn.preprocessing import StandardScaler
```

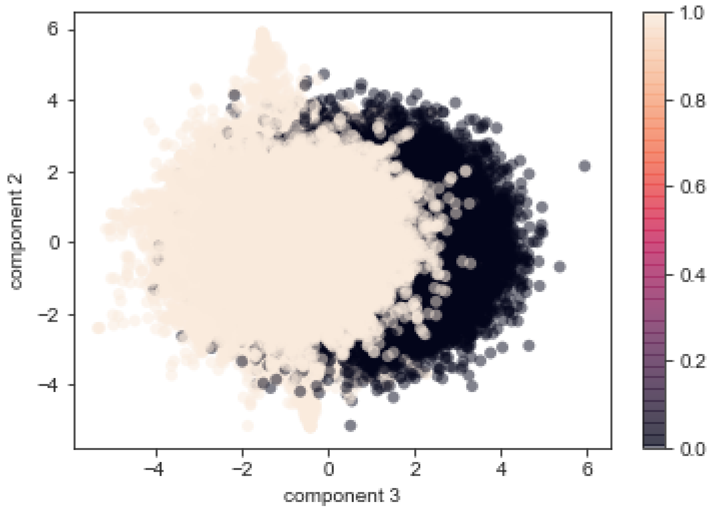
In [71]:
```
x = StandardScaler().fit_transform(xy.drop(['target'],axis=1))
```

In [72]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=170)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents)
print(sum(pca.explained_variance_))
print(sum(pca.explained_variance_ratio_))
```

```
172.6247411126679
0.8631213066914379
```

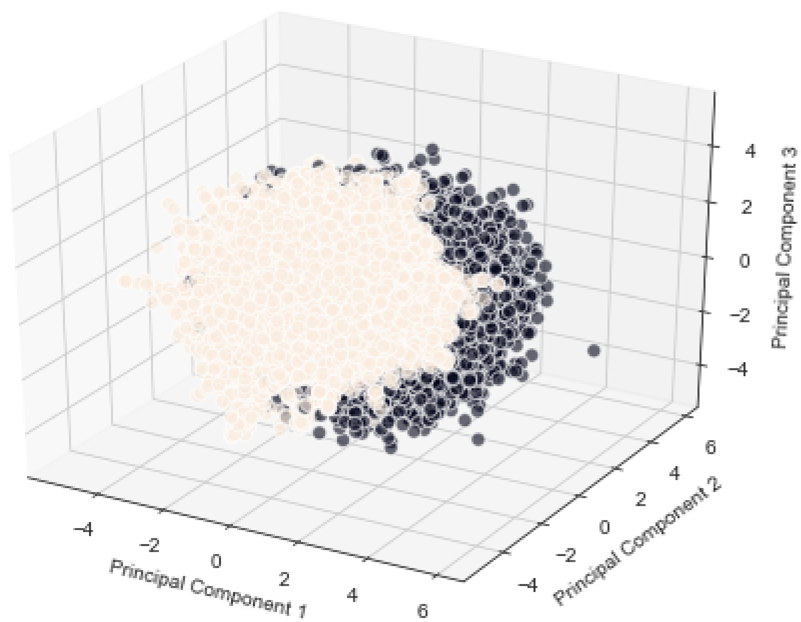
In [73]:
```
X1 = principalDf
y1 = xy['target']
X_train_PC,X_test_PC,y_train_PC,y_test_PC=train_test_split(X1,y1,random_state=42
```

In [74]:
```
plt2.scatter(principalDf.iloc[:, 0], principalDf.iloc[:, 1],
             c=xy['target'], edgecolor='none', alpha=0.5)
plt2.xlabel('component 3')
plt2.ylabel('component 2')
plt2.colorbar();
```



In [75]:
```
fig = plt2.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

xs = principalDf.iloc[:,0]
ys = principalDf.iloc[:,1]
zs = principalDf.iloc[:,2]
# size = list(df_santander['target'])
ax.scatter(xs, ys, zs, alpha=0.6, edgecolors='w',c=xy['target'],s=50)

ax.set_xlabel('Principal Component 1')
ax.set_ylabel('Principal Component 2')
ax.set_zlabel('Principal Component 3')
```

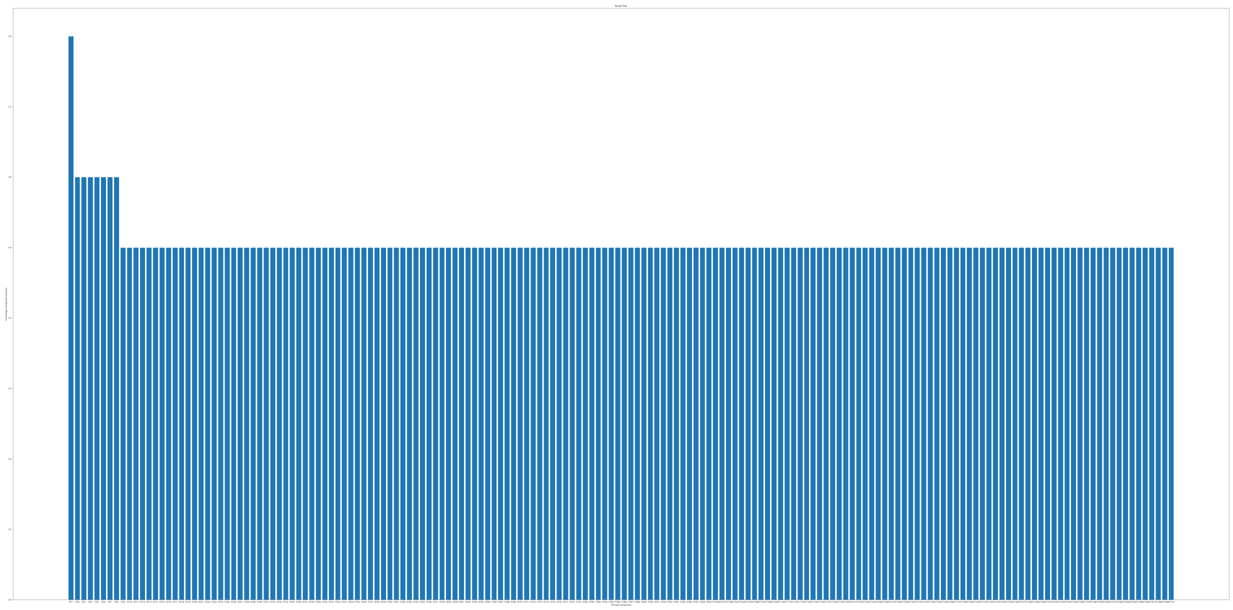
Out[75]:
```
Text(0.5, 0, 'Principal Component 3')
```



In [76]:
```
per_var = np.round(pca.explained_variance_ratio_*100, decimals=1)
```

