WADI Attack Detection

June 16, 2020

1 Detecting Attack/Intrusion in Plant using Machine Learning

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```
# Let's predefine some libraries that will be used in the notebook. Although, they wil
# positions but this cell is defined to provide information about used libraries.

from scipy import stats

from sklearn.decomposition import PCA
from mpl_toolkits import mplot3d

import datetime
from datetime import datetime
import matplotlib.dates as mpldates

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

from sklearn.svm import OneClassSVM
from sklearn.ensemble import IsolationForest
from sklearn.covariance import EllipticEnvelope
```

In [3]:
```
#To set output to be shown in float format let's run below code
pd.set_option('display.float_format', lambda x: '%.3f' % x)
```

In [4]:
```
#Below code is to load the variable in which Normal Data is stored in NDA notebook
%store -r normal_data_sorted
```

In [4]:
```
#Below code is to load the variable in which Attack Data is stored in ADA notebook so
#we don't need to Re-run below lines of code.
%store -r data_sorted
```

1.0.1 Below line of code is to load the attack data in pandas dataframe.

In [5]:
```
attack_data = pd.read_excel('./WADI_attackdata.xlsx')
```

1.0.2 One-Class SVM trial on raw data

In [181]:
```python
from sklearn.svm import OneClassSVM
#clf = OneClassSVM(gamma='auto').fit(data_sorted)
#The above code is taking huge time to execute.
```

In [58]:
```python
# Now let's first try One class SVM on a smaller part of the dataset
clf = OneClassSVM(gamma='auto').fit(normal_data_sorted.iloc[0:10000,0:100])
```

In [60]:
```python
prediction = clf.predict(data_sorted.iloc[0:1000,0:100])
prediction
```

Out[60]:
```
array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
```

```
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1])
```

In [29]:
```
data_sorted.head()
```

Out[29]:
```
   1_AIT_001_PV  1_AIT_002_PV  1_AIT_003_PV  1_AIT_004_PV  1_AIT_005_PV  \
0        164.21      0.529486       11.9972        482.48      0.331167
1        164.21      0.529486       11.9972        482.48      0.331167
2        164.21      0.529486       11.9972        482.48      0.331167
3        164.21      0.529486       11.9972        482.48      0.331167
4        164.21      0.529486       11.9972        482.48      0.331167

   1_FIT_001_PV  1_LS_001_AL  1_LS_002_AL  1_LT_001_PV  1_MV_001_STATUS  \
0      0.001273            0            0       48.482                1
1      0.001273            0            0       48.482                1
2      0.001273            0            0       48.482                1
3      0.001273            0            0       48.482                1
4      0.001273            0            0       48.482                1

           ...            3_MV_001_STATUS  3_MV_002_STATUS  \
0          ...                          1                1
1          ...                          1                1
2          ...                          1                1
3          ...                          1                1
4          ...                          1                1

   3_MV_003_STATUS  3_P_001_STATUS  3_P_002_STATUS  3_P_003_STATUS  \
0                1               1               1               1
```

```
1              1              1              1              1
2              1              1              1              1
3              1              1              1              1
4              1              1              1              1

   3_P_004_STATUS  LEAK_DIFF_PRESSURE  PLANT_START_STOP_LOG  \
0               1             62.6226                     1
1               1             62.6226                     1
2               1             62.6226                     1
3               1             62.6226                     1
4               1             62.6226                     1

   TOTAL_CONS_REQUIRED_FLOW
0                      0.39
1                      0.39
2                      0.39
3                      0.39
4                      0.39

[5 rows x 123 columns]
```

1.0.3 Isolation Forest trial on raw data

```
In [20]: from sklearn.ensemble import IsolationForest

In [72]: clf = IsolationForest(contamination=0,random_state=0, behaviour='new').fit(data_sorte

In [74]: clf.predict(normal_data_sorted.iloc[0:1000,0:50])

Out[74]: array([ 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
                 1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
```

```
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
       -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,
        1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1,  1])
```

```
In [15]: data_sorted.memory_usage()
```

```
Out[15]: Index                            80
         1_AIT_001_PV               1382408
         1_AIT_003_PV               1382408
         1_AIT_004_PV               1382408
         1_AIT_005_PV               1382408
         1_FIT_001_PV               1382408
```

1_LS_001_AL	1382408
1_LS_002_AL	1382408
1_LT_001_PV	1382408
1_MV_001_STATUS	1382408
1_MV_002_STATUS	1382408
1_MV_003_STATUS	1382408
1_MV_004_STATUS	1382408
1_P_001_STATUS	1382408
1_P_002_STATUS	1382408
1_P_003_STATUS	1382408
1_P_004_STATUS	1382408
1_P_005_STATUS	1382408
1_P_006_STATUS	1382408
2_DPIT_001_PV	1382408
2_FIC_101_CO	1382408
2_FIC_101_PV	1382408
2_FIC_101_SP	1382408
2_FIC_201_CO	1382408
2_FIC_201_PV	1382408
2_FIC_201_SP	1382408
2_FIC_301_CO	1382408
2_FIC_301_PV	1382408
2_FIC_301_SP	1382408
2_FIC_401_CO	1382408
	...
2_SV_301_STATUS	1382408
2_SV_401_STATUS	1382408
2_SV_501_STATUS	1382408
2_SV_601_STATUS	1382408
2A_AIT_001_PV	1382408
2A_AIT_002_PV	1382408
2A_AIT_003_PV	1382408
2A_AIT_004_PV	1382408
2B_AIT_001_PV	1382408
2B_AIT_002_PV	1382408
2B_AIT_003_PV	1382408
2B_AIT_004_PV	1382408
3_AIT_001_PV	1382408
3_AIT_002_PV	1382408
3_AIT_003_PV	1382408
3_AIT_004_PV	1382408
3_AIT_005_PV	1382408
3_FIT_001_PV	1382408
3_LS_001_AL	1382408
3_LT_001_PV	1382408
3_MV_001_STATUS	1382408
3_MV_002_STATUS	1382408
3_MV_003_STATUS	1382408

```
3_P_001_STATUS              1382408
3_P_002_STATUS              1382408
3_P_003_STATUS              1382408
3_P_004_STATUS              1382408
LEAK_DIFF_PRESSURE          1382408
PLANT_START_STOP_LOG        1382408
TOTAL_CONS_REQUIRED_FLOW    1382408
Length: 123, dtype: int64
```

Ok, so now we had tried fitting two models on both our data but they are not showing satisfiable accuracy. Probably that is because of large dimension of data. So, let's try to reduce dimensions for better operability.

1.1 Dimensionality Reduction

Low Variance Filter Let's see if applying a filter on variance of features can help us in reducing dimensions.

```
In [68]: data_sorted.describe()
```

Out[68]:

	1_AIT_001_PV	1_AIT_002_PV	1_AIT_003_PV	1_AIT_004_PV	1_AIT_005_PV	\
count	172801.000	172795.000	172801.000	172801.000	172801.000	
mean	176.210	0.649	11.928	453.784	0.275	
std	18.669	0.352	0.139	18.863	0.038	
min	0.000	0.000	0.000	0.000	0.202	
25%	170.866	0.589	11.911	440.867	0.241	
50%	177.234	0.631	11.928	454.977	0.274	
75%	179.533	0.661	11.952	468.240	0.306	
max	634.492	6.000	12.110	484.871	0.351	

	1_FIT_001_PV	1_LS_001_AL	1_LS_002_AL	1_LT_001_PV	1_MV_001_STATUS	\
count	172801.000	172801.000	172801.000	172801.000	172801.000	
mean	0.543	0.000	0.000	55.540	1.274	
std	0.862	0.000	0.000	8.707	0.453	
min	0.001	0.000	0.000	37.002	0.000	
25%	0.001	0.000	0.000	47.830	1.000	
50%	0.001	0.000	0.000	55.933	1.000	
75%	1.872	0.000	0.000	62.489	2.000	
max	2.495	0.000	0.000	75.216	2.000	

	...	3_MV_001_STATUS	3_MV_002_STATUS	\
count	...	172801.000	172801.000	
mean	...	1.000	1.000	
std	...	0.000	0.000	
min	...	1.000	1.000	
25%	...	1.000	1.000	
50%	...	1.000	1.000	
75%	...	1.000	1.000	
max	...	1.000	1.000	

```
       3_MV_003_STATUS  3_P_001_STATUS  3_P_002_STATUS  3_P_003_STATUS  \
count       172801.000      172801.000      172801.000      172801.000
mean             1.000           1.000           1.000           1.000
std              0.000           0.000           0.000           0.000
min              1.000           1.000           1.000           1.000
25%              1.000           1.000           1.000           1.000
50%              1.000           1.000           1.000           1.000
75%              1.000           1.000           1.000           1.000
max              1.000           1.000           1.000           1.000

       3_P_004_STATUS  LEAK_DIFF_PRESSURE  PLANT_START_STOP_LOG  \
count      172801.000          172801.000             172801.000
mean            1.000              62.707                  1.000
std             0.000               6.059                  0.000
min             1.000              56.604                  1.000
25%             1.000              60.227                  1.000
50%             1.000              61.371                  1.000
75%             1.000              63.096                  1.000
max             1.000             141.175                  1.000

       TOTAL_CONS_REQUIRED_FLOW
count                172801.000
mean                      0.553
std                       0.460
min                       0.000
25%                       0.220
50%                       0.550
75%                       0.710
max                       2.330

[8 rows x 123 columns]
```

In [41]:
```
normal_data_sorted.describe()
```

Out[41]:
```
       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_AIT_001_PV  \
count                                          1048571.000
mean                                               167.182
std                                                 13.919
min                                                  0.000
25%                                                155.981
50%                                                162.614
75%                                                174.473
max                                                214.311

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_AIT_002_PV  \
count                                          1048559.000
mean                                                 0.622
```

```
std                                                  0.060
min                                                  0.000
25%                                                  0.583
50%                                                  0.625
75%                                                  0.661
max                                                  2.059

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_AIT_003_PV  \
count                                          1048571.000
mean                                                11.717
std                                                  0.172
min                                                  0.000
25%                                                 11.598
50%                                                 11.777
75%                                                 11.817
max                                                 12.013

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_AIT_004_PV  \
count                                          1048565.000
mean                                              493.427
std                                                18.054
min                                                 0.000
25%                                               483.170
50%                                               496.371
75%                                               505.307
max                                               526.529

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_AIT_005_PV  \
count                                          1048571.000
mean                                                 0.306
std                                                  0.049
min                                                  0.208
25%                                                  0.261
50%                                                  0.306
75%                                                  0.345
max                                                  0.422

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_FIT_001_PV  \
count                                          1048571.000
mean                                                 0.524
std                                                  0.852
min                                                  0.001
25%                                                  0.001
50%                                                  0.001
75%                                                  1.853
max                                                  2.077

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_LS_001_AL  \
```

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_LS_002_AL \
count	1048571.000
mean	0.000
std	0.000
min	0.000
25%	0.000
50%	0.000
75%	0.000
max	0.000

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_LT_001_PV \
count	1048571.000
mean	57.125
std	11.977
min	0.027
25%	48.828
50%	57.689
75%	63.212
max	100.217

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_MV_001_STATUS \
count	1048571.000
mean	1.270
std	0.448
min	0.000
25%	1.000
50%	1.000
75%	2.000
max	2.000

	...	\
count	...	
mean	...	
std	...	
min	...	
25%	...	
50%	...	
75%	...	
max	...	

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\3_MV_001_STATUS \
count	1048571.000
mean	1.000
std	0.000
min	1.000
25%	1.000
50%	1.000
75%	1.000
max	1.000

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\3_MV_002_STATUS \
count	1048571.000
mean	1.000
std	0.000
min	1.000
25%	1.000
50%	1.000
75%	1.000
max	1.000

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\3_MV_003_STATUS \
count	1048571.000
mean	1.000
std	0.000
min	1.000
25%	1.000
50%	1.000
75%	1.000
max	1.000

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\3_P_001_STATUS \
count	1048571.000
mean	1.000
std	0.000
min	1.000
25%	1.000
50%	1.000
75%	1.000
max	1.000

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\3_P_002_STATUS \
count	1048571.000
mean	1.000
std	0.000
min	1.000
25%	1.000
50%	1.000

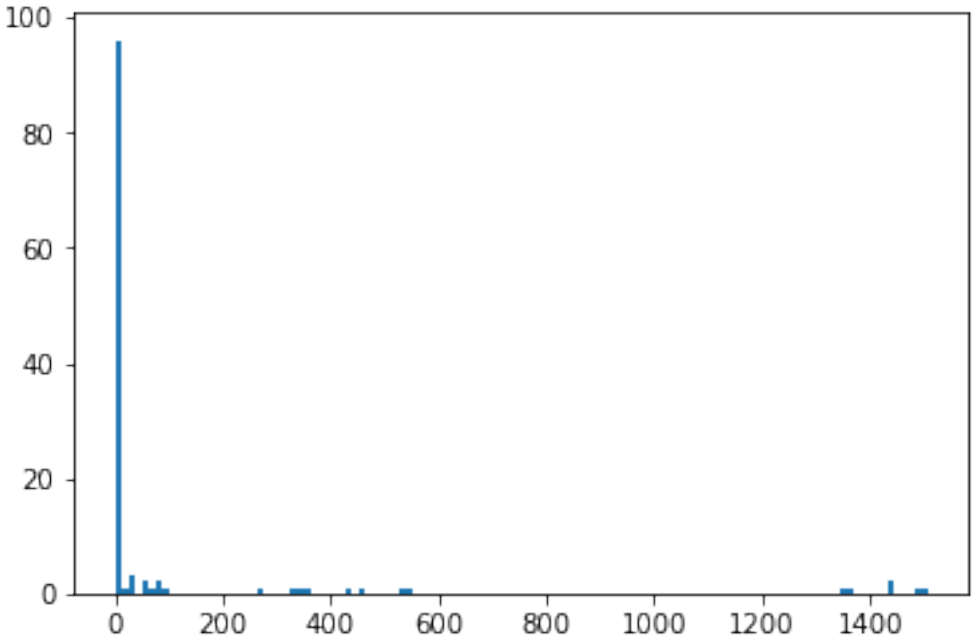
```
75%                                                 1.000
max                                                 1.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\3_P_003_STATUS  \
count                                         1048571.000
mean                                                1.000
std                                                 0.000
min                                                 1.000
25%                                                 1.000
50%                                                 1.000
75%                                                 1.000
max                                                 1.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\3_P_004_STATUS  \
count                                         1048571.000
mean                                                1.000
std                                                 0.000
min                                                 1.000
25%                                                 1.000
50%                                                 1.000
75%                                                 1.000
max                                                 1.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\LEAK_DIFF_PRESSURE  \
count                                         1048571.000
mean                                             63.061
std                                               5.940
min                                              45.397
25%                                              60.530
50%                                              61.764
75%                                              63.619
max                                             147.295

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\PLANT_START_STOP_LOG  \
count                                         1048571.000
mean                                                1.000
std                                                 0.000
min                                                 1.000
25%                                                 1.000
50%                                                 1.000
75%                                                 1.000
max                                                 1.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\TOTAL_CONS_REQUIRED_FLOW
count                                         1048571.000
mean                                                0.548
std                                                 0.444
min                                                 0.000
```

```
25%                                                 0.220
50%                                                 0.530
75%                                                 0.710
max                                                 2.260

[8 rows x 123 columns]
```

```
In [80]: plt.hist(data_sorted.var().drop(['2B_AIT_002_PV','3_AIT_004_PV','2_DPIT_001_PV']),bins
```

```
Out[80]: (array([96.,  1.,  3.,  0.,  2.,  1.,  2.,  1.,  0.,  0.,  0.,  0.,  0.,
                 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,
                 1.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  1.,  0.,  0.,
                 0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                 0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  0.,  2.,  0.,  0.,
                 0.,  1.,  1.]),
          array([   0.        ,   12.56516271,   25.13032542,   37.69548813,
                   50.26065084,   62.82581355,   75.39097625,   87.95613896,
                  100.52130167,  113.08646438,  125.65162709,  138.2167898 ,
                  150.78195251,  163.34711522,  175.91227793,  188.47744064,
                  201.04260335,  213.60776606,  226.17292876,  238.73809147,
                  251.30325418,  263.86841689,  276.4335796 ,  288.99874231,
                  301.56390502,  314.12906773,  326.69423044,  339.25939315,
                  351.82455586,  364.38971856,  376.95488127,  389.52004398,
                  402.08520669,  414.6503694 ,  427.21553211,  439.78069482,
                  452.34585753,  464.91102024,  477.47618295,  490.04134566,
                  502.60650836,  515.17167107,  527.73683378,  540.30199649,
                  552.8671592 ,  565.43232191,  577.99748462,  590.56264733,
                  603.12781004,  615.69297275,  628.25813546,  640.82329817,
                  653.38846087,  665.95362358,  678.51878629,  691.083949  ,
                  703.64911171,  716.21427442,  728.77943713,  741.34459984,
                  753.90976255,  766.47492526,  779.04008797,  791.60525067,
                  804.17041338,  816.73557609,  829.3007388 ,  841.86590151,
                  854.43106422,  866.99622693,  879.56138964,  892.12655235,
                  904.69171506,  917.25687777,  929.82204047,  942.38720318,
                  954.95236589,  967.5175286 ,  980.08269131,  992.64785402,
                 1005.21301673, 1017.77817944, 1030.34334215, 1042.90850486,
                 1055.47366757, 1068.03883028, 1080.60399298, 1093.16915569,
                 1105.7343184 , 1118.29948111, 1130.86464382, 1143.42980653,
                 1155.99496924, 1168.56013195, 1181.12529466, 1193.69045737,
                 1206.25562008, 1218.82078278, 1231.38594549, 1243.9511082 ,
                 1256.51627091, 1269.08143362, 1281.64659633, 1294.21175904,
                 1306.77692175, 1319.34208446, 1331.90724717, 1344.47240988,
                 1357.03757259, 1369.60273529, 1382.167898  , 1394.73306071,
                 1407.29822342, 1419.86338613, 1432.42854884, 1444.99371155,
```

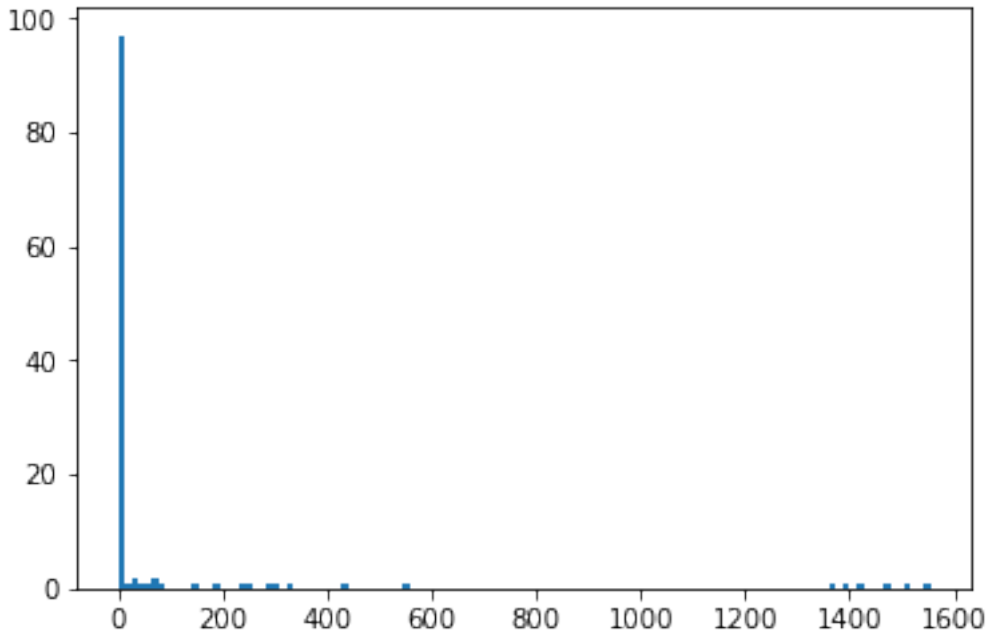
```
       1457.55887426, 1470.12403697, 1482.68919968, 1495.25436239,
       1507.81952509]),
 <a list of 120 Patch objects>)
```




```
In [109]: plt.hist(normal_data_sorted.var().drop(['\\\\WIN-25J4RO10SBF\\LOG_DATA\\SUTD_WADI\\L
```

```
Out[109]: (array([97.,  1.,  2.,  1.,  1.,  2.,  1.,  0.,  0.,  0.,  0.,  1.,  0.,
                  0.,  1.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  1.,  1.,  0.,  1.,
                  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,
                  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
                  0.,  1.,  0.,  1.,  0.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,
                  0.,  0.,  1.]),
           array([   0.        ,   12.96384886,   25.92769773,   38.89154659,
                    51.85539545,   64.81924431,   77.78309318,   90.74694204,
                   103.7107909 ,  116.67463977,  129.63848863,  142.60233749,
                   155.56618635,  168.53003522,  181.49388408,  194.45773294,
                   207.42158181,  220.38543067,  233.34927953,  246.3131284 ,
                   259.27697726,  272.24082612,  285.20467498,  298.16852385,
                   311.13237271,  324.09622157,  337.06007044,  350.0239193 ,
                   362.98776816,  375.95161702,  388.91546589,  401.87931475,
                   414.84316361,  427.80701248,  440.77086134,  453.7347102 ,
                   466.69855906,  479.66240793,  492.62625679,  505.59010565,
```

```
        518.55395452,  531.51780338,  544.48165224,  557.4455011 ,
        570.40934997,  583.37319883,  596.33704769,  609.30089656,
        622.26474542,  635.22859428,  648.19244315,  661.15629201,
        674.12014087,  687.08398973,  700.0478386 ,  713.01168746,
        725.97553632,  738.93938519,  751.90323405,  764.86708291,
        777.83093177,  790.79478064,  803.7586295 ,  816.72247836,
        829.68632723,  842.65017609,  855.61402495,  868.57787381,
        881.54172268,  894.50557154,  907.4694204 ,  920.43326927,
        933.39711813,  946.36096699,  959.32481585,  972.28866472,
        985.25251358,  998.21636244, 1011.18021131, 1024.14406017,
       1037.10790903, 1050.0717579 , 1063.03560676, 1075.99945562,
       1088.96330448, 1101.92715335, 1114.89100221, 1127.85485107,
       1140.81869994, 1153.7825488 , 1166.74639766, 1179.71024652,
       1192.67409539, 1205.63794425, 1218.60179311, 1231.56564198,
       1244.52949084, 1257.4933397 , 1270.45718856, 1283.42103743,
       1296.38488629, 1309.34873515, 1322.31258402, 1335.27643288,
       1348.24028174, 1361.2041306 , 1374.16797947, 1387.13182833,
       1400.09567719, 1413.05952606, 1426.02337492, 1438.98722378,
       1451.95107265, 1464.91492151, 1477.87877037, 1490.84261923,
       1503.8064681 , 1516.77031696, 1529.73416582, 1542.69801469,
       1555.66186355]),
 <a list of 120 Patch objects>)
```




```
In [243]: data_sorted.var().describe()

Out[243]: count        123.000
          mean      136410.928
```

```
std      1468432.166
min            0.000
25%            0.000
50%            0.018
75%            0.504
max     16283256.307
dtype: float64
```

```
In [182]: # Applying Low Variance Filter on Attack Data.
          data_sorted_variance = pd.DataFrame(data_sorted.var(), columns=['Variance'])
          high_variance_features = data_sorted_variance[data_sorted_variance['Variance']>=0.018
          feature_reduced_attack_data = data_sorted[high_variance_features]
```

```
In [198]: normal_data_sorted.var().describe()
```

```
Out[198]: count       123.000
          mean      70957.005
          std      754192.653
          min           0.000
          25%           0.000
          50%           0.016
          75%           0.319
          max     8360617.546
          dtype: float64
```

```
In [205]: # Applying Low Variance Filter on Normal Data.
          normal_data_sorted_variance = pd.DataFrame(normal_data_sorted.var(), columns=['Varian
          high_variance_features = normal_data_sorted_variance[normal_data_sorted_variance['Va
          feature_reduced_normal_data = normal_data_sorted[high_variance_features]
```

```
In [351]: feature_reduced_normal_data.shape
```

```
Out[351]: (1048571, 62)
```

1.1.1 One Class SVM on low-variance filtered data

```
In [368]: #Now let's try One-Class SVM on reduced data

          clf = OneClassSVM(gamma='auto').fit(feature_reduced_normal_data.iloc[0:3000,:])
```

```
In [370]: np.count_nonzero(clf.predict(feature_reduced_normal_data.iloc[:1000,:])==1)
```

```
Out[370]: 331
```

```
In [298]: np.count_nonzero(clf.predict(feature_reduced_attack_data[feature_reduced_attack_data
```

```
Out[298]: 172801
```

```
In [294]: feature_reduced_attack_data.columns[:11].drop('1_AIT_002_PV')
```

```
Out[294]: Index(['1_AIT_001_PV', '1_AIT_003_PV', '1_AIT_004_PV', '1_FIT_001_PV',
                 '1_LT_001_PV', '1_MV_001_STATUS', '1_MV_004_STATUS', '1_P_001_STATUS',
                 '1_P_003_STATUS', '1_P_005_STATUS'],
                dtype='object')
```

```
In [295]: feature_reduced_attack_data[feature_reduced_attack_data.columns[:11].drop('1_AIT_002
```

Out[295]:

	1_AIT_001_PV	1_AIT_003_PV	1_AIT_004_PV	1_FIT_001_PV	1_LT_001_PV	\
0	164.210	11.997	482.480	0.001	48.482	
1	164.210	11.997	482.480	0.001	48.482	
2	164.210	11.997	482.480	0.001	48.482	
3	164.210	11.997	482.480	0.001	48.482	
4	164.210	11.997	482.480	0.001	48.482	
5	164.210	11.997	482.480	0.001	48.482	
6	164.212	11.995	482.474	0.001	48.403	
7	164.212	11.995	482.474	0.001	48.403	
8	164.212	11.995	482.474	0.001	48.403	
9	164.212	11.995	482.474	0.001	48.403	
10	164.212	11.995	482.474	0.001	48.403	
11	164.213	11.998	482.480	0.001	48.488	
12	164.213	11.998	482.480	0.001	48.488	
13	164.213	11.998	482.480	0.001	48.488	
14	164.213	11.998	482.480	0.001	48.488	
15	164.213	11.998	482.480	0.001	48.488	
16	164.213	11.998	482.480	0.001	48.488	
17	164.218	11.997	482.469	0.001	48.470	
18	164.218	11.997	482.469	0.001	48.470	
19	164.218	11.997	482.469	0.001	48.470	
20	164.218	11.997	482.469	0.001	48.470	
21	164.218	11.997	482.469	0.001	48.470	
22	164.218	11.997	482.469	0.001	48.470	
23	164.216	11.999	482.452	0.001	48.469	
24	164.216	11.999	482.452	0.001	48.469	
25	164.216	11.999	482.452	0.001	48.469	
26	164.216	11.999	482.452	0.001	48.469	
27	164.216	11.999	482.452	0.001	48.469	
28	164.206	11.998	482.452	0.001	48.367	
29	164.206	11.998	482.452	0.001	48.367	
...	...	...	...	...	...	
172771	172.981	11.921	465.838	0.001	55.571	
172772	172.981	11.921	465.838	0.001	55.571	
172773	172.981	11.921	465.838	0.001	55.571	
172774	172.981	11.921	465.838	0.001	55.571	
172775	172.951	11.918	465.883	0.001	55.611	
172776	172.951	11.918	465.883	0.001	55.611	
172777	172.951	11.918	465.883	0.001	55.611	
172778	172.951	11.918	465.883	0.001	55.611	
172779	172.951	11.918	465.883	0.001	55.611	

172780	172.951	11.918	465.883	0.001	55.611
172781	172.945	11.921	465.933	0.001	55.630
172782	172.945	11.921	465.933	0.001	55.630
172783	172.945	11.921	465.933	0.001	55.630
172784	172.945	11.921	465.933	0.001	55.630
172785	172.945	11.921	465.933	0.001	55.630
172786	172.945	11.921	465.933	0.001	55.630
172787	172.952	11.920	466.001	0.001	55.588
172788	172.952	11.920	466.001	0.001	55.588
172789	172.952	11.920	466.001	0.001	55.588
172790	172.952	11.920	466.001	0.001	55.588
172791	172.952	11.920	466.001	0.001	55.588
172792	172.952	11.920	466.001	0.001	55.588
172793	172.959	11.918	466.034	0.001	55.559
172794	172.959	11.918	466.034	0.001	55.559
172795	172.959	11.918	466.034	0.001	55.559
172796	172.959	11.918	466.034	0.001	55.559
172797	172.959	11.918	466.034	0.001	55.559
172798	172.915	11.921	466.051	0.001	55.726
172799	172.915	11.921	466.051	0.001	55.726
172800	172.915	11.921	466.051	0.001	55.726

	1_MV_001_STATUS	1_MV_004_STATUS	1_P_001_STATUS	1_P_003_STATUS	\
0	1	1	1	1	
1	1	1	1	1	
2	1	1	1	1	
3	1	1	1	1	
4	1	1	1	1	
5	1	1	1	1	
6	1	1	1	1	
7	1	1	1	1	
8	1	1	1	1	
9	1	1	1	1	
10	1	1	1	1	
11	1	1	1	1	
12	1	1	1	1	
13	1	1	1	1	
14	1	1	1	1	
15	1	1	1	1	
16	1	1	1	1	
17	1	1	1	1	
18	1	1	1	1	
19	1	1	1	1	
20	1	1	1	1	
21	1	1	1	1	
22	1	1	1	1	
23	1	1	1	1	
24	1	1	1	1	

25	1	1	1	1
26	1	1	1	1
27	1	1	1	1
28	1	1	1	1
29	1	1	1	1
...	...	...	...	...
172771	1	1	1	1
172772	1	1	1	1
172773	1	1	1	1
172774	1	1	1	1
172775	1	1	1	1
172776	1	1	1	1
172777	1	1	1	1
172778	1	1	1	1
172779	1	1	1	1
172780	1	1	1	1
172781	1	1	1	1
172782	1	1	1	1
172783	1	1	1	1
172784	1	1	1	1
172785	1	1	1	1
172786	1	1	1	1
172787	1	1	1	1
172788	1	1	1	1
172789	1	1	1	1
172790	1	1	1	1
172791	1	1	1	1
172792	1	1	1	1
172793	1	1	1	1
172794	1	1	1	1
172795	1	1	1	1
172796	1	1	1	1
172797	1	1	1	1
172798	1	1	1	1
172799	1	1	1	1
172800	1	1	1	1

	1_P_005_STATUS
0	1
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1

10	1
11	1
12	1
13	1
14	1
15	1
16	1
17	1
18	1
19	1
20	1
21	1
22	1
23	1
24	1
25	1
26	1
27	1
28	1
29	1
...	...
172771	1
172772	1
172773	1
172774	1
172775	1
172776	1
172777	1
172778	1
172779	1
172780	1
172781	1
172782	1
172783	1
172784	1
172785	1
172786	1
172787	1
172788	1
172789	1
172790	1
172791	1
172792	1
172793	1
172794	1
172795	1
172796	1
172797	1

```
172798               1
172799               1
172800               1

[172801 rows x 10 columns]
```

1.1.2 Isolation Forest on low-variance filtered data

```
In [299]: from sklearn.ensemble import IsolationForest

In [458]: clf = IsolationForest(contamination='auto',behaviour="deprecated").fit(feature_reduce

In [459]: np.count_nonzero(clf.predict(feature_reduced_normal_data.iloc[:2000,:10])==1)

Out[459]: 1332

In [460]: clf.predict(feature_reduced_attack_data[feature_reduced_attack_data.columns[:11].dro

Out[460]: array([-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                 -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1])

In [405]: np.count_nonzero(clf.predict(feature_reduced_attack_data[feature_reduced_attack_data

Out[405]: 1000

In [213]: feature_reduced_attack_data.corr()