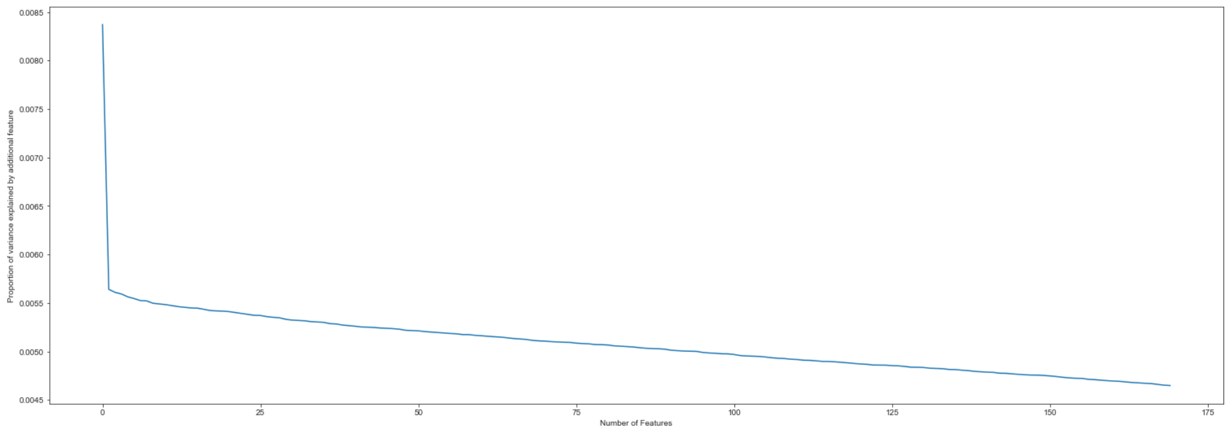
In [77]:
```
labels = ['PC'+str(x) for x in range(1,len(per_var)+1)]
```

In [78]:
```
fig= plt2.figure(figsize=(100,50))
plt2.bar(x=range(1,len(per_var)+1), height=per_var, tick_label=labels)
plt2.ylabel('Percentage of Explained Variance')
plt2.xlabel('Principal Component')
plt2.title('Scree Plot')
plt2.show()
```



In [79]:
```
plt2.figure(figsize=(26,9))
plt2.plot(pca.explained_variance_ratio_)
# plt2.xticks(range(80))
plt2.xlabel("Number of Features")
plt2.ylabel("Proportion of variance explained by additional feature")
```

Out[79]:
```
Text(0, 0.5, 'Proportion of variance explained by additional feature')
```



Model

In [80]:
```
def draw_confusion_mx(y_test,y_pred):
    print('\n######### Confusion Matrix #########\n')
    cm=pd.crosstab(y_test,y_pred)
    print(cm)

def draw_classification_report(y_test,y_pred):
    print('\n######### Classification Report #########\n')
    print(classification_report(y_test,y_pred))

def draw_roc_auc(y_test,y_pred):  ##y_pred in form of probabilities
    ns_probs = [0 for _ in range(len(y_test))]
    ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
    lr_fpr, lr_tpr, _ = roc_curve(y_test, y_pred)
    plt2.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
    plt2.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
    auc_score=auc(lr_fpr,lr_tpr)
    plt2.title('ROC(area=%0.3f)' %auc_score)

    plt2.xlabel('False Positive Rate')
    plt2.ylabel('True Positive Rate')

    plt2.legend()

    plt2.show()

def draw_precision_recall(y_test,y_pred):
    precision, recall, _ = precision_recall_curve(y_test, y_pred)

    # In matplotlib < 1.5, plt.fill_between does not have a 'step' argument
    step_kwargs = ({'step': 'post'}
                   if 'step' in signature(plt2.fill_between).parameters
                   else {})
    plt2.step(recall, precision, color='b', alpha=0.2,
         where='post')
    plt2.fill_between(recall, precision, alpha=0.2, color='b', **step_kwargs)

    plt2.xlabel('Recall')
    plt2.ylabel('Precision')
    plt2.ylim([0.0, 1.05])
    plt2.xlim([0.0, 1.0])
    plt2.title(' Precision-Recall curve: PR_AUC={0:0.3f}'.format( auc(recall, pr
    plt2.show()
```

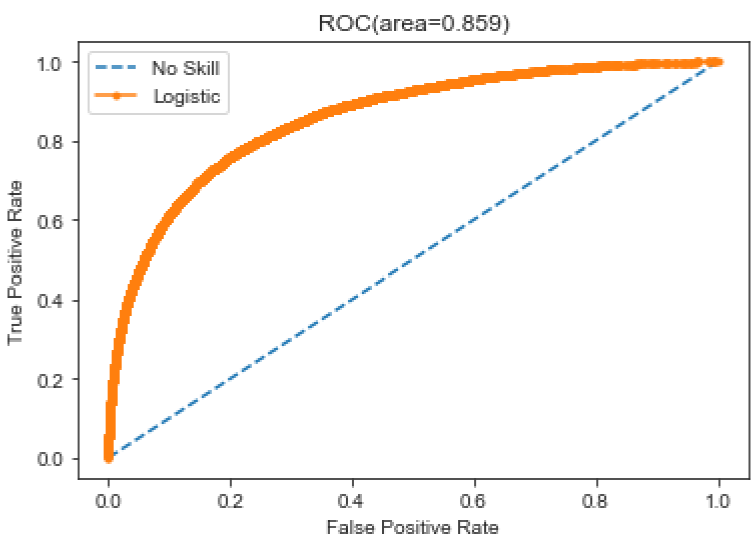
In [81]:
```
def fit_N_predict(model,X_train,X_test,y_train,y_test,model_code,testData,PCA=0):

    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    y_pred2 = model.predict_proba(X_test)
    y_pred2 = y_pred2[:,1]

    draw_confusion_mx(y_test,y_pred)

    draw_classification_report(y_test,y_pred)

    draw_roc_auc(y_test,y_pred2)

    draw_precision_recall(y_test,y_pred2)
    if(PCA == 0):
        if(model_code!="XGB"):
            print('\n\nModel performance on test data:\n',)
            print(model.predict(testData.drop(['ID_code'],axis=1)))
        else:
            print('\n\nModel performance on test data:\n',)
            print(model.predict(testData.drop(['ID_code'],axis=1).values))
```

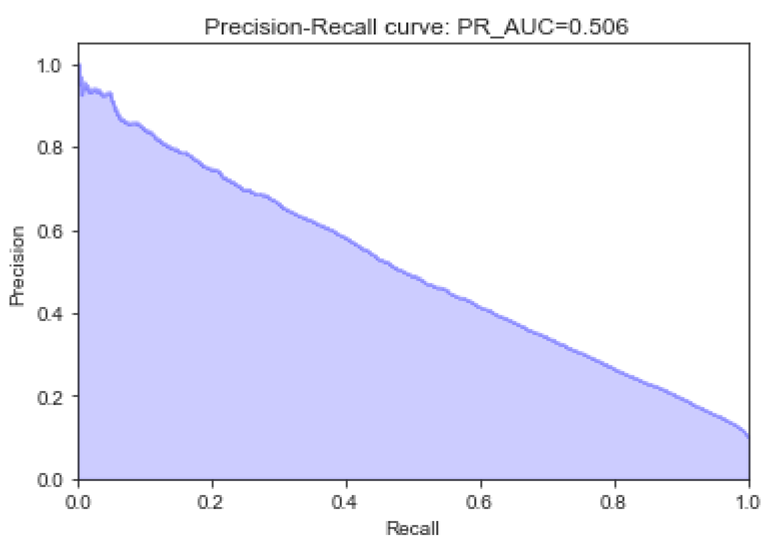
Logistic Regression Model

In [173]:
```
lr_model=LogisticRegression(random_state=42,class_weight = 'balanced')
```

In [174]:
```
print("LOGISTIC REGRESSION ON ORIGINAL DATASET\n\n")
fit_N_predict(lr_model,X_train,X_test,y_train,y_test,model_code='LR',testData=df_
```

```
######### Confusion Matrix #########

col_0       0      1
target
0       42282  11669
1        1375   4674

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.97      0.78      0.87     53951
           1       0.29      0.77      0.42      6049

    accuracy                           0.78     60000
   macro avg       0.63      0.78      0.64     60000
weighted avg       0.90      0.78      0.82     60000
```



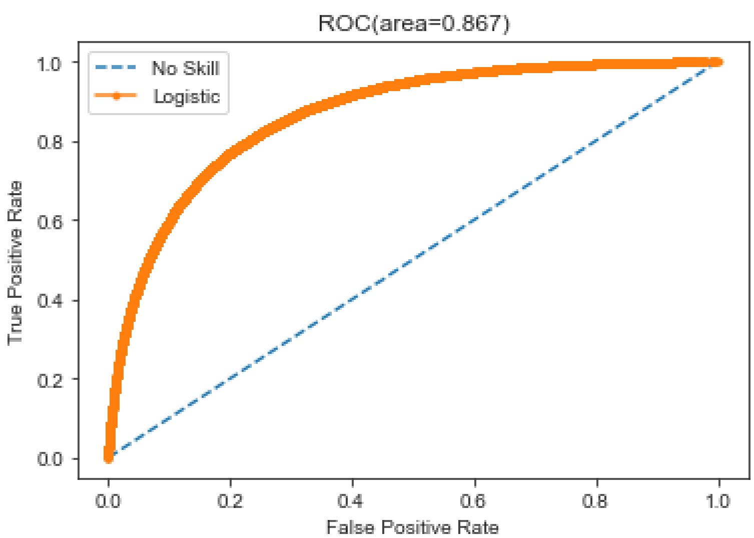


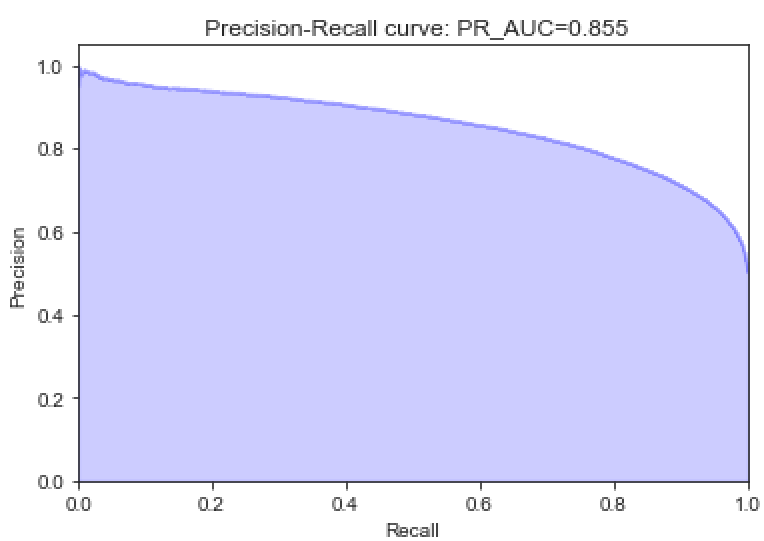
```
Model performance on test data:

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

Logistic Regression after applying SMOTE

In [149]:
```
print("LOGISTIC REGRESSION SMOTE DATASET\n\n")
fit_N_predict(lr_model,X_smote,X_smote_v,y_smote,y_smote_v,model_code='LR',testDa
```

```
######### Confusion Matrix #########

col_0       0      1
target
0       42258  11693
1       11565  42386

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.79      0.78      0.78     53951
           1       0.78      0.79      0.78     53951

    accuracy                           0.78    107902
   macro avg       0.78      0.78      0.78    107902
weighted avg       0.78      0.78      0.78    107902
```



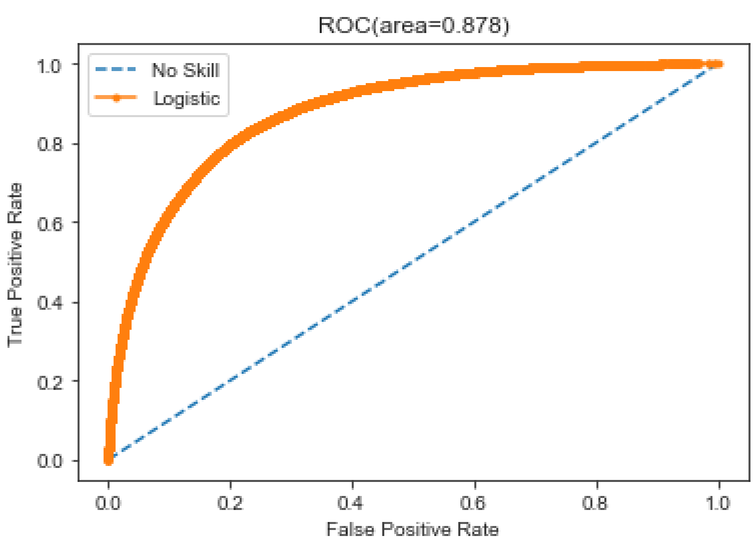


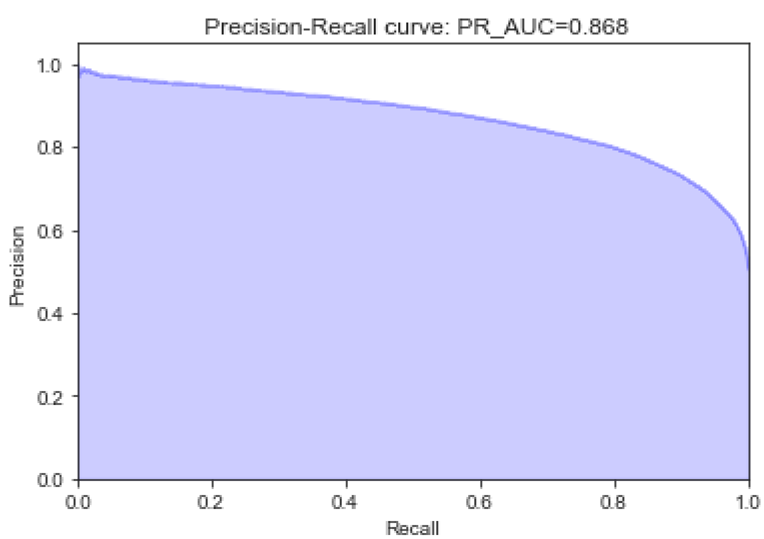
```
Model performance on test data:

[1 1 0 ... 0 0 1]
```

LR on SMOTE dataset and PCA

In [152]:
```
print("LOGISTIC REGRESSION ON PCA+SMOTE DATASET\n\n")
fit_N_predict(lr_model,X_train_PC,X_test_PC,y_train_PC,y_test_PC,model_code='LR'
```