Out[213]:
```

	1_AIT_001_PV	1_AIT_002_PV	1_AIT_003_PV	\
1_AIT_001_PV	1.000	0.007	0.104	
1_AIT_002_PV	0.007	1.000	0.041	
1_AIT_003_PV	0.104	0.041	1.000	
1_AIT_004_PV	-0.045	0.018	0.229	
1_FIT_001_PV	0.096	0.081	0.094	
1_LT_001_PV	-0.022	-0.108	0.000	
1_MV_001_STATUS	0.097	0.082	0.092	
1_MV_004_STATUS	-0.048	-0.051	0.005	
1_P_001_STATUS	0.095	0.081	0.092	
1_P_003_STATUS	0.095	0.081	0.091	
1_P_005_STATUS	-0.011	0.047	-0.003	
2_DPIT_001_PV	-0.099	0.005	-0.020	
2_FIC_101_CO	-0.042	-0.072	-0.048	
2_FIC_101_PV	0.108	0.066	0.042	
2_FIC_101_SP	0.111	-0.040	0.035	
2_FIC_201_CO	-0.041	0.028	-0.012	
2_FIC_301_CO	0.060	0.010	0.048	

2_FIC_401_CO	-0.051	0.024	-0.045
2_FIC_401_PV	0.109	0.088	0.042
2_FIC_501_CO	-0.009	0.022	-0.007
2_FIC_501_PV	0.113	-0.041	0.008
2_FIC_601_CO	0.041	0.009	0.060
2_FIT_001_PV	-0.009	0.046	0.011
2_FIT_002_PV	-0.038	0.070	0.002
2_FIT_003_PV	0.114	-0.025	0.022
2_FQ_101_PV	0.108	0.066	0.042
2_FQ_401_PV	0.110	0.087	0.042
2_FQ_501_PV	0.113	-0.042	0.008
2_LT_001_PV	0.084	-0.020	0.052
2_LT_002_PV	0.015	-0.001	-0.003
...	...	...	...
2_MCV_101_CO	-0.036	-0.017	-0.034
2_MCV_201_CO	0.024	-0.038	-0.009
2_MCV_301_CO	0.010	0.247	0.037
2_MCV_401_CO	-0.037	-0.017	-0.018
2_MCV_501_CO	0.021	-0.029	0.004
2_MCV_601_CO	0.029	0.097	0.041
2_MV_003_STATUS	-0.012	0.046	0.001
2_MV_006_STATUS	0.104	-0.033	0.022
2_MV_101_STATUS	-0.045	-0.098	-0.055
2_MV_201_STATUS	-0.040	0.033	-0.013
2_MV_301_STATUS	0.058	0.014	0.042
2_MV_401_STATUS	-0.043	-0.089	-0.049
2_MV_501_STATUS	-0.018	0.034	-0.003
2_MV_601_STATUS	0.042	0.013	0.057
2_P_003_SPEED	0.107	-0.032	0.022
2_P_003_STATUS	0.110	-0.033	0.022
2_PIC_003_CO	-0.105	0.040	-0.027
2_PIC_003_PV	0.091	-0.037	0.021
2_PIT_001_PV	-0.153	0.008	-0.025
2_PIT_002_PV	-0.097	0.004	-0.025
2_PIT_003_PV	0.091	-0.037	0.021
2A_AIT_004_PV	-0.033	0.019	-0.043
2B_AIT_002_PV	0.161	-0.051	-0.135
2B_AIT_004_PV	0.005	-0.038	-0.020
3_AIT_003_PV	0.004	0.015	0.012
3_AIT_004_PV	-0.002	-0.014	-0.005
3_FIT_001_PV	0.005	-0.014	0.019
3_LT_001_PV	-0.011	-0.075	-0.036
LEAK_DIFF_PRESSURE	0.057	-0.006	0.022
TOTAL_CONS_REQUIRED_FLOW	0.122	0.039	0.029

	1_AIT_004_PV	1_FIT_001_PV	1_LT_001_PV	\
1_AIT_001_PV	-0.045	0.096	-0.022	
1_AIT_002_PV	0.018	0.081	-0.108	

1_AIT_003_PV	0.229	0.094	0.000
1_AIT_004_PV	1.000	-0.514	-0.354
1_FIT_001_PV	-0.514	1.000	-0.121
1_LT_001_PV	-0.354	-0.121	1.000
1_MV_001_STATUS	-0.519	0.970	-0.120
1_MV_004_STATUS	-0.125	-0.347	0.642
1_P_001_STATUS	-0.505	0.990	-0.115
1_P_003_STATUS	-0.506	0.991	-0.115
1_P_005_STATUS	-0.253	0.169	-0.083
2_DPIT_001_PV	0.250	-0.242	0.114
2_FIC_101_CO	0.196	-0.257	0.057
2_FIC_101_PV	-0.139	0.243	-0.106
2_FIC_101_SP	-0.277	0.088	0.084
2_FIC_201_CO	0.493	-0.334	-0.006
2_FIC_301_CO	0.317	-0.196	0.029
2_FIC_401_CO	0.137	-0.205	0.085
2_FIC_401_PV	-0.028	0.161	-0.117
2_FIC_501_CO	0.264	-0.169	0.018
2_FIC_501_PV	-0.237	0.212	-0.088
2_FIC_601_CO	0.364	-0.223	0.010
2_FIT_001_PV	-0.248	0.167	-0.091
2_FIT_002_PV	-0.109	0.134	-0.068
2_FIT_003_PV	-0.351	0.281	-0.066
2_FQ_101_PV	-0.139	0.243	-0.107
2_FQ_401_PV	-0.029	0.162	-0.118
2_FQ_501_PV	-0.238	0.213	-0.089
2_LT_001_PV	-0.196	0.073	0.019
2_LT_002_PV	0.141	-0.142	0.157
...	...	...	...
2_MCV_101_CO	-0.285	0.131	-0.003
2_MCV_201_CO	-0.235	0.195	-0.089
2_MCV_301_CO	0.037	0.122	-0.151
2_MCV_401_CO	-0.271	0.182	-0.116
2_MCV_501_CO	-0.349	0.212	-0.031
2_MCV_601_CO	-0.063	0.220	-0.158
2_MV_003_STATUS	-0.227	0.149	-0.063
2_MV_006_STATUS	-0.432	0.283	-0.030
2_MV_101_STATUS	0.170	-0.289	0.112
2_MV_201_STATUS	0.510	-0.341	0.009
2_MV_301_STATUS	0.339	-0.214	0.060
2_MV_401_STATUS	0.081	-0.198	0.123
2_MV_501_STATUS	0.313	-0.219	0.048
2_MV_601_STATUS	0.399	-0.229	0.004
2_P_003_SPEED	-0.416	0.281	-0.030
2_P_003_STATUS	-0.449	0.292	-0.023
2_PIC_003_CO	0.461	-0.273	0.010
2_PIC_003_PV	-0.386	0.231	0.003
2_PIT_001_PV	0.233	-0.233	0.157

2_PIT_002_PV	0.261	-0.261	0.116
2_PIT_003_PV	-0.385	0.231	0.003
2A_AIT_004_PV	0.551	-0.259	-0.022
2B_AIT_002_PV	-0.002	-0.054	-0.082
2B_AIT_004_PV	0.262	-0.108	-0.072
3_AIT_003_PV	0.108	-0.067	0.014
3_AIT_004_PV	-0.071	0.040	-0.004
3_FIT_001_PV	0.133	-0.140	0.042
3_LT_001_PV	-0.014	-0.157	0.036
LEAK_DIFF_PRESSURE	-0.238	0.238	-0.080
TOTAL_CONS_REQUIRED_FLOW	-0.555	0.471	-0.160

	1_MV_001_STATUS	1_MV_004_STATUS	1_P_001_STATUS	\
1_AIT_001_PV	0.097	-0.048	0.095	
1_AIT_002_PV	0.082	-0.051	0.081	
1_AIT_003_PV	0.092	0.005	0.092	
1_AIT_004_PV	-0.519	-0.125	-0.505	
1_FIT_001_PV	0.970	-0.347	0.990	
1_LT_001_PV	-0.120	0.642	-0.115	
1_MV_001_STATUS	1.000	-0.338	0.961	
1_MV_004_STATUS	-0.338	1.000	-0.344	
1_P_001_STATUS	0.961	-0.344	1.000	
1_P_003_STATUS	0.962	-0.344	0.999	
1_P_005_STATUS	0.167	-0.081	0.169	
2_DPIT_001_PV	-0.238	0.104	-0.243	
2_FIC_101_CO	-0.253	0.154	-0.248	
2_FIC_101_PV	0.237	-0.133	0.241	
2_FIC_101_SP	0.089	0.009	0.092	
2_FIC_201_CO	-0.333	0.155	-0.327	
2_FIC_301_CO	-0.194	0.112	-0.187	
2_FIC_401_CO	-0.203	0.169	-0.196	
2_FIC_401_PV	0.151	-0.146	0.154	
2_FIC_501_CO	-0.167	0.146	-0.174	
2_FIC_501_PV	0.206	-0.114	0.217	
2_FIC_601_CO	-0.222	0.091	-0.216	
2_FIT_001_PV	0.162	-0.086	0.167	
2_FIT_002_PV	0.137	-0.125	0.142	
2_FIT_003_PV	0.273	-0.108	0.273	
2_FQ_101_PV	0.238	-0.133	0.242	
2_FQ_401_PV	0.151	-0.147	0.154	
2_FQ_501_PV	0.207	-0.114	0.218	
2_LT_001_PV	0.055	0.056	0.066	
2_LT_002_PV	-0.132	0.040	-0.146	
...	...	...	...	
2_MCV_101_CO	0.135	-0.059	0.130	
2_MCV_201_CO	0.197	-0.105	0.200	
2_MCV_301_CO	0.103	-0.139	0.123	
2_MCV_401_CO	0.186	-0.075	0.182	

2_MCV_501_CO	0.215	-0.138	0.213
2_MCV_601_CO	0.212	-0.138	0.222
2_MV_003_STATUS	0.155	-0.071	0.149
2_MV_006_STATUS	0.274	-0.072	0.270
2_MV_101_STATUS	-0.281	0.170	-0.283
2_MV_201_STATUS	-0.341	0.169	-0.337
2_MV_301_STATUS	-0.214	0.109	-0.208
2_MV_401_STATUS	-0.188	0.179	-0.192
2_MV_501_STATUS	-0.218	0.156	-0.226
2_MV_601_STATUS	-0.230	0.096	-0.224
2_P_003_SPEED	0.273	-0.080	0.270
2_P_003_STATUS	0.283	-0.075	0.279
2_PIC_003_CO	-0.266	0.020	-0.259
2_PIC_003_PV	0.224	-0.025	0.221
2_PIT_001_PV	-0.224	0.084	-0.235
2_PIT_002_PV	-0.256	0.107	-0.262
2_PIT_003_PV	0.223	-0.024	0.220
2A_AIT_004_PV	-0.258	0.070	-0.247
2B_AIT_002_PV	-0.051	-0.130	-0.061
2B_AIT_004_PV	-0.107	0.045	-0.098
3_AIT_003_PV	-0.066	0.042	-0.064
3_AIT_004_PV	0.040	-0.020	0.039
3_FIT_001_PV	-0.132	0.083	-0.146
3_LT_001_PV	-0.146	0.047	-0.155
LEAK_DIFF_PRESSURE	0.235	-0.105	0.238
TOTAL_CONS_REQUIRED_FLOW	0.461	-0.244	0.468

	1_P_003_STATUS	...	\
1_AIT_001_PV	0.095	...	
1_AIT_002_PV	0.081	...	
1_AIT_003_PV	0.091	...	
1_AIT_004_PV	-0.506	...	
1_FIT_001_PV	0.991	...	
1_LT_001_PV	-0.115	...	
1_MV_001_STATUS	0.962	...	
1_MV_004_STATUS	-0.344	...	
1_P_001_STATUS	0.999	...	
1_P_003_STATUS	1.000	...	
1_P_005_STATUS	0.170	...	
2_DPIT_001_PV	-0.243	...	
2_FIC_101_CO	-0.248	...	
2_FIC_101_PV	0.240	...	
2_FIC_101_SP	0.092	...	
2_FIC_201_CO	-0.328	...	
2_FIC_301_CO	-0.188	...	
2_FIC_401_CO	-0.197	...	
2_FIC_401_PV	0.153	...	
2_FIC_501_CO	-0.175	...	

2_FIC_501_PV	0.218	...
2_FIC_601_CO	-0.217	...
2_FIT_001_PV	0.167	...
2_FIT_002_PV	0.141	...
2_FIT_003_PV	0.274	...
2_FQ_101_PV	0.241	...
2_FQ_401_PV	0.154	...
2_FQ_501_PV	0.219	...
2_LT_001_PV	0.065	...
2_LT_002_PV	-0.147	...
...	...	...
2_MCV_101_CO	0.131	...
2_MCV_201_CO	0.201	...
2_MCV_301_CO	0.121	...
2_MCV_401_CO	0.183	...
2_MCV_501_CO	0.214	...
2_MCV_601_CO	0.221	...
2_MV_003_STATUS	0.150	...
2_MV_006_STATUS	0.270	...
2_MV_101_STATUS	-0.282	...
2_MV_201_STATUS	-0.337	...
2_MV_301_STATUS	-0.208	...
2_MV_401_STATUS	-0.192	...
2_MV_501_STATUS	-0.227	...
2_MV_601_STATUS	-0.225	...
2_P_003_SPEED	0.270	...
2_P_003_STATUS	0.279	...
2_PIC_003_CO	-0.260	...
2_PIC_003_PV	0.221	...
2_PIT_001_PV	-0.236	...
2_PIT_002_PV	-0.263	...
2_PIT_003_PV	0.220	...
2A_AIT_004_PV	-0.247	...
2B_AIT_002_PV	-0.060	...
2B_AIT_004_PV	-0.098	...
3_AIT_003_PV	-0.065	...
3_AIT_004_PV	0.039	...
3_FIT_001_PV	-0.147	...
3_LT_001_PV	-0.155	...
LEAK_DIFF_PRESSURE	0.239	...
TOTAL_CONS_REQUIRED_FLOW	0.468	...

	2_PIT_003_PV	2A_AIT_004_PV	2B_AIT_002_PV	\
1_AIT_001_PV	0.091	-0.033	0.161	
1_AIT_002_PV	-0.037	0.019	-0.051	
1_AIT_003_PV	0.021	-0.043	-0.135	
1_AIT_004_PV	-0.385	0.551	-0.002	
1_FIT_001_PV	0.231	-0.259	-0.054	

1_LT_001_PV	0.003	-0.022	-0.082
1_MV_001_STATUS	0.223	-0.258	-0.051
1_MV_004_STATUS	-0.024	0.070	-0.130
1_P_001_STATUS	0.220	-0.247	-0.061
1_P_003_STATUS	0.220	-0.247	-0.060
1_P_005_STATUS	0.196	-0.203	-0.022
2_DPIT_001_PV	-0.455	0.235	0.047
2_FIC_101_CO	-0.192	-0.043	0.027
2_FIC_101_PV	0.400	-0.069	-0.041
2_FIC_101_SP	0.347	-0.529	0.078
2_FIC_201_CO	-0.296	0.331	0.038
2_FIC_301_CO	-0.068	0.206	0.153
2_FIC_401_CO	-0.105	-0.120	-0.017
2_FIC_401_PV	0.326	-0.046	-0.024
2_FIC_501_CO	-0.168	0.096	0.110
2_FIC_501_PV	0.405	-0.231	0.076
2_FIC_601_CO	-0.192	0.177	0.053
2_FIT_001_PV	0.191	-0.202	-0.021
2_FIT_002_PV	-0.427	0.046	0.076
2_FIT_003_PV	0.800	-0.371	-0.076
2_FQ_101_PV	0.397	-0.069	-0.041
2_FQ_401_PV	0.323	-0.046	-0.024
2_FQ_501_PV	0.402	-0.232	0.076
2_LT_001_PV	0.242	-0.618	0.114
2_LT_002_PV	-0.162	0.139	0.119
...	...	...	...
2_MCV_101_CO	-0.088	-0.150	0.041
2_MCV_201_CO	-0.129	-0.176	0.101
2_MCV_301_CO	-0.154	0.079	-0.013
2_MCV_401_CO	-0.127	-0.172	-0.089
2_MCV_501_CO	-0.047	-0.219	0.101
2_MCV_601_CO	-0.113	0.102	0.023
2_MV_003_STATUS	0.171	-0.181	-0.021
2_MV_006_STATUS	0.868	-0.471	-0.104
2_MV_101_STATUS	-0.090	-0.083	0.009
2_MV_201_STATUS	-0.204	0.337	-0.013
2_MV_301_STATUS	-0.013	0.221	0.148
2_MV_401_STATUS	-0.002	-0.136	-0.008
2_MV_501_STATUS	-0.065	0.153	0.038
2_MV_601_STATUS	-0.122	0.205	0.017
2_P_003_SPEED	0.937	-0.451	-0.092
2_P_003_STATUS	0.927	-0.491	-0.101
2_PIC_003_CO	-0.906	0.548	0.127
2_PIC_003_PV	0.999	-0.450	-0.104
2_PIT_001_PV	-0.377	0.263	0.112
2_PIT_002_PV	-0.479	0.233	0.085
2_PIT_003_PV	1.000	-0.450	-0.104
2A_AIT_004_PV	-0.450	1.000	0.175

2B_AIT_002_PV	-0.104	0.175	1.000
2B_AIT_004_PV	0.019	0.583	0.022
3_AIT_003_PV	-0.022	0.065	0.002
3_AIT_004_PV	0.021	-0.052	-0.011
3_FIT_001_PV	0.113	-0.073	-0.027
3_LT_001_PV	0.107	-0.328	0.014
LEAK_DIFF_PRESSURE	0.484	-0.187	-0.064
TOTAL_CONS_REQUIRED_FLOW	0.472	-0.467	-0.051

	2B_AIT_004_PV	3_AIT_003_PV	3_AIT_004_PV	\
1_AIT_001_PV	0.005	0.004	-0.002	
1_AIT_002_PV	-0.038	0.015	-0.014	
1_AIT_003_PV	-0.020	0.012	-0.005	
1_AIT_004_PV	0.262	0.108	-0.071	
1_FIT_001_PV	-0.108	-0.067	0.040	
1_LT_001_PV	-0.072	0.014	-0.004	
1_MV_001_STATUS	-0.107	-0.066	0.040	
1_MV_004_STATUS	0.045	0.042	-0.020	
1_P_001_STATUS	-0.098	-0.064	0.039	
1_P_003_STATUS	-0.098	-0.065	0.039	
1_P_005_STATUS	-0.085	-0.059	0.037	
2_DPIT_001_PV	0.080	0.056	-0.035	
2_FIC_101_CO	0.008	0.129	-0.058	
2_FIC_101_PV	-0.005	-0.008	-0.003	
2_FIC_101_SP	-0.177	0.092	-0.039	
2_FIC_201_CO	0.238	0.207	-0.120	
2_FIC_301_CO	0.252	0.190	-0.104	
2_FIC_401_CO	0.049	0.119	-0.051	
2_FIC_401_PV	-0.063	0.010	-0.013	
2_FIC_501_CO	0.102	0.191	-0.112	
2_FIC_501_PV	-0.127	-0.056	0.039	
2_FIC_601_CO	0.115	0.187	-0.102	
2_FIT_001_PV	-0.085	-0.061	0.038	
2_FIT_002_PV	-0.332	-0.129	0.063	
2_FIT_003_PV	0.015	-0.036	0.030	
2_FQ_101_PV	-0.005	-0.009	-0.002	
2_FQ_401_PV	-0.063	0.009	-0.013	
2_FQ_501_PV	-0.128	-0.056	0.039	
2_LT_001_PV	-0.396	0.065	-0.025	
2_LT_002_PV	-0.067	0.044	-0.029	
...	...	...	...	
2_MCV_101_CO	-0.215	-0.127	0.073	
2_MCV_201_CO	-0.290	-0.084	0.053	
2_MCV_301_CO	-0.059	-0.008	-0.009	
2_MCV_401_CO	-0.261	-0.140	0.080	
2_MCV_501_CO	-0.241	-0.132	0.078	
2_MCV_601_CO	-0.034	-0.021	-0.001	
2_MV_003_STATUS	-0.076	-0.052	0.033	

2_MV_006_STATUS	0.044	-0.032	0.030
2_MV_101_STATUS	0.015	0.102	-0.041
2_MV_201_STATUS	0.270	0.206	-0.119
2_MV_301_STATUS	0.282	0.193	-0.107
2_MV_401_STATUS	0.052	0.090	-0.031
2_MV_501_STATUS	0.212	0.185	-0.109
2_MV_601_STATUS	0.184	0.193	-0.108
2_P_003_SPEED	0.021	-0.032	0.028
2_P_003_STATUS	0.036	-0.035	0.032
2_PIC_003_CO	-0.056	0.030	-0.029
2_PIC_003_PV	0.019	-0.022	0.022
2_PIT_001_PV	0.006	0.048	-0.034
2_PIT_002_PV	0.071	0.063	-0.038
2_PIT_003_PV	0.019	-0.022	0.021
2A_AIT_004_PV	0.583	0.065	-0.052
2B_AIT_002_PV	0.022	0.002	-0.011
2B_AIT_004_PV	1.000	0.069	-0.042
3_AIT_003_PV	0.069	1.000	-0.851
3_AIT_004_PV	-0.042	-0.851	1.000
3_FIT_001_PV	0.134	0.171	-0.069
3_LT_001_PV	-0.063	0.060	-0.020
LEAK_DIFF_PRESSURE	-0.092	-0.060	0.032
TOTAL_CONS_REQUIRED_FLOW	-0.185	-0.128	0.080

	3_FIT_001_PV	3_LT_001_PV	LEAK_DIFF_PRESSURE	\
1_AIT_001_PV	0.005	-0.011	0.057	
1_AIT_002_PV	-0.014	-0.075	-0.006	
1_AIT_003_PV	0.019	-0.036	0.022	
1_AIT_004_PV	0.133	-0.014	-0.238	
1_FIT_001_PV	-0.140	-0.157	0.238	
1_LT_001_PV	0.042	0.036	-0.080	
1_MV_001_STATUS	-0.132	-0.146	0.235	
1_MV_004_STATUS	0.083	0.047	-0.105	
1_P_001_STATUS	-0.146	-0.155	0.238	
1_P_003_STATUS	-0.147	-0.155	0.239	
1_P_005_STATUS	-0.138	-0.147	0.181	
2_DPIT_001_PV	0.131	0.157	-0.619	
2_FIC_101_CO	0.613	0.585	-0.291	
2_FIC_101_PV	-0.125	-0.326	0.488	
2_FIC_101_SP	0.381	0.295	0.097	
2_FIC_201_CO	0.493	0.298	-0.338	
2_FIC_301_CO	0.516	0.112	-0.140	
2_FIC_401_CO	0.604	0.702	-0.286	
2_FIC_401_PV	-0.093	-0.383	0.419	
2_FIC_501_CO	0.506	0.336	-0.284	
2_FIC_501_PV	-0.048	-0.135	0.450	
2_FIC_601_CO	0.435	0.040	-0.181	
2_FIT_001_PV	-0.140	-0.148	0.176	

2_FIT_002_PV	-0.495	-0.400	0.030
2_FIT_003_PV	0.039	-0.056	0.682
2_FQ_101_PV	-0.126	-0.327	0.488
2_FQ_401_PV	-0.094	-0.384	0.418
2_FQ_501_PV	-0.048	-0.136	0.451
2_LT_001_PV	0.267	0.300	0.027
2_LT_002_PV	0.120	0.093	-0.169
...	...	...	...
2_MCV_101_CO	-0.316	-0.019	0.089
2_MCV_201_CO	-0.194	-0.149	0.108
2_MCV_301_CO	-0.288	-0.496	0.077
2_MCV_401_CO	-0.425	-0.115	0.116
2_MCV_501_CO	-0.292	-0.257	0.150
2_MCV_601_CO	-0.327	-0.484	0.131
2_MV_003_STATUS	-0.118	-0.131	0.158
2_MV_006_STATUS	0.097	0.060	0.450
2_MV_101_STATUS	0.613	0.653	-0.275
2_MV_201_STATUS	0.496	0.337	-0.317
2_MV_301_STATUS	0.546	0.129	-0.151
2_MV_401_STATUS	0.594	0.736	-0.245
2_MV_501_STATUS	0.487	0.332	-0.268
2_MV_601_STATUS	0.460	0.051	-0.179
2_P_003_SPEED	0.087	0.027	0.551
2_P_003_STATUS	0.108	0.060	0.477
2_PIC_003_CO	-0.128	-0.172	-0.354
2_PIC_003_PV	0.113	0.107	0.483
2_PIT_001_PV	0.100	0.102	-0.421
2_PIT_002_PV	0.148	0.192	-0.631
2_PIT_003_PV	0.113	0.107	0.484
2A_AIT_004_PV	-0.073	-0.328	-0.187
2B_AIT_002_PV	-0.027	0.014	-0.064
2B_AIT_004_PV	0.134	-0.063	-0.092
3_AIT_003_PV	0.171	0.060	-0.060
3_AIT_004_PV	-0.069	-0.020	0.032
3_FIT_001_PV	1.000	0.477	-0.150
3_LT_001_PV	0.477	1.000	-0.205
LEAK_DIFF_PRESSURE	-0.150	-0.205	1.000
TOTAL_CONS_REQUIRED_FLOW	-0.287	-0.342	0.500

	TOTAL_CONS_REQUIRED_FLOW
1_AIT_001_PV	0.122
1_AIT_002_PV	0.039
1_AIT_003_PV	0.029
1_AIT_004_PV	-0.555
1_FIT_001_PV	0.471
1_LT_001_PV	-0.160
1_MV_001_STATUS	0.461
1_MV_004_STATUS	-0.244

1_P_001_STATUS	0.468
1_P_003_STATUS	0.468
1_P_005_STATUS	0.398
2_DPIT_001_PV	-0.540
2_FIC_101_CO	-0.447
2_FIC_101_PV	0.487
2_FIC_101_SP	0.256
2_FIC_201_CO	-0.599
2_FIC_301_CO	-0.258
2_FIC_401_CO	-0.425
2_FIC_401_PV	0.440
2_FIC_501_CO	-0.454
2_FIC_501_PV	0.489
2_FIC_601_CO	-0.311
2_FIT_001_PV	0.399
2_FIT_002_PV	0.256
2_FIT_003_PV	0.591
2_FQ_101_PV	0.488
2_FQ_401_PV	0.442
2_FQ_501_PV	0.491
2_LT_001_PV	0.151
2_LT_002_PV	-0.381
...	...
2_MCV_101_CO	0.341
2_MCV_201_CO	0.384
2_MCV_301_CO	0.363
2_MCV_401_CO	0.333
2_MCV_501_CO	0.521
2_MCV_601_CO	0.423
2_MV_003_STATUS	0.351
2_MV_006_STATUS	0.558
2_MV_101_STATUS	-0.524
2_MV_201_STATUS	-0.665
2_MV_301_STATUS	-0.320
2_MV_401_STATUS	-0.467
2_MV_501_STATUS	-0.529
2_MV_601_STATUS	-0.383
2_P_003_SPEED	0.572
2_P_003_STATUS	0.575
2_PIC_003_CO	-0.517
2_PIC_003_PV	0.472
2_PIT_001_PV	-0.555
2_PIT_002_PV	-0.575
2_PIT_003_PV	0.472
2A_AIT_004_PV	-0.467
2B_AIT_002_PV	-0.051
2B_AIT_004_PV	-0.185
3_AIT_003_PV	-0.128