```
######### Confusion Matrix #########

col_0       0      1
target
0       35480   9342
1        8802  36327

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.80      0.79      0.80     44822
           1       0.80      0.80      0.80     45129

    accuracy                           0.80     89951
   macro avg       0.80      0.80      0.80     89951
weighted avg       0.80      0.80      0.80     89951
```



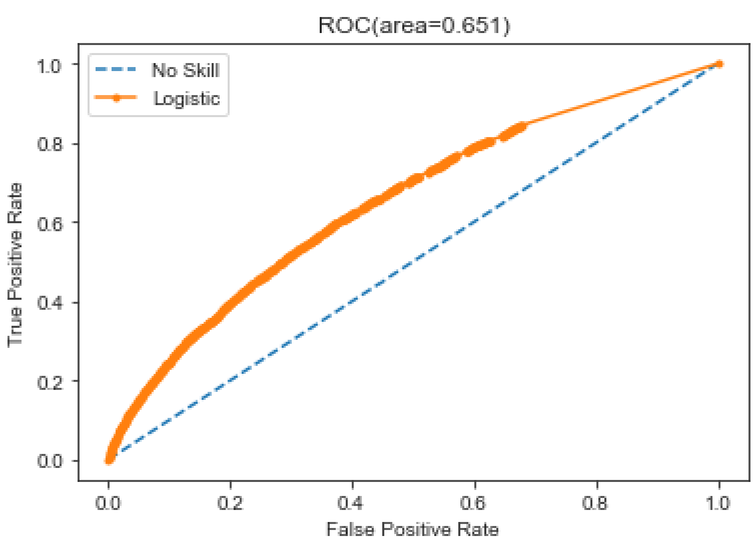


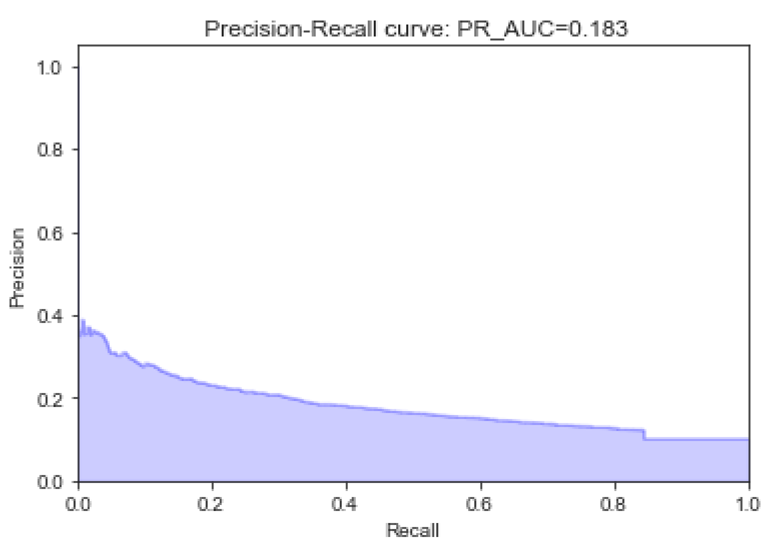
Decision Tree

In [153]:
```
tree_clf = DecisionTreeClassifier(class_weight='balanced', random_state = 2019,
                                  max_features = 0.7, min_samples_leaf = 80)
```

In [154]:
```
print("DECISION TREE ON ORIGINAL DATASET\n\n")
fit_N_predict(tree_clf,X_train,X_test,y_train,y_test,model_code='DT',testData=df_
```

```
######### Confusion Matrix #########

col_0       0      1
target
0       35288  18663
1        2631   3418

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.93      0.65      0.77     53951
           1       0.15      0.57      0.24      6049

    accuracy                           0.65     60000
   macro avg       0.54      0.61      0.51     60000
weighted avg       0.85      0.65      0.72     60000
```



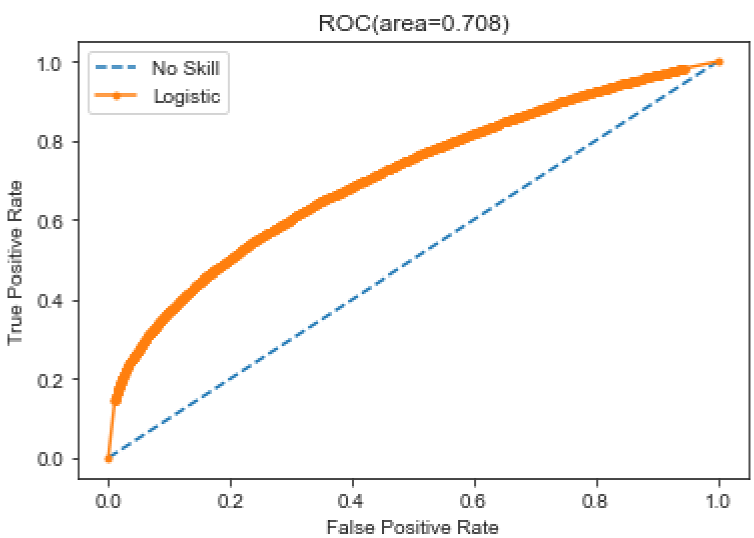


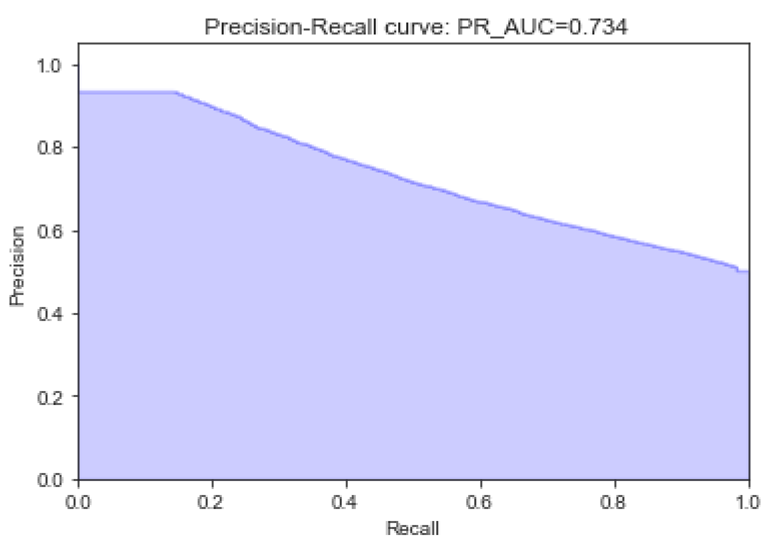
```
Model performance on test data:

[0 0 0 ... 0 1 0]
```

Decision Tree after applying SMOTE

In [155]:
```
print("DECISION TREE ON SMOTE DATASET\n\n")
fit_N_predict(tree_clf,X_smote,X_smote_v,y_smote,y_smote_v,model_code='DT',testD
```

```
######### Confusion Matrix #########

col_0       0      1
target
0       39740  14211
1       23352  30599

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.63      0.74      0.68     53951
           1       0.68      0.57      0.62     53951