```
3_AIT_004_PV                          0.080
3_FIT_001_PV                         -0.287
3_LT_001_PV                          -0.342
LEAK_DIFF_PRESSURE                    0.500
TOTAL_CONS_REQUIRED_FLOW              1.000

[61 rows x 61 columns]
```

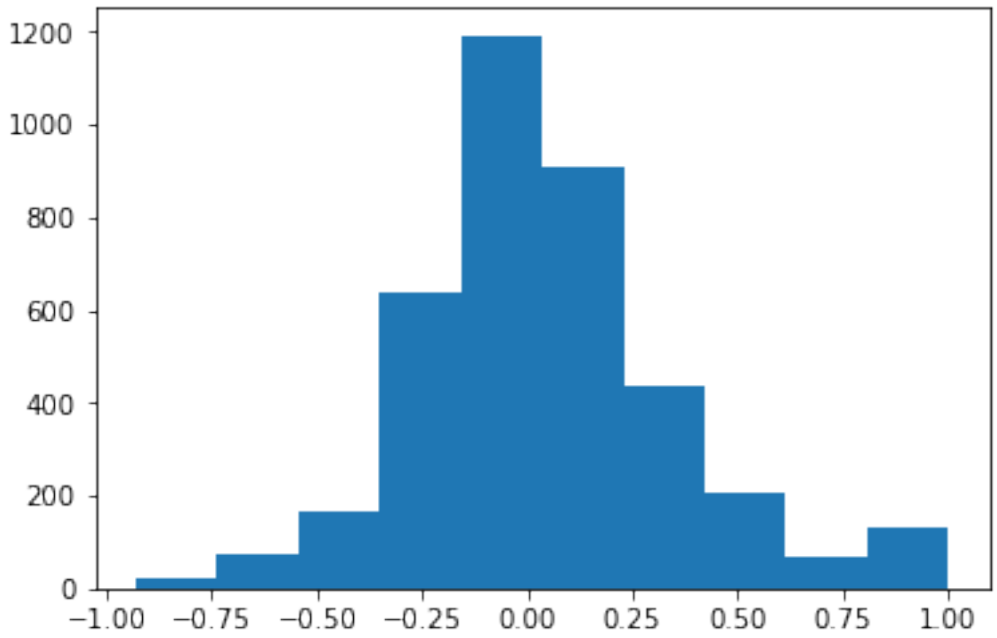
```
In [217]: corr_matrix_normal_data = feature_reduced_normal_data.corr()
```

```
In [230]: correlation_list =[]
          correlation_list_revised =[]
          for index, rows in corr_matrix_normal_data.iterrows():
              correlation_list.append(rows)
          for i in range(len(correlation_list)):
              for j in range(i+1,len(correlation_list[i])):
                  correlation_list_revised.append(correlation_list[i][j])
          len(correlation_list_revised)
```

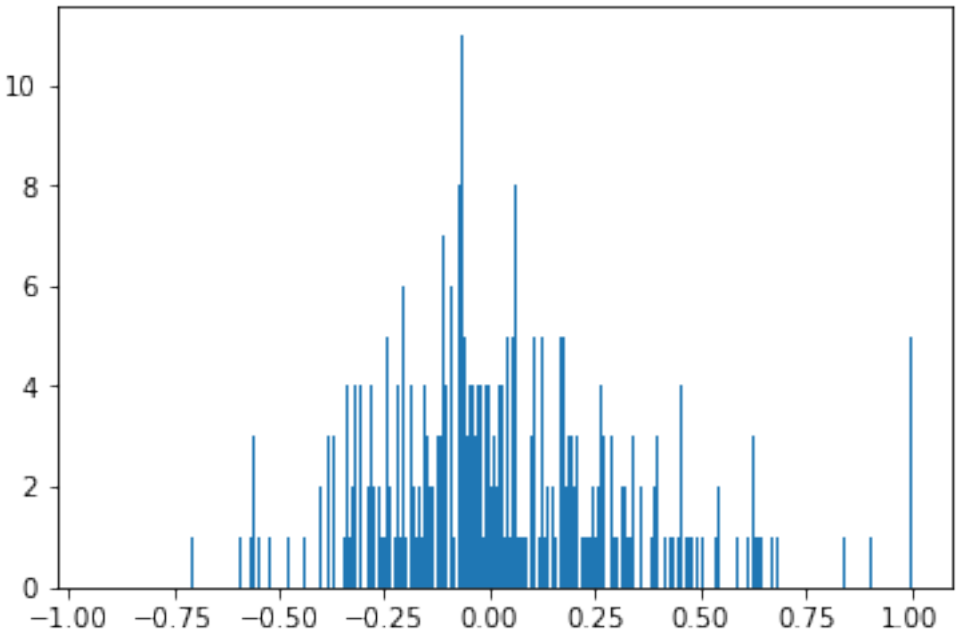
```
Out[230]: 1891
```

```
In [227]: plt.hist(correlation_list_revised)
```

```
Out[227]: (array([  20.,   72.,  168.,  640., 1192.,  910.,  436.,  206.,   66.,
                   134.]),
           array([-0.93051612, -0.73746451, -0.5444129 , -0.35136128, -0.15830967,
                   0.03474194,  0.22779355,  0.42084516,  0.61389678,  0.80694839,
                   1.        ]),
           <a list of 10 Patch objects>)
```



```
In [240]: plt.hist(correlation_list_revised,bins=1800)

Out[240]: (array([2., 0., 0., ..., 0., 2., 5.]),
           array([-0.93051612, -0.92944361, -0.9283711 , ...,  0.99785498,
                   0.99892749,  1.        ]),
           <a list of 1800 Patch objects>)
```



So by now we had seen that first we applied models directly to complete dataset which doesn't provided appropriate results. Then, we reduced the dimensions of dataset and again modelled it which resulted in improvement of the model's performance. But, still it is not considerable.

1.1.3 Now, the next strategy is to separate the data in 3 subsequent processes (P1, P2 and P3) and apply PCA to confine a large part of the variance within a few principal components (features).

2 P1 process

Here we'll apply PCA in two ways just to see how much difference it makes on result.

- First plot: Normal data will be fitted on PCA function and we'll only transform the attack data using fitted PCA.
- Second plot: Here, attack data itself, will be fit and transformed using PCA.

```
In [5]: from sklearn.decomposition import PCA

In [7]: pca = PCA(0.95)

In [8]: # selecting P1 variables with 1_LT_001_PV in range (40,70)
        P1_normal = normal_data_sorted[(normal_data_sorted['\\\\WIN-25J4RO10SBF\\LOG_DATA\\SUTI
        P1_normal = P1_normal[P1_normal.columns[:19]]

In [9]: P1_normal.describe()

Out[9]:        \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_AIT_001_PV  \
        count                                          961726.000
        mean                                              167.568
        std                                                14.092
        min                                                 0.000
        25%                                               155.891
        50%                                               163.121
        75%                                               175.035
        max                                               214.311

               \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_AIT_002_PV  \
        count                                          961714.000
        mean                                                0.622
        std                                                 0.060
        min                                                 0.000
        25%                                                 0.583
        50%                                                 0.625
        75%                                                 0.655
        max                                                 2.059

               \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_AIT_003_PV  \
        count                                          961726.000
        mean                                               11.713
        std                                                 0.173
        min                                                 0.000
        25%                                                11.588
        50%                                                11.778
        75%                                                11.818
        max                                                12.013

               \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_AIT_004_PV  \
        count                                          961720.000
        mean                                              494.111
        std                                                17.915
        min                                                 0.000
        25%                                               484.338
        50%                                               496.938
        75%                                               505.997
        max                                               526.529
```

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_AIT_005_PV \
count	961726.000
mean	0.308
std	0.048
min	0.208
25%	0.267
50%	0.312
75%	0.345
max	0.422

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_FIT_001_PV \
count	961726.000
mean	0.538
std	0.859
min	0.001
25%	0.001
50%	0.001
75%	1.858
max	2.066

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_LS_001_AL \
count	961726.000
mean	0.000
std	0.000
min	0.000
25%	0.000
50%	0.000
75%	0.000
max	0.000

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_LS_002_AL \
count	961726.000
mean	0.000
std	0.000
min	0.000
25%	0.000
50%	0.000
75%	0.000
max	0.000

	\\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_LT_001_PV \
count	961726.000
mean	55.844
std	8.489
min	40.001
25%	48.611
50%	56.628

```
75%                                                  62.149
max                                                  70.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_MV_001_STATUS  \
count                                           961726.000
mean                                                 1.281
std                                                  0.450
min                                                  0.000
25%                                                  1.000
50%                                                  1.000
75%                                                  2.000
max                                                  2.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_MV_002_STATUS  \
count                                           961726.000
mean                                                 1.000
std                                                  0.000
min                                                  1.000
25%                                                  1.000
50%                                                  1.000
75%                                                  1.000
max                                                  1.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_MV_003_STATUS  \
count                                           961726.000
mean                                                 1.000
std                                                  0.000
min                                                  1.000
25%                                                  1.000
50%                                                  1.000
75%                                                  1.000
max                                                  1.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_MV_004_STATUS  \
count                                           961726.000
mean                                                 1.237
std                                                  0.428
min                                                  0.000
25%                                                  1.000
50%                                                  1.000
75%                                                  1.000
max                                                  2.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_P_001_STATUS  \
count                                           961726.000
mean                                                 1.281
std                                                  0.450
min                                                  1.000
```

```
25%                                                 1.000
50%                                                 1.000
75%                                                 2.000
max                                                 2.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_P_002_STATUS  \
count                                          961726.000
mean                                                1.000
std                                                 0.000
min                                                 1.000
25%                                                 1.000
50%                                                 1.000
75%                                                 1.000
max                                                 1.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_P_003_STATUS  \
count                                          961726.000
mean                                                1.281
std                                                 0.450
min                                                 1.000
25%                                                 1.000
50%                                                 1.000
75%                                                 2.000
max                                                 2.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_P_004_STATUS  \
count                                          961726.000
mean                                                1.000
std                                                 0.000
min                                                 1.000
25%                                                 1.000
50%                                                 1.000
75%                                                 1.000
max                                                 1.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_P_005_STATUS  \
count                                          961726.000
mean                                                1.234
std                                                 0.424
min                                                 1.000
25%                                                 1.000
50%                                                 1.000
75%                                                 1.000
max                                                 2.000

       \\WIN-25J4RO10SBF\LOG_DATA\SUTD_WADI\LOG_DATA\1_P_006_STATUS
count                                          961726.000
mean                                                1.000
```

```
std                                                    0.000
min                                                    1.000
25%                                                    1.000
50%                                                    1.000
75%                                                    1.000
max                                                    1.000
```

```
In [10]: pca.fit(P1_normal.dropna())
```

```
Out[10]: PCA(copy=True, iterated_power='auto', n_components=0.95, random_state=None,
           svd_solver='auto', tol=0.0, whiten=False)
```

```
In [11]: normal_PC = pca.transform(P1_normal.dropna())
```

```
In [12]: pca.explained_variance_ratio_
```

```
Out[12]: array([0.60433615, 0.27824022, 0.11554166])
```

It's awesome to see that these three components contain almost 99.8 % variance of complete data.

```
In [13]: normal_PC = pd.DataFrame(data = normal_PC
                        , columns = ['PC1', 'PC2', 'PC3'])
```

```
In [14]: normal_PC.describe()
```

```
Out[14]:              PC1        PC2        PC3
         count 961708.000 961708.000 961708.000
         mean       0.000     -0.000      0.000
         std       18.936     12.848      8.280
         min      -48.130    -21.643    -16.720
         25%      -11.616    -10.058     -6.409
         50%       -0.356     -2.684     -0.460
         75%       11.353      6.906      6.917
         max      517.782     52.103     61.828
```

```
In [22]: # before proceeding further let's first remove some outliers from the normal data so
         # the data become consistent and appropriate for analysis.
         normal_PC = normal_PC[(normal_PC['PC1']<100)]
```

```
In [27]: # These are the new statistics of normal data
         normal_PC.describe()
```

```
Out[27]:              PC1        PC2        PC3
         count 961623.000 961623.000 961623.000
         mean      -0.046     -0.004     -0.004
         std       18.303     12.840      8.267
         min      -48.130    -21.643    -16.720
         25%      -11.617    -10.058     -6.410
         50%       -0.358     -2.685     -0.462
         75%       11.350      6.902      6.914
         max       59.600     44.231     21.240
```

```
In [28]: normal_PC.head()
```

```
Out[28]:       PC1    PC2   PC3
         0 -11.694 -2.123 6.789
         1 -11.694 -2.123 6.789
         2 -11.694 -2.123 6.789
         3 -11.731 -2.153 6.924
         4 -11.731 -2.153 6.924
```

Now let's try to apply the same for the attack data

```
In [104]: attack_PC = pca.transform(attack_data[attack_data.columns[3:22]].dropna())
```

```
In [ ]: pca.fit_transform
```

Here again we obtained around 99.83 % variance of complete data.