    accuracy                           0.65    107902
   macro avg       0.66      0.65      0.65    107902
weighted avg       0.66      0.65      0.65    107902
```



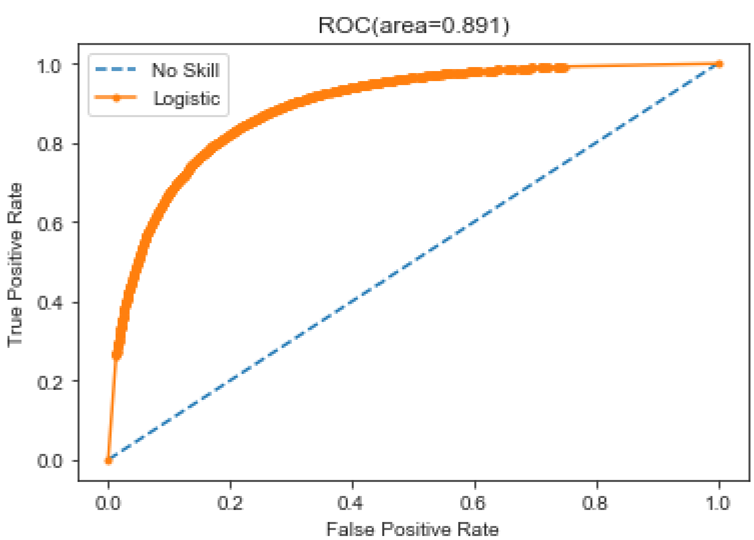


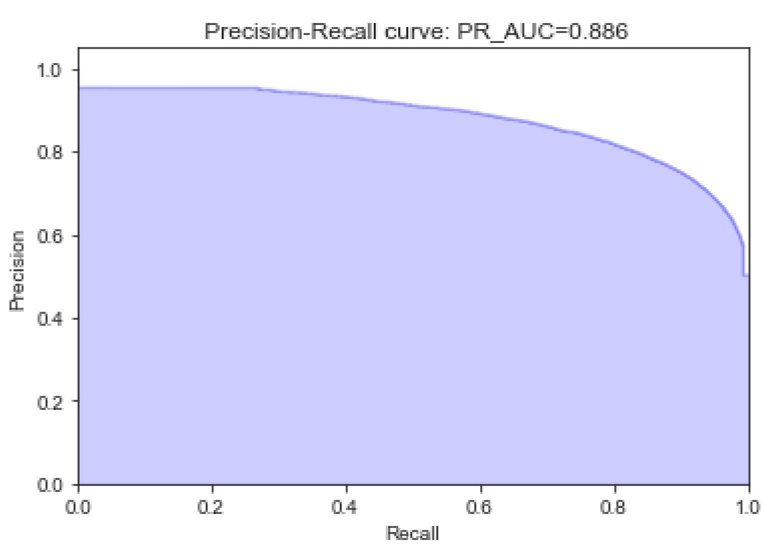
```
Model performance on test data:

[0 0 1 ... 1 0 1]
```

DT + SMOTE + PCA

In [158]:
```
print("DECISION TREE ON PCA+SMOTE DATASET\n\n")
fit_N_predict(tree_clf,X_train_PC,X_test_PC,y_train_PC,y_test_PC,model_code='DT'
```

```
######### Confusion Matrix #########

col_0       0      1
target
0       36194   8628
1        8386  36743

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.81      0.81      0.81     44822
           1       0.81      0.81      0.81     45129

    accuracy                           0.81     89951
   macro avg       0.81      0.81      0.81     89951
weighted avg       0.81      0.81      0.81     89951
```



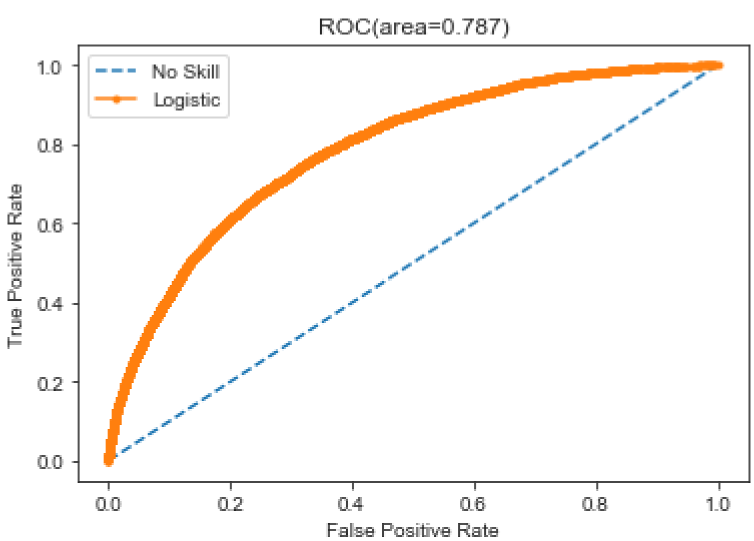


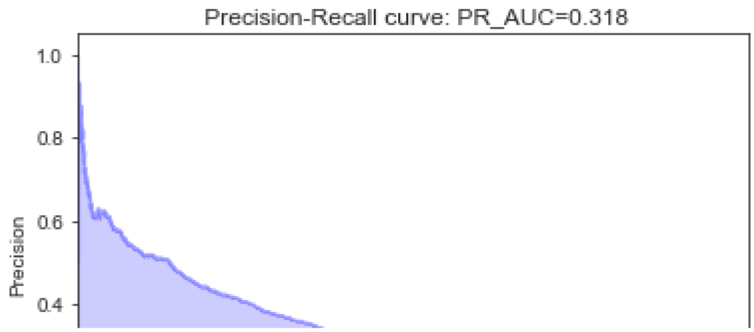
Random Forest

In [159]:
```
random_forest = RandomForestClassifier(n_estimators=100, random_state=2019, verb
                                       class_weight='balanced', max_features = 0.
                                        min_samples_leaf = 100,n_jobs=-1)
```

In [160]:
```
print("RANDOM FOREST ON ORIGINAL DATASET\n\n")
fit_N_predict(random_forest,X_train,X_test,y_train,y_test,model_code='RF',testDa
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:  7.2min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 16.5min finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.6s finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.6s finished


######### Confusion Matrix #########

col_0       0     1
target
0       46483  7468
1        2970  3079

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.94      0.86      0.90     53951
           1       0.29      0.51      0.37      6049

    accuracy                           0.83     60000
   macro avg       0.62      0.69      0.64     60000
weighted avg       0.87      0.83      0.85     60000
```

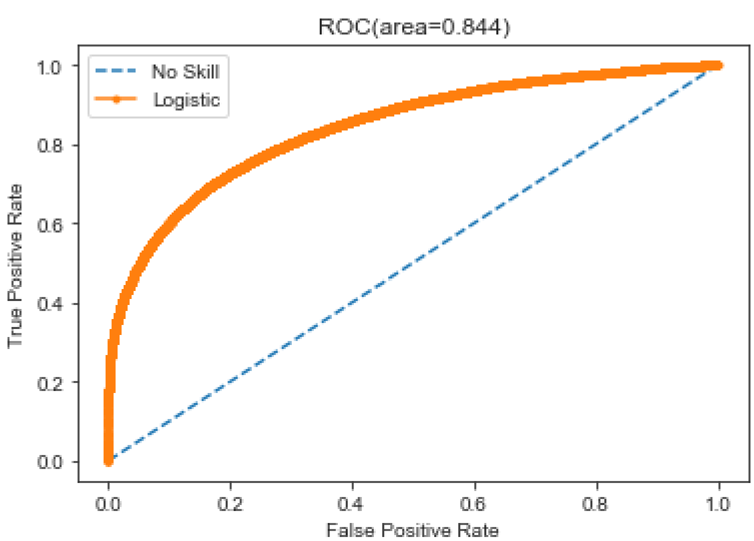


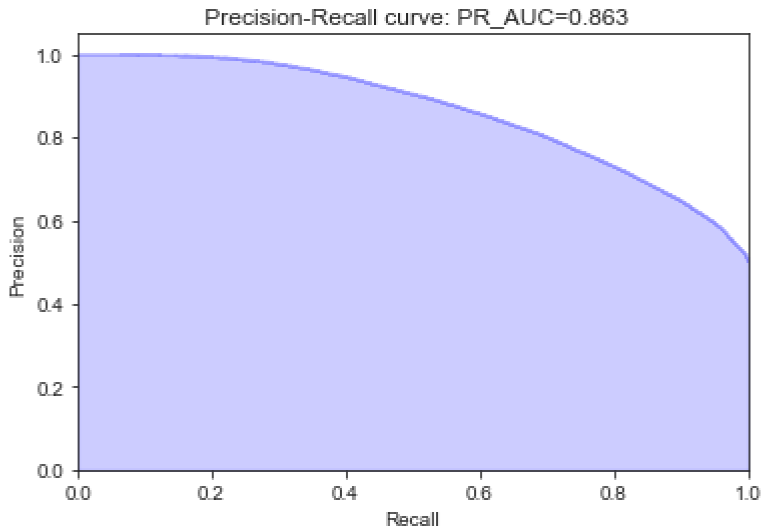