```
In [105]: attack_PC = pd.DataFrame(data = attack_PC
                       , columns = ['PC1', 'PC2', 'PC3'])
```

```
In [106]: attack_PC.describe()
```

```
Out[106]:              PC1        PC2        PC3
          count 172795.000 172795.000 172795.000
          mean      32.752     24.209      6.201
          std       18.700     19.030      8.293
          min     -154.241     -1.151    -11.174
          25%       19.668     16.157     -0.012
          50%       31.937     23.026      5.025
          75%       45.057     30.612     12.568
          max      518.222    441.598     64.498
```

```
In [20]: # Although removing outliers in attack data is not meaningful but for the visualizati
         # that we are going to plot it is better to clean the extreme outliers out of it.
         attack_PC = attack_PC[(attack_PC['PC1']<=100) & (attack_PC['PC1']>=-100)]
```

```
In [22]: # The new statistics of PCA reduced attack data is
         attack_PC.describe()
```

```
Out[22]:              PC1        PC2        PC3
         count 172503.000 172503.000 172503.000
         mean      32.974     23.556      6.134
         std       16.399      9.597      8.136
         min        2.103     -1.151    -11.174
         25%       19.737     16.143     -0.016
         50%       31.967     22.999      5.007
         75%       45.078     30.568     12.524
         max       71.790     46.524     27.345
```

```
In [23]: attack_PC.head()
```

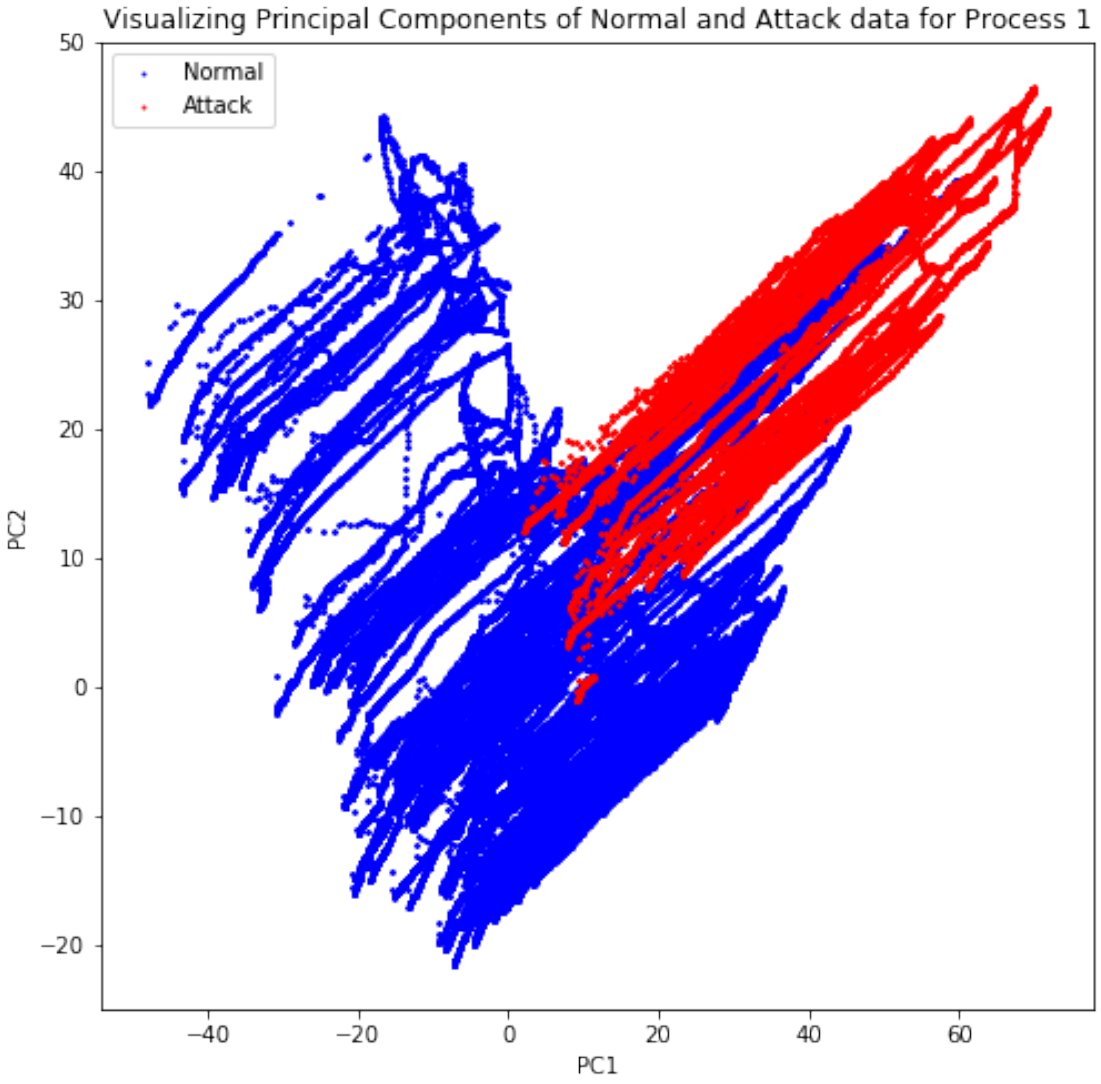
Out[23]:

	PC1	PC2	PC3
0	11.289	0.717	8.455
1	11.289	0.717	8.455
2	11.289	0.717	8.455
3	11.289	0.717	8.455
4	11.289	0.717	8.455

2.0.1 So, by now we had successfully reduced normal and attack data into 3 principal components each. Let's try to visualize them now.

In [22]:
```
from mpl_toolkits import mplot3d
```

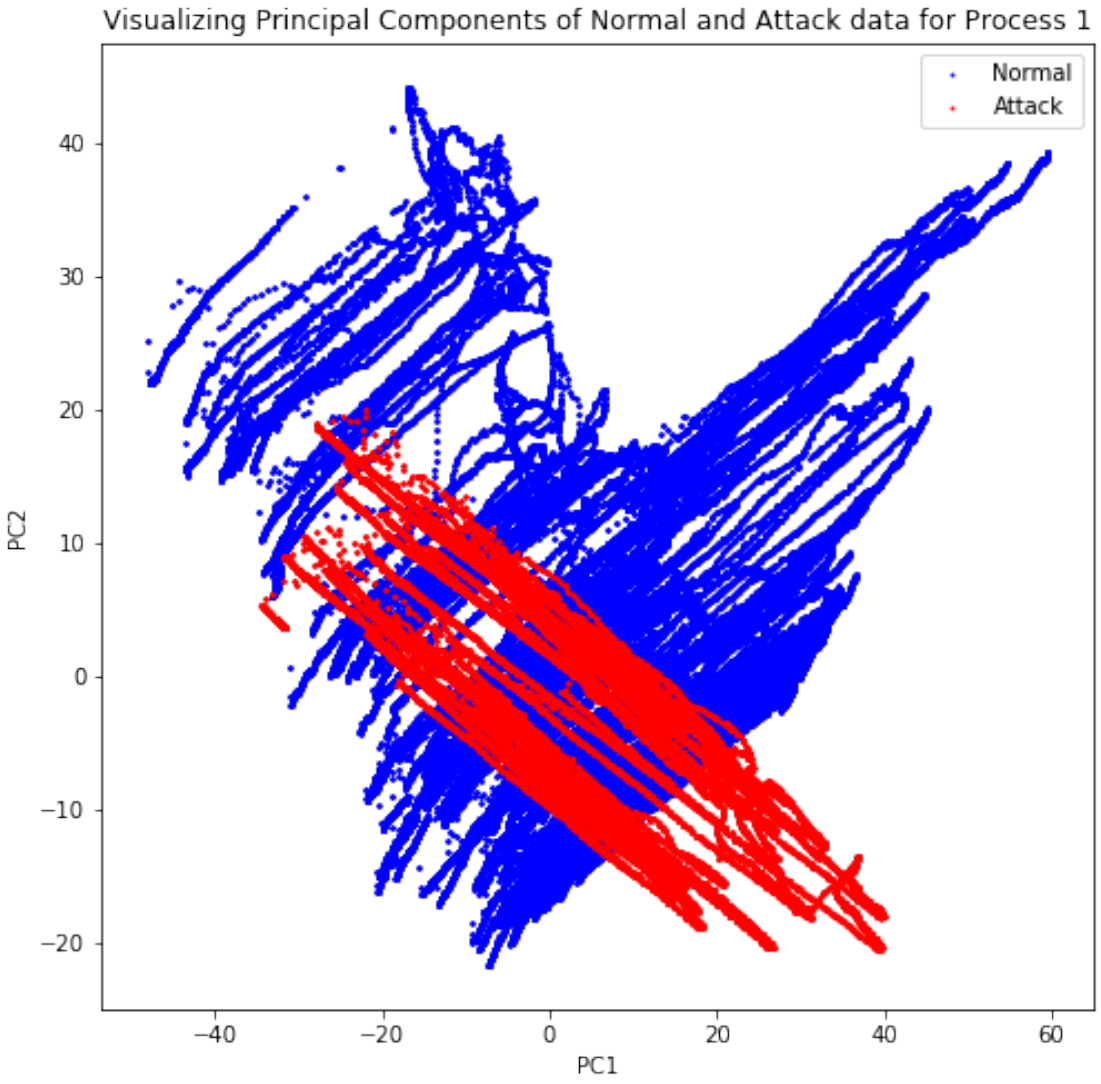
In [45]:
```
# First, let's try to see whether by plotting only first two PC, can we differentiate
# between the data points.
%matplotlib inline
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,8))
ax = plt.axes()
#sample_normal = normal_PC.sample(n=100000)
PC1_normal = normal_PC['PC1']
PC2_normal = normal_PC['PC2']
ax.scatter(PC1_normal, PC2_normal, s=1,c ='b');
PC1_attack = attack_PC['PC1']
PC2_attack = attack_PC['PC2']
ax.scatter(PC1_attack, PC2_attack, s=1,c ='r');
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.legend(['Normal','Attack'])
ax.set_title('Visualizing Principal Components of Normal and Attack data for Process
plt.show()
```



Note that the above graph is plotted by removing some extreme outliers from the transformed PC attack data.

```
In [31]: # First, let's try to see whether by plotting only first two PC, can we differentiate
         # between the data points.
         %matplotlib inline
         import matplotlib.pyplot as plt
         fig = plt.figure(figsize=(8,8))
         ax = plt.axes()
         #sample_normal = normal_PC.sample(n=100000)
         PC1_normal = normal_PC['PC1']
         PC2_normal = normal_PC['PC2']
         ax.scatter(PC1_normal, PC2_normal, s=1,c ='b');
         PC1_attack = attack_PC['PC1']
```

```
PC2_attack = attack_PC['PC2']
ax.scatter(PC1_attack, PC2_attack, s=1,c ='r');
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.legend(['Normal','Attack'])
ax.set_title('Visualizing Principal Components of Normal and Attack data for Process 1
plt.show()
```



The above graph is plotted by removing some extreme outliers from the transformed and fit-transformed PC attack data respectively. The difference in these two plots is pretty clear.

Above charts represents that not all the data points in attack log are actually anomalies. Infact, it seems based on the two plotted PCs that most entries of attack data in Process 1 are under

the standard operation as observed in normal data. Next, we need to detect the attacks that has been conducted on process 1 by modelling the principal component reduced normal data.

2.1 Before training the model, let's first arrange all the actual attacks in a dataframe.

```
In [107]: actual_attacks = attack_data.iloc[attack_PC.index][['Date','Time']]
          actual_attacks['DateTime'] = actual_attacks['Date']+' '+actual_attacks['Time']

In [108]: actual_attacks['datetime'] = list(pd.to_datetime(actual_attacks['DateTime']))

In [109]: actual_attacks['state'] = 1

In [110]: # Now we'll set the datetime column as index for better accessibility of the datafra
          actual_attacks = actual_attacks.set_index('datetime')

In [121]: # Let's take out the datetime index of actual attacks by slicing the data based on a
          # in the attack details file for all the 15 attacks.
          timestamps_attack = pd.concat([actual_attacks.loc['2017-10-09 19:25:00':'2017-10-09
                                         actual_attacks.loc['2017-10-10 10:24:10':'2017-10-10 10:3
                                         actual_attacks.loc['2017-10-10 10:55:00':'2017-10-10 11:2
                                         actual_attacks.loc['2017-10-10 11:30:40':'2017-10-10 11:4
                                         actual_attacks.loc['2017-10-10 13:39:30':'2017-10-10 13:5
                                         actual_attacks.loc['2017-10-10 14:48:17':'2017-10-10 14:5
                                         actual_attacks.loc['2017-10-10 17:40:00':'2017-10-10 17:4
                                         actual_attacks.loc['2017-10-11 10:55:00':'2017-10-11 10:5
                                         actual_attacks.loc['2017-10-11 11:17:54':'2017-10-11 11:3
                                         actual_attacks.loc['2017-10-11 11:36:31':'2017-10-11 11:4
                                         actual_attacks.loc['2017-10-11 11:59:00':'2017-10-11 12:0
                                         actual_attacks.loc['2017-10-11 12:07:30':'2017-10-11 12:1
                                         actual_attacks.loc['2017-10-11 12:16:00':'2017-10-11 12:2
                                         actual_attacks.loc['2017-10-11 15:26:30':'2017-10-11 15:3

In [123]: # Let's change the state of above timestamps to -1 (which mean attack) in the actual_
          actual_attacks.loc[timestamps_attack,'state'] = -1

In [127]: actual_attacks['state'].value_counts()

Out[127]:  1    162847
          -1      9948
          Name: state, dtype: int64
```

Above, we have successfully arranged actual classified attacks data where 1 means normal and -1 are the points when we attacked the system. Hereafter, we'll train different models and then compare the model predictions of attacks with above actual attack data in order to compute confusion matrix.

2.1.1 Now, it's time to model the PC reduced normal data using different algorithms.

2.2 ---> One Class SVM on Principal Component reconstructed Process 1 normal data

In [128]:
```
from sklearn.svm import OneClassSVM
```

In [193]:
```
# defining the model with its parameters. One thing to note here is varying the para
# impact on the outcome of model predictions on attack data. Thus it needs to be adj
OC_SVM = OneClassSVM(gamma='scale',nu=0.0001)
```

In [194]:
```
OC_SVM.fit(normal_PC.sample(500000))
```

Out[194]:
```
OneClassSVM(cache_size=200, coef0=0.0, degree=3, gamma='scale', kernel='rbf',
      max_iter=-1, nu=0.0001, random_state=None, shrinking=True, tol=0.001,
      verbose=False)
```

In [195]:
```
# Let's predict how many data points has been identified as normal in attack data.
print(np.count_nonzero(OC_SVM.predict(attack_PC)==1),'data points has been identifie
```

```
153976 data points has been identified as normal
```

In [144]:
```
# Now before predicting attack data let's first make a space to store the prediction
# corresponding timeframe by defining a variable
P1_attack_predicted = attack_data.loc[attack_PC.index][['Date','Time']]
P1_attack_predicted.head()
```

Out[144]:

	Date	Time
0	10/9/2017	6:00:00.000 PM
1	10/9/2017	6:00:01.000 PM
2	10/9/2017	6:00:02.000 PM
3	10/9/2017	6:00:03.000 PM
4	10/9/2017	6:00:04.000 PM

In [145]:
```
# Now let's add a column of Date + Time which will be helpful for us in visualizatio
P1_attack_predicted['DateTime'] = P1_attack_predicted['Date']+' '+P1_attack_predicte
P1_attack_predicted.head()
```

Out[145]:

	Date	Time	DateTime
0	10/9/2017	6:00:00.000 PM	10/9/2017 6:00:00.000 PM
1	10/9/2017	6:00:01.000 PM	10/9/2017 6:00:01.000 PM
2	10/9/2017	6:00:02.000 PM	10/9/2017 6:00:02.000 PM
3	10/9/2017	6:00:03.000 PM	10/9/2017 6:00:03.000 PM
4	10/9/2017	6:00:04.000 PM	10/9/2017 6:00:04.000 PM

In [146]:
```
P1_attack_predicted['datetime'] = list(pd.to_datetime(P1_attack_predicted['DateTime'
```

In [196]:
```
# Let's store these predictions in a separate column in P1_attack_predicted for bett
P1_attack_predicted['predictions'] = OC_SVM.predict(attack_PC)
P1_attack_predicted.head()
```

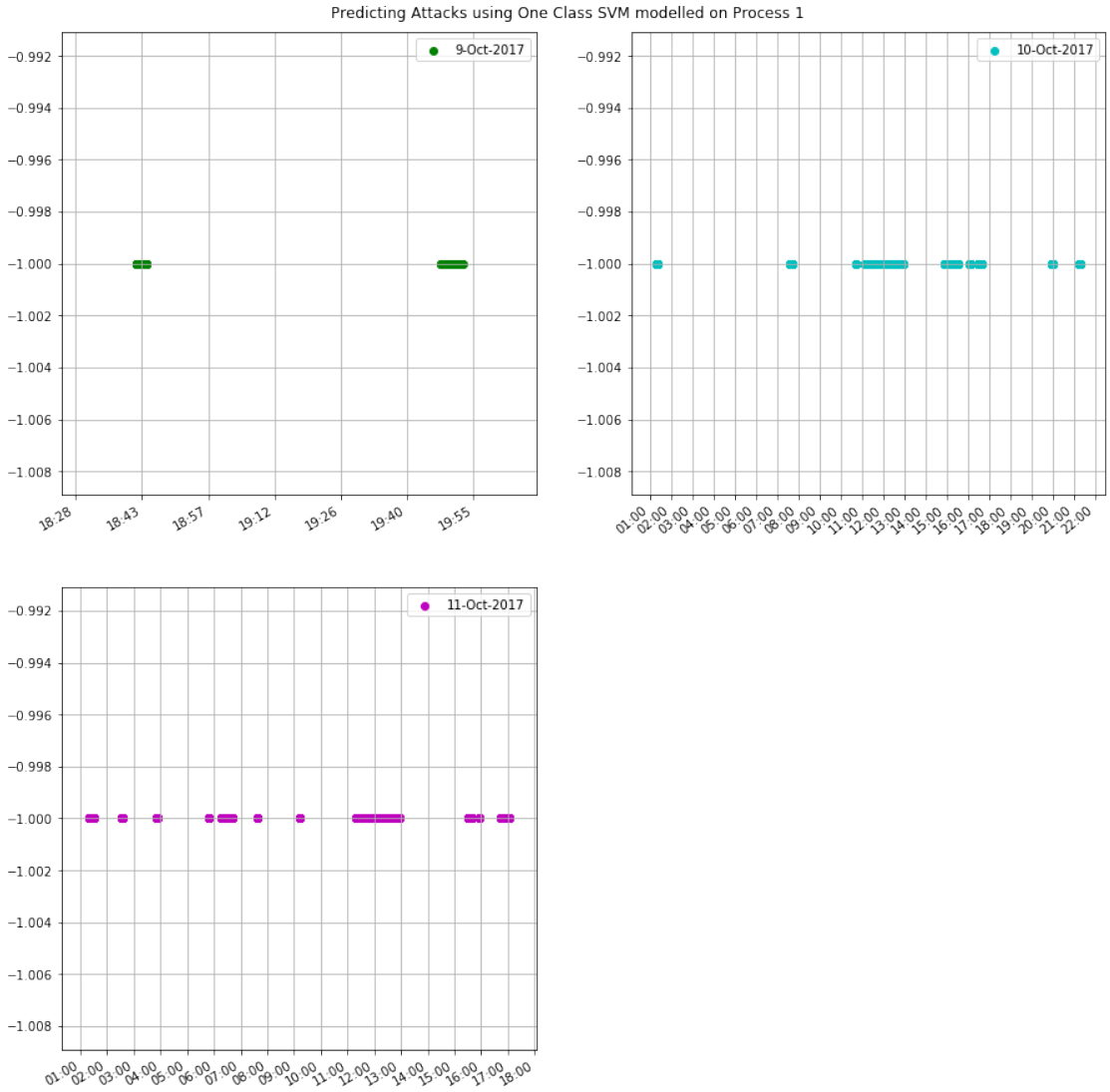
Out[196]:

	Date	Time	DateTime	datetime	predictions
0	10/9/2017	6:00:00.000 PM	10/9/2017 6:00:00.000 PM	2017-10-09 18:00:00	1
1	10/9/2017	6:00:01.000 PM	10/9/2017 6:00:01.000 PM	2017-10-09 18:00:01	1
2	10/9/2017	6:00:02.000 PM	10/9/2017 6:00:02.000 PM	2017-10-09 18:00:02	1
3	10/9/2017	6:00:03.000 PM	10/9/2017 6:00:03.000 PM	2017-10-09 18:00:03	1
4	10/9/2017	6:00:04.000 PM	10/9/2017 6:00:04.000 PM	2017-10-09 18:00:04	1

In [151]:
```
# Let's slice the entries which are predicted as outlier (attacked). We will use the
# instances predicted as attacks with the time.
# Here, -1 is the notation for attack predictions
predicted_attacks = P1_attack_predicted[P1_attack_predicted['predictions']==-1]
```

In [322]:
```
# Above plot can be simplified for better understanding