```
Model performance on test data:

[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.4s

[1 1 1 ... 0 1 1]

[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.9s finished
```

In [161]:
```
print("RANDOM FOREST ON SMOTE DATASET\n\n")
fit_N_predict(random_forest,X_smote,X_smote_v,y_smote,y_smote_v,model_code='RF',
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed: 16.7min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 39.5min finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.5s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    1.5s finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    1.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    2.1s finished


######### Confusion Matrix #########

col_0       0      1
target
0       47641   6310
1       20146  33805

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.70      0.88      0.78     53951
           1       0.84      0.63      0.72     53951

    accuracy                           0.75    107902
   macro avg       0.77      0.75      0.75    107902
weighted avg       0.77      0.75      0.75    107902
```



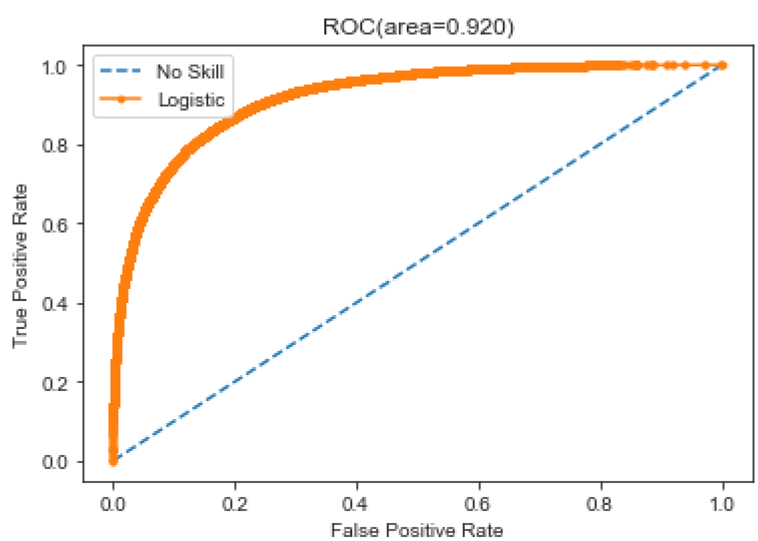


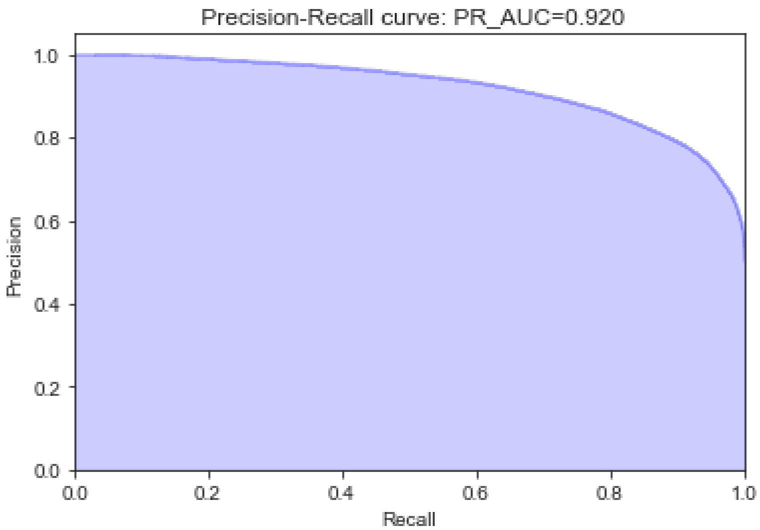

```
Model performance on test data:

[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    1.0s

[0 1 0 ... 0 0 0]

[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    2.8s finished
```

RF + SMOTE + PCA

In [162]:
```
print("RANDOM FOREST ON PCA+SMOTE DATASET\n\n")
fit_N_predict(random_forest,X_train_PC,X_test_PC,y_train_PC,y_test_PC,model_code
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed: 18.2min
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed: 42.3min finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.5s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    1.3s finished
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.5s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    1.3s finished


######### Confusion Matrix #########

col_0       0      1
target
0       37035   7787
1        7010  38119

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.84      0.83      0.83     44822
           1       0.83      0.84      0.84     45129

    accuracy                           0.84     89951
   macro avg       0.84      0.84      0.84     89951
weighted avg       0.84      0.84      0.84     89951
```



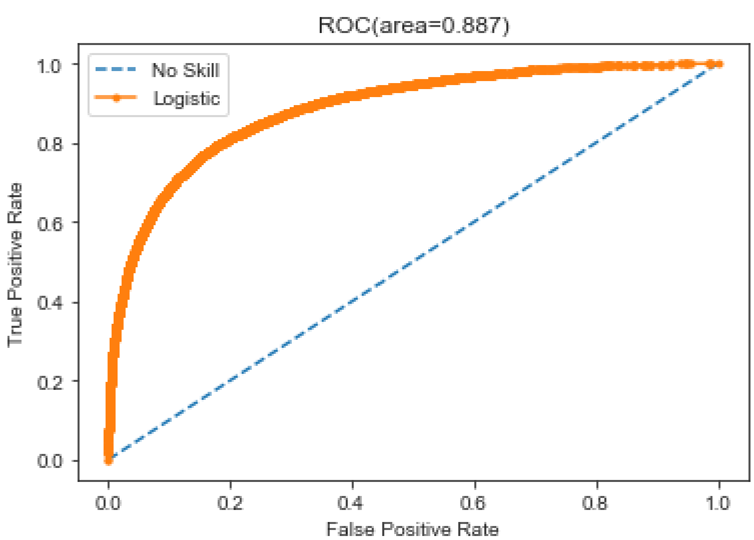


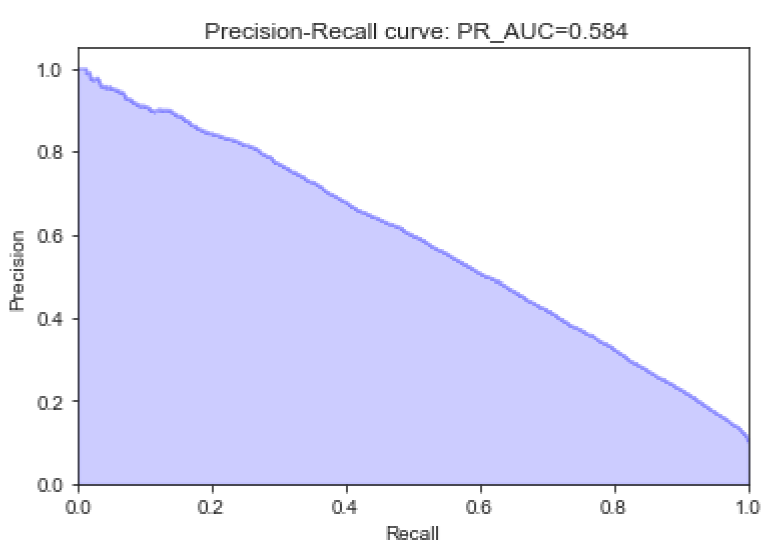
NaiveBayes

In [163]:
```
from sklearn.naive_bayes import GaussianNB
NB_model = GaussianNB()
```

In [164]:
```
print("NAIVE BAYES ON ORIGINAL DATASET\n\n")
fit_N_predict(NB_model,X_train,X_test,y_train,y_test,model_code='NB',testData=df_
```