# Let's first specify some customized formatting for the plot
#fig, axs = plt.subplots(3,figsize=(10,10))
figure = plt.figure(figsize=(15,15))
formatter = mpldates.DateFormatter('%H:%M')
plt.suptitle('Predicting Attacks using One Class SVM modelled on Process 1',y=0.9)


#Plot for 9th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predi
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,1)
axes.xaxis.set_major_formatter(formatter)
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/9/2017']['predictions
axes.legend(['9-Oct-2017'])
#plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=30, ha='right')
plt.grid()

#Plot for 10th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predi
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,2)
axes.xaxis.set_major_formatter(formatter)
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/10/2017']['predictio
axes.legend(['10-Oct-2017'])
plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=40, ha='right')
```

```
plt.grid()

#Plot for 11th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predi
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,3)
axes.xaxis.set_major_formatter(formatter)
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/11/2017']['predictio
axes.legend(['11-Oct-2017'])
plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=30, ha='right')
plt.grid()

plt.show()
```



```
In [597]: # By running the code below we can see that the level indicator 1_LT_001_PV value cr
          # the attack started
          attack_data.iloc[6440:6450,[1,2,11]]
```

Out[597]:

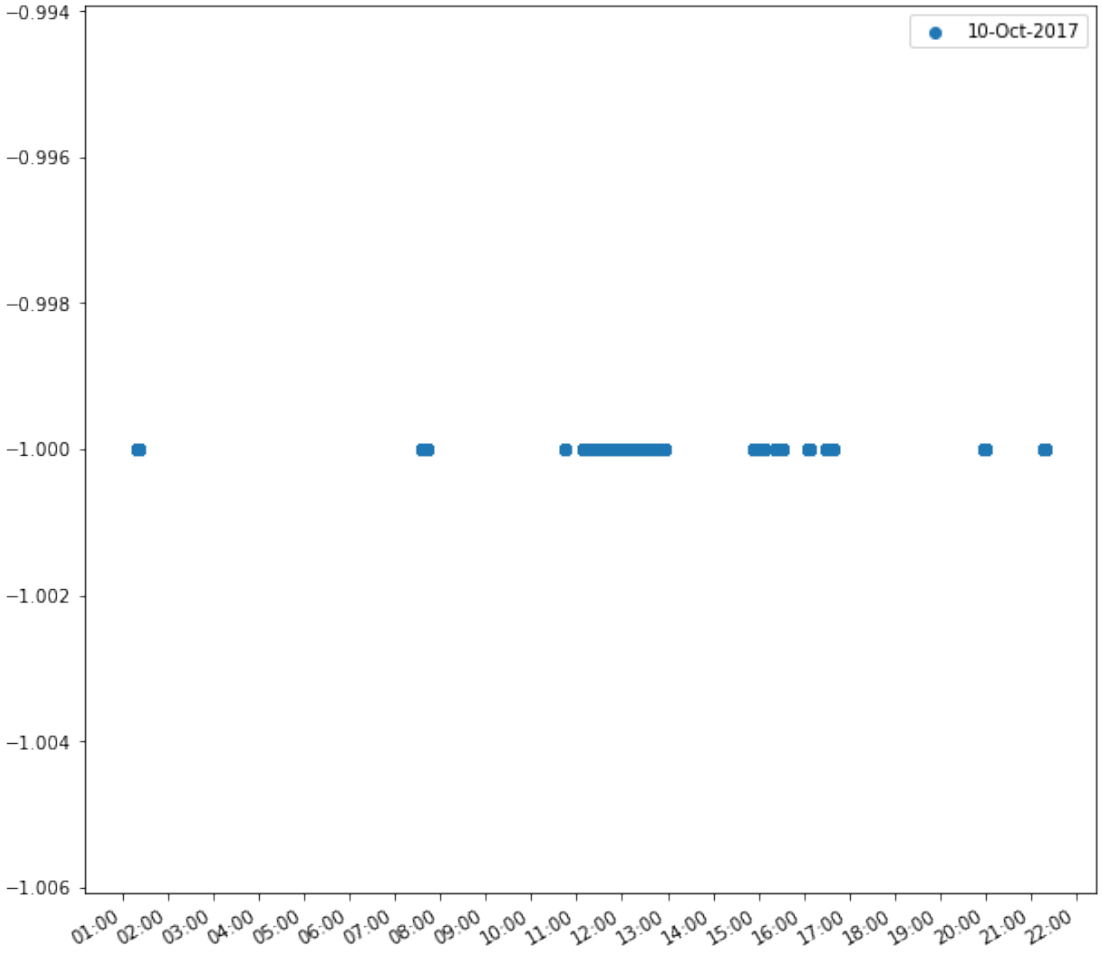
	Date	Time	1_LT_001_PV
6440	10/9/2017	7:47:20.000 PM	70.000
6441	10/9/2017	7:47:21.000 PM	70.000
6442	10/9/2017	7:47:22.000 PM	70.000
6443	10/9/2017	7:47:23.000 PM	70.000
6444	10/9/2017	7:47:24.000 PM	70.000
6445	10/9/2017	7:47:25.000 PM	70.000
6446	10/9/2017	7:47:26.000 PM	70.238
6447	10/9/2017	7:47:27.000 PM	70.238
6448	10/9/2017	7:47:28.000 PM	70.238
6449	10/9/2017	7:47:29.000 PM	70.238

2.2.1 Note: There is a ambiguity in the prediction of attack imposed on 9th Oct on Process 1. According to the attack details the attack started at 19:25:00 which should result in overflow of primary tank. But the level indicator 1_LT_001_PV went above 70 (high setpoint) at 19:47:26. That's why the model started indicating anomaly at that time.

Time to compute the Confusion Matrix to check the performance of our model.

```
In [158]: from sklearn.metrics import confusion_matrix
          from sklearn.metrics import accuracy_score
          from sklearn.metrics import classification_report

In [200]: print(confusion_matrix(actual_attacks['state'],P1_attack_predicted['predictions'],lab

[[148772  14075]
 [  5204   4744]]


In [198]: print('Accuracy Score:', accuracy_score(actual_attacks['state'],P1_attack_predicted[

Accuracy Score: 0.888428484620504


In [199]: print(classification_report(actual_attacks['state'],P1_attack_predicted['predictions

              precision    recall  f1-score   support

          -1       0.25      0.48      0.33      9948
           1       0.97      0.91      0.94    162847

   micro avg       0.89      0.89      0.89    172795
   macro avg       0.61      0.70      0.63    172795
weighted avg       0.93      0.89      0.90    172795
```

```
In [227]: a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predi
          x = mpldates.date2num(a)
          formatter = matplotlib.dates.DateFormatter('%H:%M')

          figure = plt.figure(figsize=(10,10))
          axes = figure.add_subplot(1, 1, 1)

          axes.xaxis.set_major_formatter(formatter)

          axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/10/2017']['predictio
          axes.legend(['10-Oct-2017'])
          plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
          plt.gcf().autofmt_xdate()

          plt.show() #plt.plot(x,predicted_attacks[predicted_attacks['Date']=='10/10/2017']['p
```



2.3 ---> Isolation Forest on Principal Component reconstructed Process 1 normal data

```
In [323]: from sklearn.ensemble import IsolationForest

In [243]: # Here again, we'll tune the parameters in accordance to provide the best outputs.
          Isolation_Forest = IsolationForest(contamination=0.0001, n_estimators=300, behaviour=

In [246]: Isolation_Forest.fit(normal_PC.sample(200000))

Out[246]: IsolationForest(behaviour='new', bootstrap=False, contamination=0.0001,
                  max_features=1.0, max_samples='auto', n_estimators=300,
                  n_jobs=None, random_state=None, verbose=0)

In [247]: # Let's predict how many data points has been identified as normal in attack data.
          print(np.count_nonzero(Isolation_Forest.predict(attack_PC)==1),'data points has been

162579 data points has been identified as normal


In [615]: # Here again, before predicting attack data let's first make a space to store the pr
          # corresponding timeframe by defining a variable
          P1_attack_predicted = attack_data.loc[attack_PC.index][['Date','Time']]
          P1_attack_predicted.head()

Out[615]:         Date            Time
          0  10/9/2017  6:00:00.000 PM
          1  10/9/2017  6:00:01.000 PM
          2  10/9/2017  6:00:02.000 PM
          3  10/9/2017  6:00:03.000 PM
          4  10/9/2017  6:00:04.000 PM

In [616]: # Let's now join the date and time column in order to perform 'pd.to_datetime' opera
          P1_attack_predicted['DateTime'] = P1_attack_predicted['Date']+' '+P1_attack_predicte
          P1_attack_predicted

Out[616]:              Date            Time                   DateTime
          0        10/9/2017  6:00:00.000 PM   10/9/2017 6:00:00.000 PM
          1        10/9/2017  6:00:01.000 PM   10/9/2017 6:00:01.000 PM
          2        10/9/2017  6:00:02.000 PM   10/9/2017 6:00:02.000 PM
          3        10/9/2017  6:00:03.000 PM   10/9/2017 6:00:03.000 PM
          4        10/9/2017  6:00:04.000 PM   10/9/2017 6:00:04.000 PM
          5        10/9/2017  6:00:05.000 PM   10/9/2017 6:00:05.000 PM
          6        10/9/2017  6:00:06.000 PM   10/9/2017 6:00:06.000 PM
          7        10/9/2017  6:00:07.000 PM   10/9/2017 6:00:07.000 PM
          8        10/9/2017  6:00:08.000 PM   10/9/2017 6:00:08.000 PM
          9        10/9/2017  6:00:09.000 PM   10/9/2017 6:00:09.000 PM
          10       10/9/2017  6:00:10.000 PM   10/9/2017 6:00:10.000 PM
          11       10/9/2017  6:00:11.000 PM   10/9/2017 6:00:11.000 PM
          12       10/9/2017  6:00:12.000 PM   10/9/2017 6:00:12.000 PM
          13       10/9/2017  6:00:13.000 PM   10/9/2017 6:00:13.000 PM
```

14	10/9/2017	6:00:14.000 PM	10/9/2017	6:00:14.000 PM
15	10/9/2017	6:00:15.000 PM	10/9/2017	6:00:15.000 PM
16	10/9/2017	6:00:16.000 PM	10/9/2017	6:00:16.000 PM
17	10/9/2017	6:00:17.000 PM	10/9/2017	6:00:17.000 PM
18	10/9/2017	6:00:18.000 PM	10/9/2017	6:00:18.000 PM
19	10/9/2017	6:00:19.000 PM	10/9/2017	6:00:19.000 PM
20	10/9/2017	6:00:20.000 PM	10/9/2017	6:00:20.000 PM
21	10/9/2017	6:00:21.000 PM	10/9/2017	6:00:21.000 PM
22	10/9/2017	6:00:22.000 PM	10/9/2017	6:00:22.000 PM
23	10/9/2017	6:00:23.000 PM	10/9/2017	6:00:23.000 PM
24	10/9/2017	6:00:24.000 PM	10/9/2017	6:00:24.000 PM
25	10/9/2017	6:00:25.000 PM	10/9/2017	6:00:25.000 PM
26	10/9/2017	6:00:26.000 PM	10/9/2017	6:00:26.000 PM
27	10/9/2017	6:00:27.000 PM	10/9/2017	6:00:27.000 PM
28	10/9/2017	6:00:28.000 PM	10/9/2017	6:00:28.000 PM
29	10/9/2017	6:00:29.000 PM	10/9/2017	6:00:29.000 PM
...	...	...		...
172765	10/11/2017	5:59:25.000 PM	10/11/2017	5:59:25.000 PM
172766	10/11/2017	5:59:26.000 PM	10/11/2017	5:59:26.000 PM
172767	10/11/2017	5:59:27.000 PM	10/11/2017	5:59:27.000 PM
172768	10/11/2017	5:59:28.000 PM	10/11/2017	5:59:28.000 PM
172769	10/11/2017	5:59:29.000 PM	10/11/2017	5:59:29.000 PM
172770	10/11/2017	5:59:30.000 PM	10/11/2017	5:59:30.000 PM
172771	10/11/2017	5:59:31.000 PM	10/11/2017	5:59:31.000 PM
172772	10/11/2017	5:59:32.000 PM	10/11/2017	5:59:32.000 PM
172773	10/11/2017	5:59:33.000 PM	10/11/2017	5:59:33.000 PM
172774	10/11/2017	5:59:34.000 PM	10/11/2017	5:59:34.000 PM
172775	10/11/2017	5:59:35.000 PM	10/11/2017	5:59:35.000 PM
172776	10/11/2017	5:59:36.000 PM	10/11/2017	5:59:36.000 PM
172777	10/11/2017	5:59:37.000 PM	10/11/2017	5:59:37.000 PM
172778	10/11/2017	5:59:38.000 PM	10/11/2017	5:59:38.000 PM
172779	10/11/2017	5:59:39.000 PM	10/11/2017	5:59:39.000 PM
172780	10/11/2017	5:59:40.000 PM	10/11/2017	5:59:40.000 PM
172781	10/11/2017	5:59:41.000 PM	10/11/2017	5:59:41.000 PM
172782	10/11/2017	5:59:42.000 PM	10/11/2017	5:59:42.000 PM
172783	10/11/2017	5:59:43.000 PM	10/11/2017	5:59:43.000 PM
172784	10/11/2017	5:59:44.000 PM	10/11/2017	5:59:44.000 PM
172785	10/11/2017	5:59:45.000 PM	10/11/2017	5:59:45.000 PM
172786	10/11/2017	5:59:46.000 PM	10/11/2017	5:59:46.000 PM
172787	10/11/2017	5:59:47.000 PM	10/11/2017	5:59:47.000 PM
172788	10/11/2017	5:59:48.000 PM	10/11/2017	5:59:48.000 PM
172789	10/11/2017	5:59:49.000 PM	10/11/2017	5:59:49.000 PM
172790	10/11/2017	5:59:50.000 PM	10/11/2017	5:59:50.000 PM
172791	10/11/2017	5:59:51.000 PM	10/11/2017	5:59:51.000 PM
172792	10/11/2017	5:59:52.000 PM	10/11/2017	5:59:52.000 PM
172793	10/11/2017	5:59:53.000 PM	10/11/2017	5:59:53.000 PM
172794	10/11/2017	5:59:54.000 PM	10/11/2017	5:59:54.000 PM

```
          [172795 rows x 3 columns]
```

In [238]:
```
# Let's store these predictions in a separate column in P1_attack_predicted for bett
P1_attack_predicted['predictions'] = Isolation_Forest.predict(attack_PC)
P1_attack_predicted.head()
```

Out[238]:

	Date	Time	DateTime	datetime	predictions
0	10/9/2017	6:00:00.000 PM	10/9/2017 6:00:00.000 PM	2017-10-09 18:00:00	1
1	10/9/2017	6:00:01.000 PM	10/9/2017 6:00:01.000 PM	2017-10-09 18:00:01	1
2	10/9/2017	6:00:02.000 PM	10/9/2017 6:00:02.000 PM	2017-10-09 18:00:02	1
3	10/9/2017	6:00:03.000 PM	10/9/2017 6:00:03.000 PM	2017-10-09 18:00:03	1
4	10/9/2017	6:00:04.000 PM	10/9/2017 6:00:04.000 PM	2017-10-09 18:00:04	1

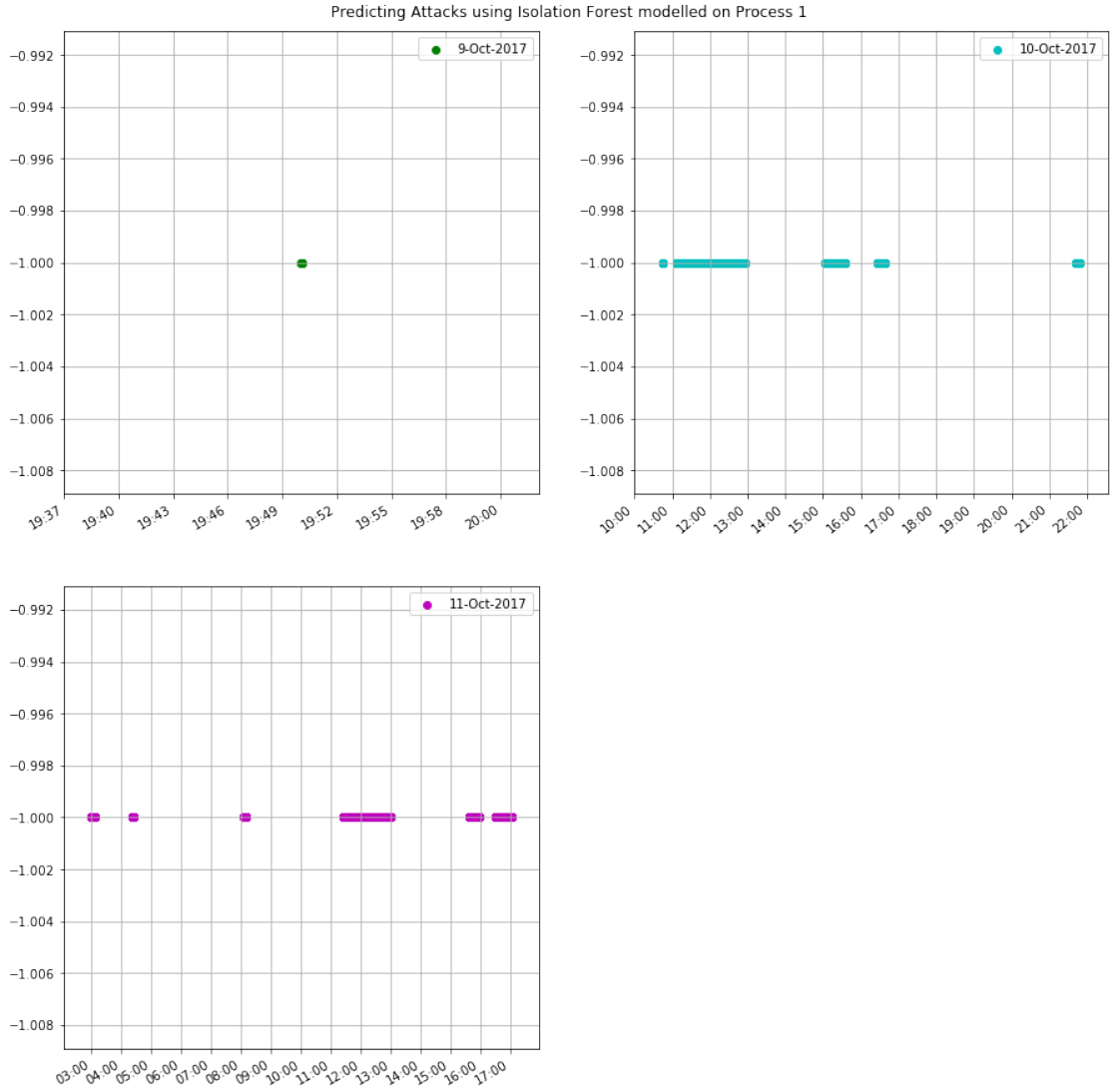
In [618]:
```
# Let's slice the entries which are predicted as outlier (attacked).
# Here, -1 is the notation for attack predictions
predicted_attacks = P1_attack_predicted[P1_attack_predicted['predictions']==-1]
```

In [619]:
```
predicted_attacks['datetime'] = list(pd.to_datetime(predicted_attacks['DateTime']))
```

```
/data/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```

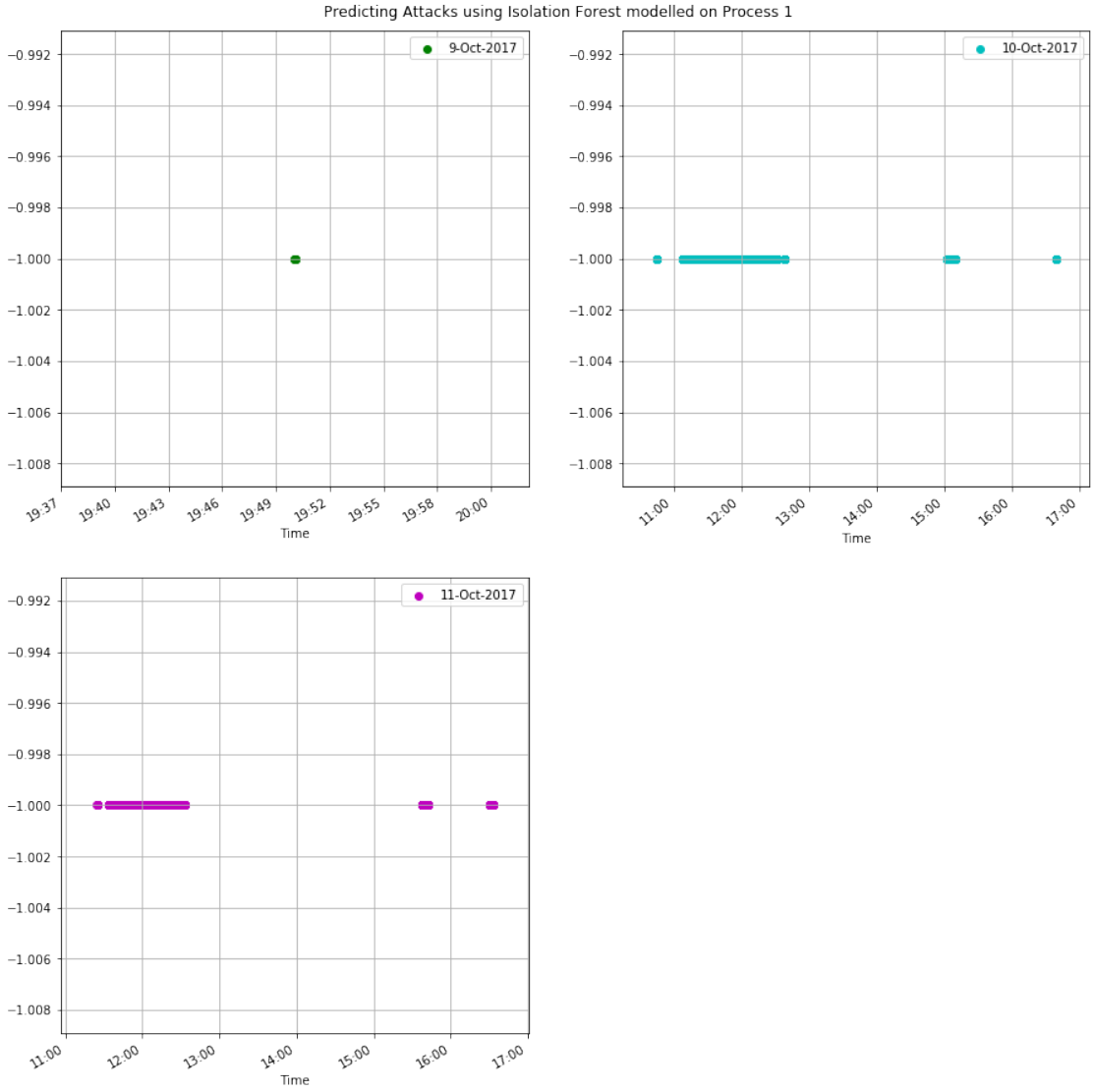
In [527]:
```
# Above plot can be simplified for better understanding

# Let's first specify some customized formatting for the plot
#fig, axs = plt.subplots(3,figsize=(10,10))
figure = plt.figure(figsize=(15,15))
formatter = mpldates.DateFormatter('%H:%M')
plt.suptitle('Predicting Attacks using Isolation Forest modelled on Process 1',y=0.9

#Plot for 9th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predi
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,1)
axes.xaxis.set_major_formatter(formatter)
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/9/2017']['predictions
```

```
axes.legend(['9-Oct-2017'])
#plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=30, ha='right')
plt.grid()

#Plot for 10th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predi
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,2)
axes.xaxis.set_major_formatter(formatter)
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/10/2017']['predictio
axes.legend(['10-Oct-2017'])
plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=40, ha='right')
plt.grid()

#Plot for 11th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predi
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,3)
axes.xaxis.set_major_formatter(formatter)
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/11/2017']['predictio
axes.legend(['11-Oct-2017'])
plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=30, ha='right')
plt.grid()

plt.show()
```



In [620]:
```
# Above plot can be simplified for better understanding

# Let's first specify some customized formatting for the plot
#fig, axs = plt.subplots(3,figsize=(10,10))
figure = plt.figure(figsize=(15,15))
formatter = mpldates.DateFormatter('%H:%M')
plt.suptitle('Predicting Attacks using Isolation Forest modelled on Process 1',y=0.9

#Plot for 9th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predi
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,1)
axes.xaxis.set_major_formatter(formatter)
```

```
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/9/2017']['predictions
axes.legend(['9-Oct-2017'])
axes.set_xlabel('Time')
#plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=30, ha='right')
plt.grid()

#Plot for 10th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predi
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,2)
axes.xaxis.set_major_formatter(formatter)
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/10/2017']['predictio
axes.legend(['10-Oct-2017'])
axes.set_xlabel('Time')
plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=40, ha='right')
plt.grid()

#Plot for 11th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predi
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,3)
axes.xaxis.set_major_formatter(formatter)
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/11/2017']['predictio
axes.legend(['11-Oct-2017'])
axes.set_xlabel('Time')
plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=30, ha='right')
plt.grid()

plt.show()
```



2.3.1 Some observations from above plot:

- Isolation Forest has been able to predict more instances as normal compared to One Class SVM.
- Although, Isolation Forest has reduced number of false alarms by a signifant amount but on the counter part it's less precise and delayed in detecting attacks. This technique has been able to locate only some major deviations.

Time to compute the Confusion Matrix to check the performance of our model.

```
In [158]: from sklearn.metrics import confusion_matrix
          from sklearn.metrics import accuracy_score
          from sklearn.metrics import classification_report
```

```
In [242]: print(confusion_matrix(actual_attacks['state'],P1_attack_predicted['predictions'],lab

[[157635   5212]
 [  6671   3277]]
```

```
In [240]: print('Accuracy Score:', accuracy_score(actual_attacks['state'],P1_attack_predicted[

Accuracy Score: 0.9312306490349836
```

```
In [241]: print(classification_report(actual_attacks['state'],P1_attack_predicted['predictions
```

	precision	recall	f1-score	support
-1	0.39	0.33	0.36	9948
1	0.96	0.97	0.96	162847
micro avg	0.93	0.93	0.93	172795
macro avg	0.67	0.65	0.66	172795
weighted avg	0.93	0.93	0.93	172795

2.4 ---> Elliptic Envelope on Principal Component reconstructed Process 1 normal data

```
In [621]: from sklearn.covariance import EllipticEnvelope
```

```
In [218]: # Here again, we'll tune the parameters in accordance to provide the best outputs.
          Elliptic_Envelope = EllipticEnvelope(contamination=0.001)
```

```
In [219]: Elliptic_Envelope.fit(normal_PC.sample(100000))

Out[219]: EllipticEnvelope(assume_centered=False, contamination=0.001,
                           random_state=None, store_precision=True, support_fraction=None)
```

```
In [220]: # Let's predict how many data points has been identified as normal in attack data.
          print(np.count_nonzero(Elliptic_Envelope.predict(attack_PC)==1),'data points has been

157496 data points has been identified as normal
```

```
In [670]: # Here again, before predicting attack data let's first make a space to store the pr
          # corresponding timeframe by defining a variable
          P1_attack_predicted = attack_data.loc[attack_PC.index][['Date','Time']]
          P1_attack_predicted.head()
```

Out[670]:

	Date	Time
0	10/9/2017	6:00:00.000 PM
1	10/9/2017	6:00:01.000 PM
2	10/9/2017	6:00:02.000 PM
3	10/9/2017	6:00:03.000 PM
4	10/9/2017	6:00:04.000 PM

In [671]:
```
# Let's now join the date and time column in order to perform 'pd.to_datetime' opera
P1_attack_predicted['DateTime'] = P1_attack_predicted['Date']+' '+P1_attack_predicte
P1_attack_predicted
```

Out[671]:

	Date	Time	DateTime
0	10/9/2017	6:00:00.000 PM	10/9/2017 6:00:00.000 PM
1	10/9/2017	6:00:01.000 PM	10/9/2017 6:00:01.000 PM
2	10/9/2017	6:00:02.000 PM	10/9/2017 6:00:02.000 PM
3	10/9/2017	6:00:03.000 PM	10/9/2017 6:00:03.000 PM
4	10/9/2017	6:00:04.000 PM	10/9/2017 6:00:04.000 PM
5	10/9/2017	6:00:05.000 PM	10/9/2017 6:00:05.000 PM
6	10/9/2017	6:00:06.000 PM	10/9/2017 6:00:06.000 PM
7	10/9/2017	6:00:07.000 PM	10/9/2017 6:00:07.000 PM
8	10/9/2017	6:00:08.000 PM	10/9/2017 6:00:08.000 PM
9	10/9/2017	6:00:09.000 PM	10/9/2017 6:00:09.000 PM
10	10/9/2017	6:00:10.000 PM	10/9/2017 6:00:10.000 PM
11	10/9/2017	6:00:11.000 PM	10/9/2017 6:00:11.000 PM
12	10/9/2017	6:00:12.000 PM	10/9/2017 6:00:12.000 PM
13	10/9/2017	6:00:13.000 PM	10/9/2017 6:00:13.000 PM
14	10/9/2017	6:00:14.000 PM	10/9/2017 6:00:14.000 PM
15	10/9/2017	6:00:15.000 PM	10/9/2017 6:00:15.000 PM
16	10/9/2017	6:00:16.000 PM	10/9/2017 6:00:16.000 PM
17	10/9/2017	6:00:17.000 PM	10/9/2017 6:00:17.000 PM
18	10/9/2017	6:00:18.000 PM	10/9/2017 6:00:18.000 PM
19	10/9/2017	6:00:19.000 PM	10/9/2017 6:00:19.000 PM
20	10/9/2017	6:00:20.000 PM	10/9/2017 6:00:20.000 PM
21	10/9/2017	6:00:21.000 PM	10/9/2017 6:00:21.000 PM
22	10/9/2017	6:00:22.000 PM	10/9/2017 6:00:22.000 PM
23	10/9/2017	6:00:23.000 PM	10/9/2017 6:00:23.000 PM
24	10/9/2017	6:00:24.000 PM	10/9/2017 6:00:24.000 PM
25	10/9/2017	6:00:25.000 PM	10/9/2017 6:00:25.000 PM
26	10/9/2017	6:00:26.000 PM	10/9/2017 6:00:26.000 PM
27	10/9/2017	6:00:27.000 PM	10/9/2017 6:00:27.000 PM
28	10/9/2017	6:00:28.000 PM	10/9/2017 6:00:28.000 PM
29	10/9/2017	6:00:29.000 PM	10/9/2017 6:00:29.000 PM
...	...	...	...
172765	10/11/2017	5:59:25.000 PM	10/11/2017 5:59:25.000 PM
172766	10/11/2017	5:59:26.000 PM	10/11/2017 5:59:26.000 PM
172767	10/11/2017	5:59:27.000 PM	10/11/2017 5:59:27.000 PM
172768	10/11/2017	5:59:28.000 PM	10/11/2017 5:59:28.000 PM
172769	10/11/2017	5:59:29.000 PM	10/11/2017 5:59:29.000 PM

```
172770  10/11/2017  5:59:30.000 PM  10/11/2017 5:59:30.000 PM
172771  10/11/2017  5:59:31.000 PM  10/11/2017 5:59:31.000 PM
172772  10/11/2017  5:59:32.000 PM  10/11/2017 5:59:32.000 PM
172773  10/11/2017  5:59:33.000 PM  10/11/2017 5:59:33.000 PM
172774  10/11/2017  5:59:34.000 PM  10/11/2017 5:59:34.000 PM
172775  10/11/2017  5:59:35.000 PM  10/11/2017 5:59:35.000 PM
172776  10/11/2017  5:59:36.000 PM  10/11/2017 5:59:36.000 PM
172777  10/11/2017  5:59:37.000 PM  10/11/2017 5:59:37.000 PM
172778  10/11/2017  5:59:38.000 PM  10/11/2017 5:59:38.000 PM
172779  10/11/2017  5:59:39.000 PM  10/11/2017 5:59:39.000 PM
172780  10/11/2017  5:59:40.000 PM  10/11/2017 5:59:40.000 PM
172781  10/11/2017  5:59:41.000 PM  10/11/2017 5:59:41.000 PM
172782  10/11/2017  5:59:42.000 PM  10/11/2017 5:59:42.000 PM
172783  10/11/2017  5:59:43.000 PM  10/11/2017 5:59:43.000 PM
172784  10/11/2017  5:59:44.000 PM  10/11/2017 5:59:44.000 PM
172785  10/11/2017  5:59:45.000 PM  10/11/2017 5:59:45.000 PM
172786  10/11/2017  5:59:46.000 PM  10/11/2017 5:59:46.000 PM
172787  10/11/2017  5:59:47.000 PM  10/11/2017 5:59:47.000 PM
172788  10/11/2017  5:59:48.000 PM  10/11/2017 5:59:48.000 PM
172789  10/11/2017  5:59:49.000 PM  10/11/2017 5:59:49.000 PM
172790  10/11/2017  5:59:50.000 PM  10/11/2017 5:59:50.000 PM
172791  10/11/2017  5:59:51.000 PM  10/11/2017 5:59:51.000 PM
172792  10/11/2017  5:59:52.000 PM  10/11/2017 5:59:52.000 PM
172793  10/11/2017  5:59:53.000 PM  10/11/2017 5:59:53.000 PM
172794  10/11/2017  5:59:54.000 PM  10/11/2017 5:59:54.000 PM

[172795 rows x 3 columns]
```