```
######### Confusion Matrix #########

col_0       0     1
target
0       53077   874
1        3857  2192

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.93      0.98      0.96     53951
           1       0.71      0.36      0.48      6049

    accuracy                           0.92     60000
   macro avg       0.82      0.67      0.72     60000
weighted avg       0.91      0.92      0.91     60000
```

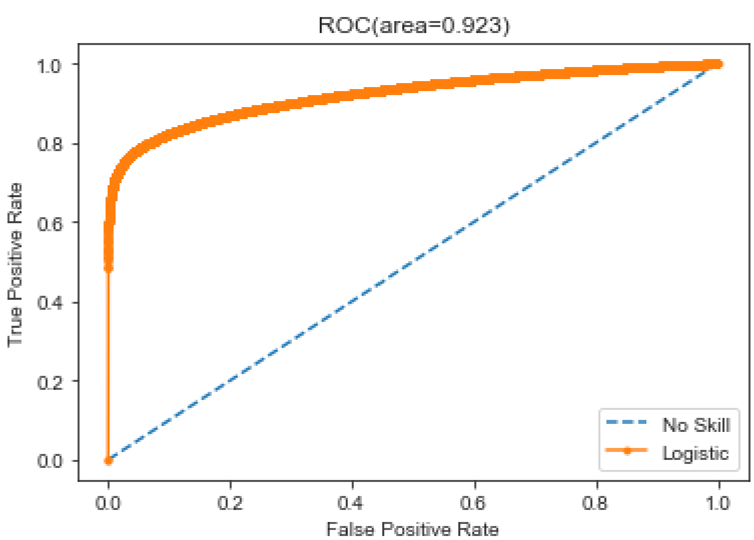


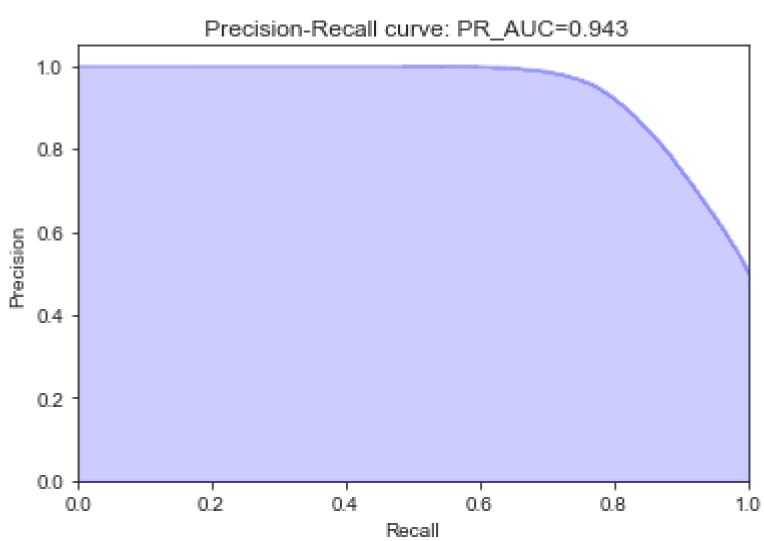


```
Model performance on test data:

[1 1 1 ... 1 1 1]
```

In [165]:
```
print("NAIVE BAYES ON SMOTE DATASET\n\n")
fit_N_predict(NB_model,X_smote,X_smote_v,y_smote,y_smote_v,model_code='NB',testDa
```

```
######### Confusion Matrix #########

col_0       0      1
target
0       51757   2194
1       12190  41761

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.81      0.96      0.88     53951
           1       0.95      0.77      0.85     53951

    accuracy                           0.87    107902
   macro avg       0.88      0.87      0.87    107902
weighted avg       0.88      0.87      0.87    107902
```

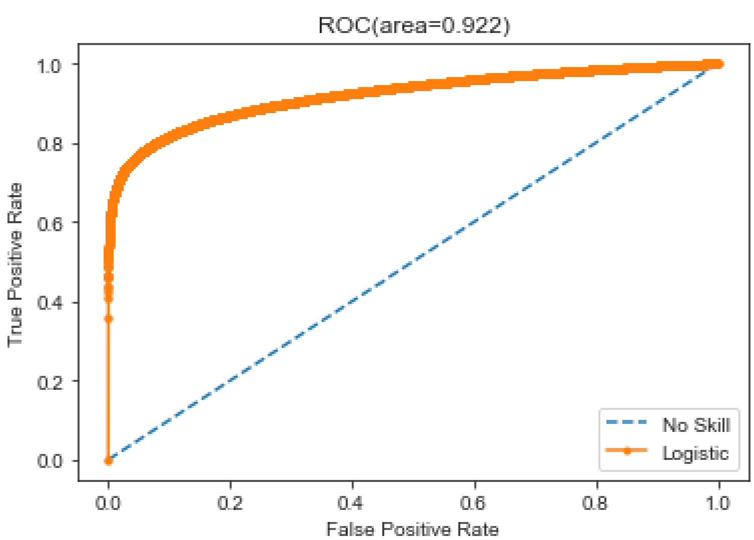


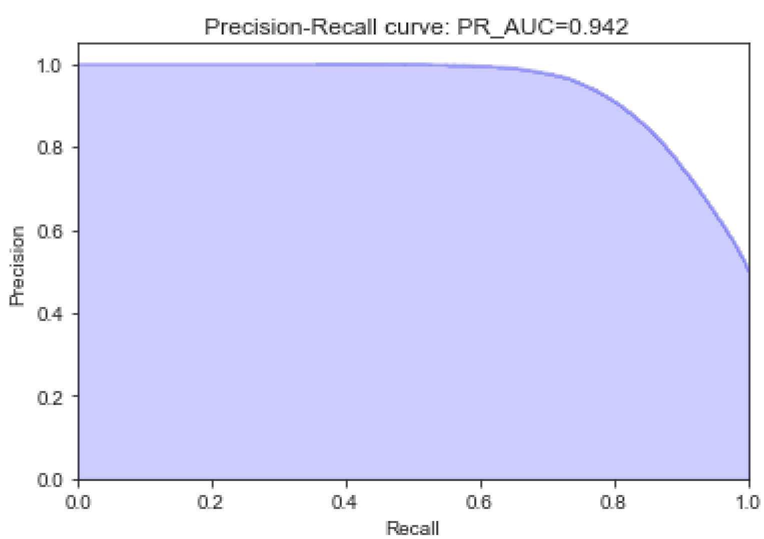


```
Model performance on test data:

[0 0 0 ... 0 0 1]
```

In [166]:
```
print("NAIVE BAYES ON PCA+SMOTE DATASET\n\n")
fit_N_predict(NB_model,X_train_PC,X_test_PC,y_train_PC,y_test_PC,model_code='NB'
```

```
######### Confusion Matrix #########

col_0       0      1
target
0       42001   2821
1        9757  35372

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.81      0.94      0.87     44822
           1       0.93      0.78      0.85     45129

    accuracy                           0.86     89951
   macro avg       0.87      0.86      0.86     89951
weighted avg       0.87      0.86      0.86     89951
```





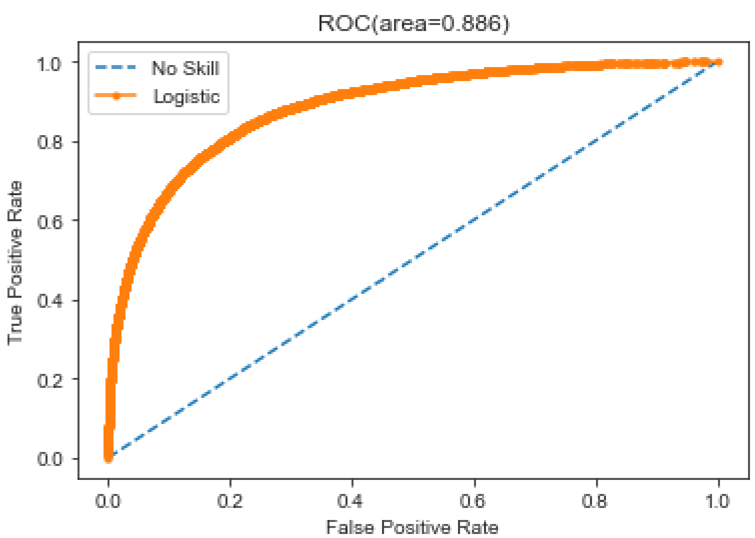
XGBoost

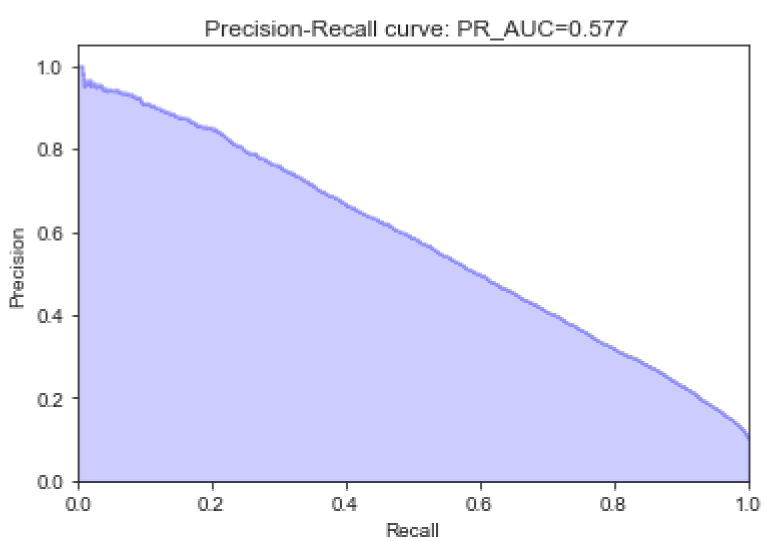
In [88]:
```
from xgboost import XGBClassifier
```

In [89]:
```
XGB = XGBClassifier(learning_rate =0.1,
 n_estimators=800,
 max_depth=5,
 min_child_weight=1,
 gamma=0,
 subsample=0.8,
 colsample_bytree=0.8,
 objective= 'binary:logistic',
 nthread=4,
 seed=27,scale_pos_weight=2)
```

In [90]:
```
print("XGBOOST CLASSIFIER ON ORIGINAL DATASET\n\n")
fit_N_predict(XGB,X_train,X_test,y_train,y_test,model_code='XGB',testData=df_san
```

```
######### Confusion Matrix #########

col_0       0     1
target
0       52794  1157
1        3672  2377

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.93      0.98      0.96     53951
           1       0.67      0.39      0.50      6049

    accuracy                           0.92     60000
   macro avg       0.80      0.69      0.73     60000
weighted avg       0.91      0.92      0.91     60000
```

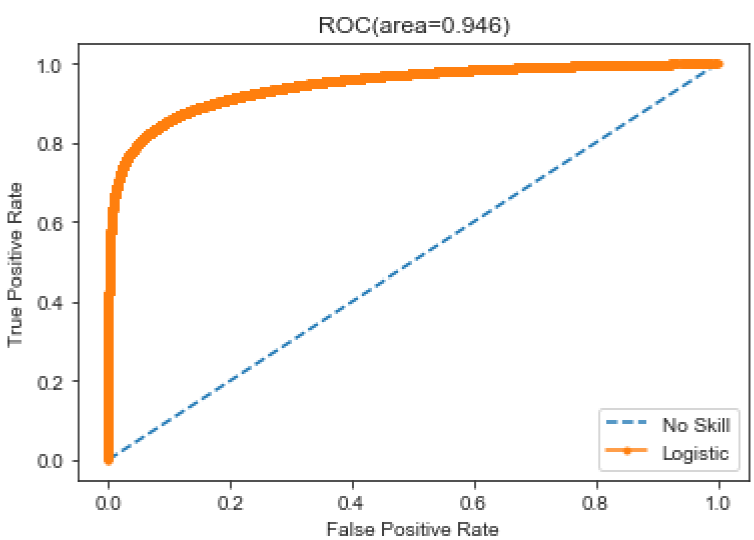


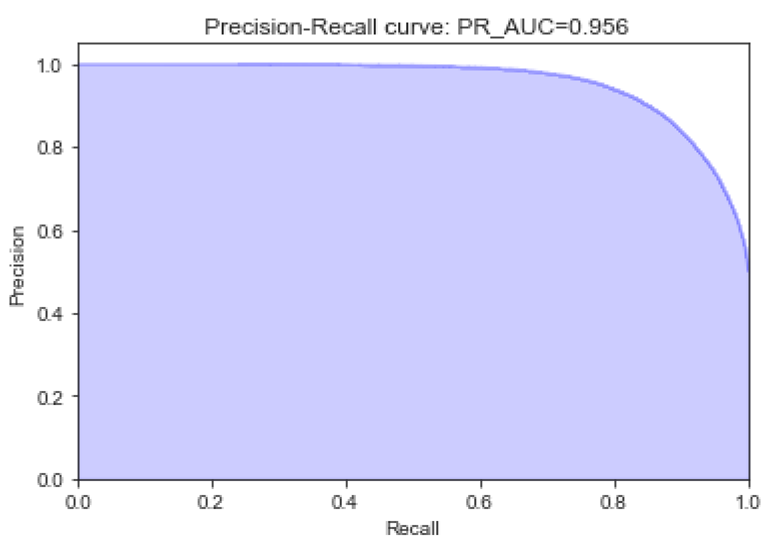


```
Model performance on test data:

[1 1 1 ... 1 1 1]
```

In [91]:
```
print("XGBOOST CLASSIFIER ON SMOTE DATASET\n\n")
fit_N_predict(XGB,X_smote,X_smote_v,y_smote,y_smote_v,model_code='XGB_SM',testDat
```

```
######### Confusion Matrix #########

col_0       0      1
target
0       47976   5975
1        7324  46627

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.87      0.89      0.88     53951
           1       0.89      0.86      0.88     53951

    accuracy                           0.88    107902
   macro avg       0.88      0.88      0.88    107902
weighted avg       0.88      0.88      0.88    107902
```





```
Model performance on test data:

[0 0 0 ... 0 0 1]
```

In [92]:
```
print("XGBOOST CLASSIFIER ON SMOTE ON PCA DATASET\n\n")
fit_N_predict(XGB,X_train_PC,X_test_PC,y_train_PC,y_test_PC,model_code='XGB',tes
```

```
######### Confusion Matrix #########

col_0       0      1
target
0       38121   6701
1        2800  42329

######### Classification Report #########

              precision    recall  f1-score   support

           0       0.93      0.85      0.89     44822
           1       0.86      0.94      0.90     45129

    accuracy                           0.89     89951
   macro avg       0.90      0.89      0.89     89951
weighted avg       0.90      0.89      0.89     89951
```