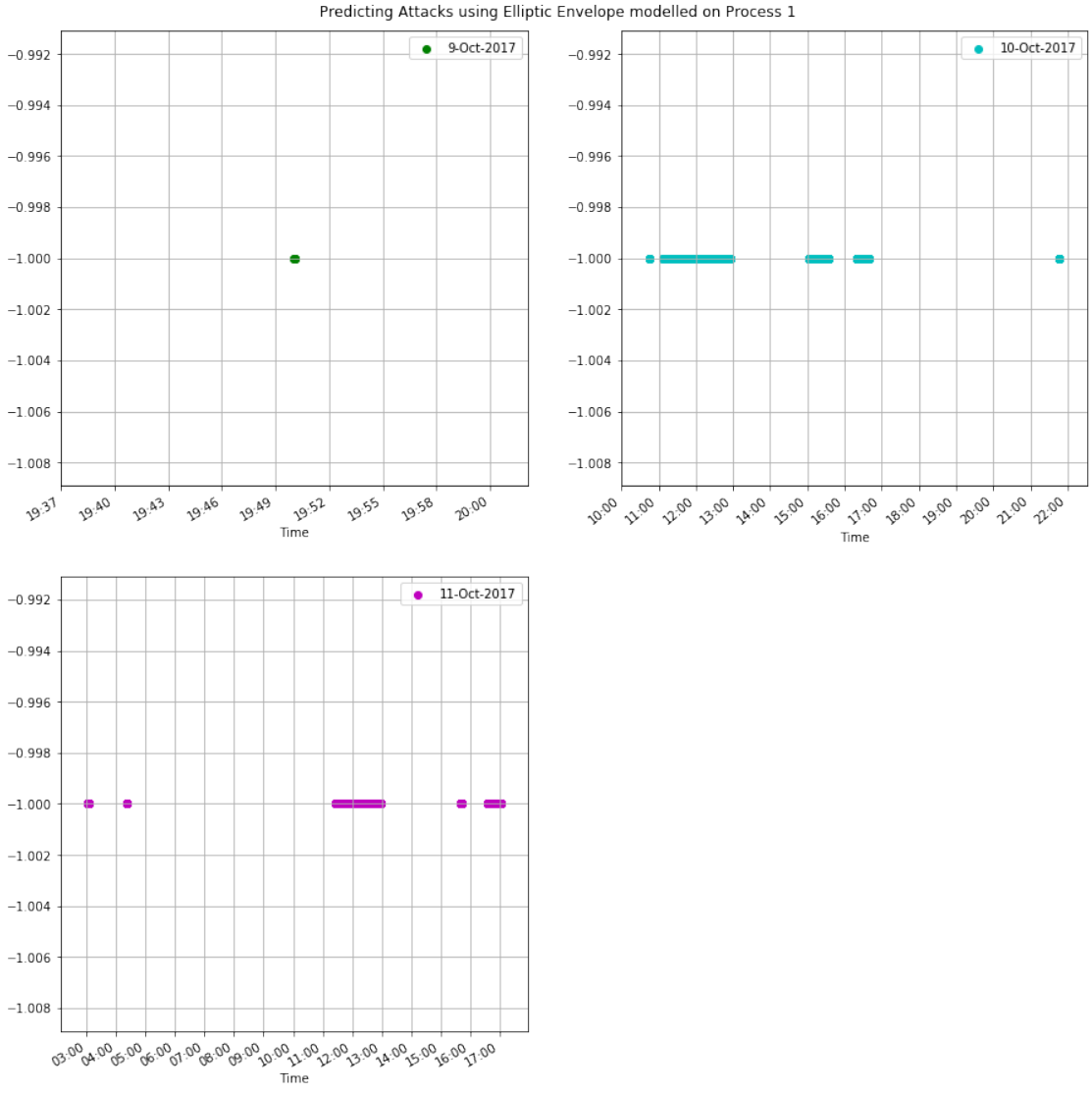
In [221]:
```
# Let's store these predictions in a separate column in P1_attack_predicted for bett
P1_attack_predicted['predictions'] = Elliptic_Envelope.predict(attack_PC)
P1_attack_predicted.head()
```

Out[221]:
```
        Date            Time                  DateTime            datetime  \
0  10/9/2017  6:00:00.000 PM  10/9/2017 6:00:00.000 PM 2017-10-09 18:00:00
1  10/9/2017  6:00:01.000 PM  10/9/2017 6:00:01.000 PM 2017-10-09 18:00:01
2  10/9/2017  6:00:02.000 PM  10/9/2017 6:00:02.000 PM 2017-10-09 18:00:02
3  10/9/2017  6:00:03.000 PM  10/9/2017 6:00:03.000 PM 2017-10-09 18:00:03
4  10/9/2017  6:00:04.000 PM  10/9/2017 6:00:04.000 PM 2017-10-09 18:00:04

   predictions
0            1
1            1
2            1
3            1
4            1
```

In [673]:
```
# Let's slice the entries which are predicted as outlier (attacked).
# Here, -1 is the notation for attack predictions
predicted_attacks = P1_attack_predicted[P1_attack_predicted['predictions']==-1]
```

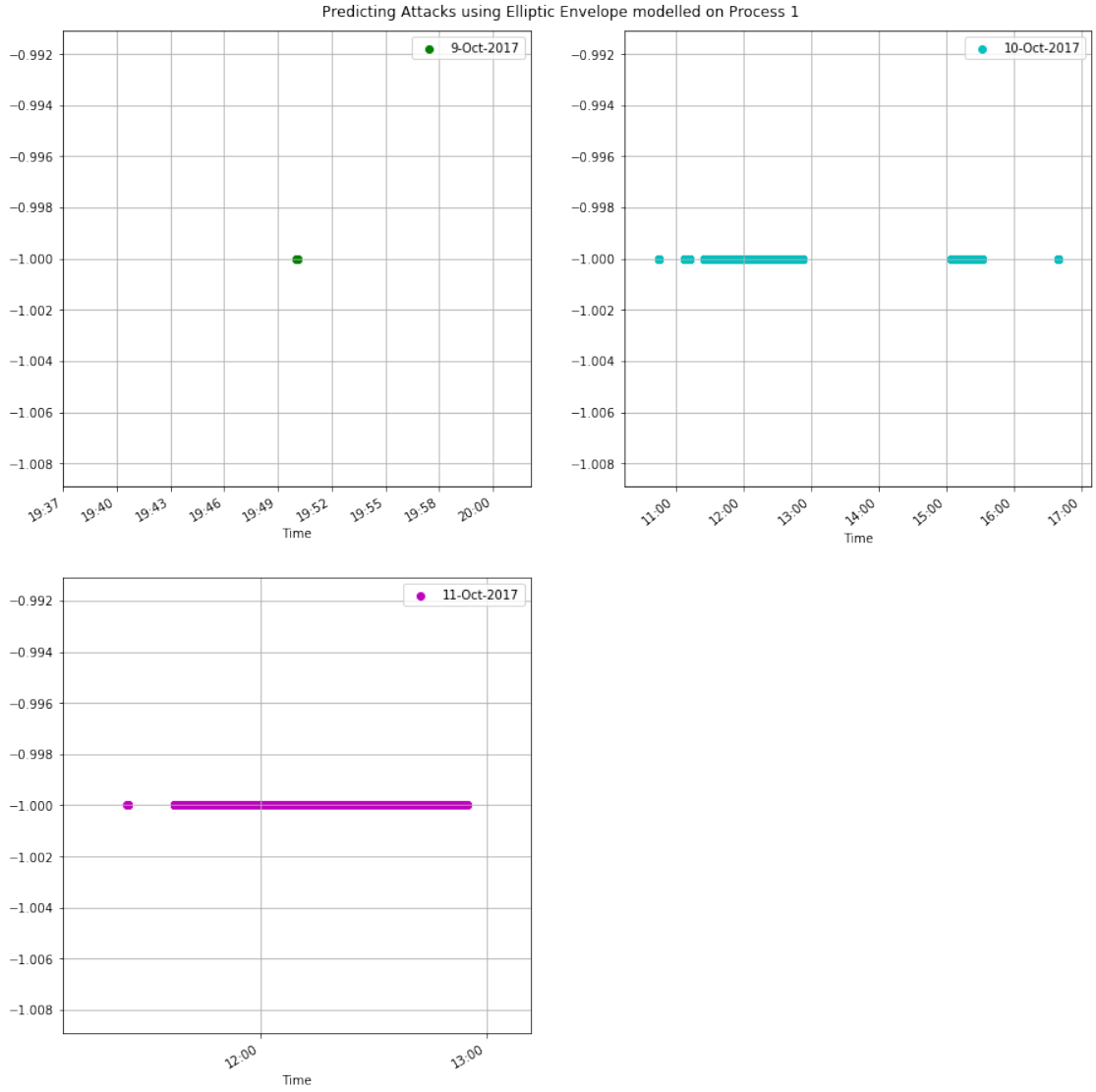
```
In [674]: predicted_attacks['datetime'] = list(pd.to_datetime(predicted_attacks['DateTime']))

/data/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.


In [660]: # Above plot can be simplified for better understanding

          # Let's first specify some customized formatting for the plot
          #fig, axs = plt.subplots(3,figsize=(10,10))
          figure = plt.figure(figsize=(15,15))
          formatter = mpldates.DateFormatter('%H:%M')
          plt.suptitle('Predicting Attacks using Elliptic Envelope modelled on Process 1',y=0.9


          #Plot for 9th Oct 2017
          a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predic
          x = mpldates.date2num(a)
          axes = figure.add_subplot(2,2,1)
          axes.xaxis.set_major_formatter(formatter)
          axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/9/2017']['predictions
          axes.legend(['9-Oct-2017'])
          axes.set_xlabel('Time')
          #plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
          plt.setp(plt.xticks()[1], rotation=30, ha='right')
          plt.grid()

          #Plot for 10th Oct 2017
          a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predic
          x = mpldates.date2num(a)
          axes = figure.add_subplot(2,2,2)
          axes.xaxis.set_major_formatter(formatter)
          axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/10/2017']['predictio
          axes.legend(['10-Oct-2017'])
          axes.set_xlabel('Time')
          plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
          plt.setp(plt.xticks()[1], rotation=40, ha='right')
          plt.grid()

          #Plot for 11th Oct 2017
          a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predic
          x = mpldates.date2num(a)
          axes = figure.add_subplot(2,2,3)
          axes.xaxis.set_major_formatter(formatter)
```

```
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/11/2017']['prediction
axes.legend(['11-Oct-2017'])
axes.set_xlabel('Time')
plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=30, ha='right')
plt.grid()

plt.show()
```



In [675]:
```
# Above plot can be simplified for better understanding

# Let's first specify some customized formatting for the plot
#fig, axs = plt.subplots(3,figsize=(10,10))
```

```
figure = plt.figure(figsize=(15,15))
formatter = mpldates.DateFormatter('%H:%M')
plt.suptitle('Predicting Attacks using Elliptic Envelope modelled on Process 1',y=0.9


#Plot for 9th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predic
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,1)
axes.xaxis.set_major_formatter(formatter)
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/9/2017']['predictions
axes.legend(['9-Oct-2017'])
axes.set_xlabel('Time')
#plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=30, ha='right')
plt.grid()

#Plot for 10th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predic
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,2)
axes.xaxis.set_major_formatter(formatter)
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/10/2017']['predictio
axes.legend(['10-Oct-2017'])
axes.set_xlabel('Time')
plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=40, ha='right')
plt.grid()

#Plot for 11th Oct 2017
a = [datetime.strptime(str(d), '%Y-%m-%d %H:%M:%S') for d in predicted_attacks[predic
x = mpldates.date2num(a)
axes = figure.add_subplot(2,2,3)
axes.xaxis.set_major_formatter(formatter)
axes.scatter(x,predicted_attacks[predicted_attacks['Date']=='10/11/2017']['predictio
axes.legend(['11-Oct-2017'])
axes.set_xlabel('Time')
plt.gca().xaxis.set_major_locator(mpldates.HourLocator())
plt.setp(plt.xticks()[1], rotation=30, ha='right')
plt.grid()

plt.show()
```



Time to compute the Confusion Matrix to check the performance of our model.

```
In [158]: from sklearn.metrics import confusion_matrix
          from sklearn.metrics import accuracy_score
          from sklearn.metrics import classification_report

In [222]: print(confusion_matrix(actual_attacks['state'],P1_attack_predicted['predictions'],la

[[150770  12077]
 [  6726   3222]]


In [223]: print('Accuracy Score:', accuracy_score(actual_attacks['state'],P1_attack_predicted[
```

```
Accuracy Score: 0.8911831939581585
```

```
In [224]: print(classification_report(actual_attacks['state'],P1_attack_predicted['predictions
```

```
              precision    recall  f1-score   support

          -1       0.21      0.32      0.26      9948
           1       0.96      0.93      0.94    162847

   micro avg       0.89      0.89      0.89    172795
   macro avg       0.58      0.62      0.60    172795
weighted avg       0.91      0.89      0.90    172795
```

2.4.1 Here, we conclude the analysis of Process 1. Next we'll start analyzing Process 2.

3 Process P2

3.0.1 Here again, we'll proceed with almost similar strategy to above.

```
In [6]: # Let's first extract Process 2 normal data variables
        P2_normal = normal_data_sorted[normal_data_sorted.columns[19:105]]
```

```
In [7]: # There is a need to remove some variables from above data before applying PCA.
        variables_toremove = list(normal_data_sorted[normal_data_sorted.columns[19:105]].colum
```

```
In [8]: P2_normal.shape
```

```
Out[8]: (1048571, 86)
```

```
In [9]: P2_normal = P2_normal.drop(variables_toremove, axis=1)
        P2_normal.shape
```

```
Out[9]: (1048571, 75)
```

Those 11 variables has been succesfully removed.

```
In [10]: # Time to implement PCA on this data
         pca = PCA(0.95)
```

```
In [11]: pca.fit(P2_normal)
```

```
Out[11]: PCA(copy=True, iterated_power='auto', n_components=0.95, random_state=None,
           svd_solver='auto', tol=0.0, whiten=False)
```

```
In [12]: pca.explained_variance_ratio_
```

```
Out[12]: array([0.65407832, 0.15775298, 0.06693462, 0.02806778, 0.0202405 ,
                0.01516197, 0.01360239])
```

3.0.2 Here, 7 principal components has been formed which contains about 95% of total data variance.

```
In [384]: P2_normal_PC = pca.transform(P2_normal)
```

```
In [385]: P2_normal_PC = pd.DataFrame(data = P2_normal_PC
                        , columns = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7'])
```

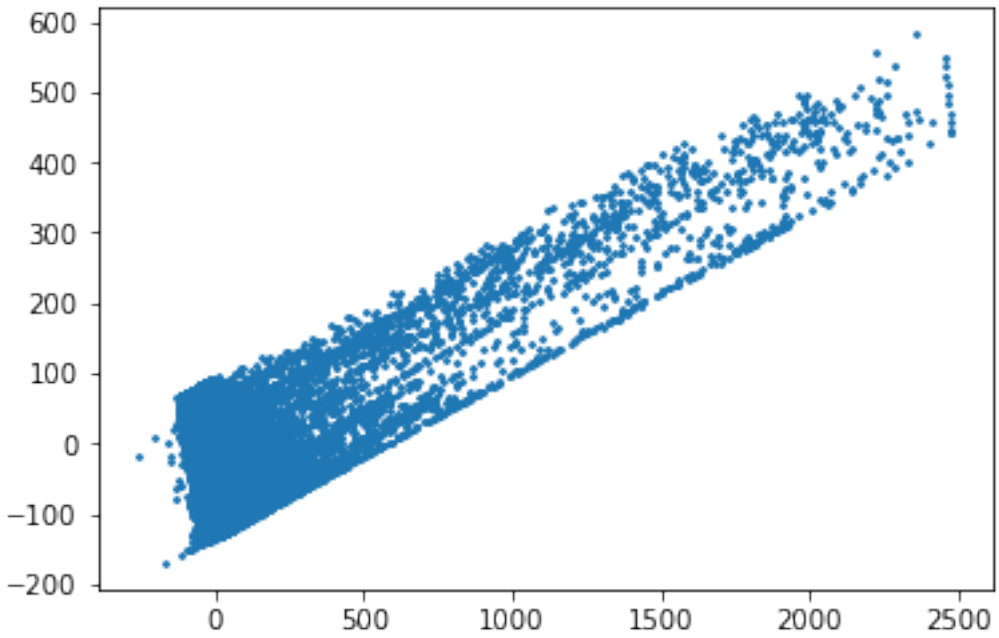
```
In [386]: P2_normal_PC.describe()
```

Out[386]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
count	1048571.000	1048571.000	1048571.000	1048571.000	1048571.000	1048571.000	1048571.000
mean	-0.000	-0.000	0.000	-0.000	0.000	0.000	0.000
std	138.865	68.197	44.423	28.766	24.428	21.142	20.026
min	-260.626	-169.351	-83.026	-87.787	-63.056	-93.621	-89.970
25%	-65.299	-62.890	-32.229	-10.584	-16.772	-6.048	-7.615
50%	-23.538	10.916	-7.512	-1.130	-1.298	0.156	0.540
75%	23.395	59.861	24.071	12.221	10.345	4.954	6.014
max	2478.567	582.280	150.553	120.475	149.577	97.695	96.591

```
In [387]: # It looks like there are some extreme outliers points in above data. Let's try to v
          plt.scatter(P2_normal_PC['PC1'],P2_normal_PC['PC2'],s=1)
```

```
Out[387]: <matplotlib.collections.PathCollection at 0x7fe3ccbaf3c8>
```



Now let's try to prepare attack data before applying to PCA

```
In [16]: P2_attack = attack_data[attack_data.columns[22:112]]

In [17]: P2_attack.shape

Out[17]: (172801, 90)

In [18]: # Here again we need to remove some unwanted variables
         P2_attack = P2_attack.drop(list(P2_attack.columns[[28,29,42,51,52,64,65,82,83,84,85,8

In [19]: P2_attack.shape

Out[19]: (172801, 75)
```

Now the columns of the attack data has been successfully reduced to 75.

```
In [20]: P2_attack_PC = pca.transform(P2_attack)

In [21]: P2_attack_PC = pd.DataFrame(data = P2_attack_PC
                       , columns = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7'])

In [22]: P2_attack_PC.describe()

Out[22]:              PC1         PC2         PC3         PC4         PC5         PC6  \
         count 172801.000 172801.000 172801.000 172801.000 172801.000 172801.000
         mean       4.694       3.455       4.360      -1.662       1.300      -0.569
```

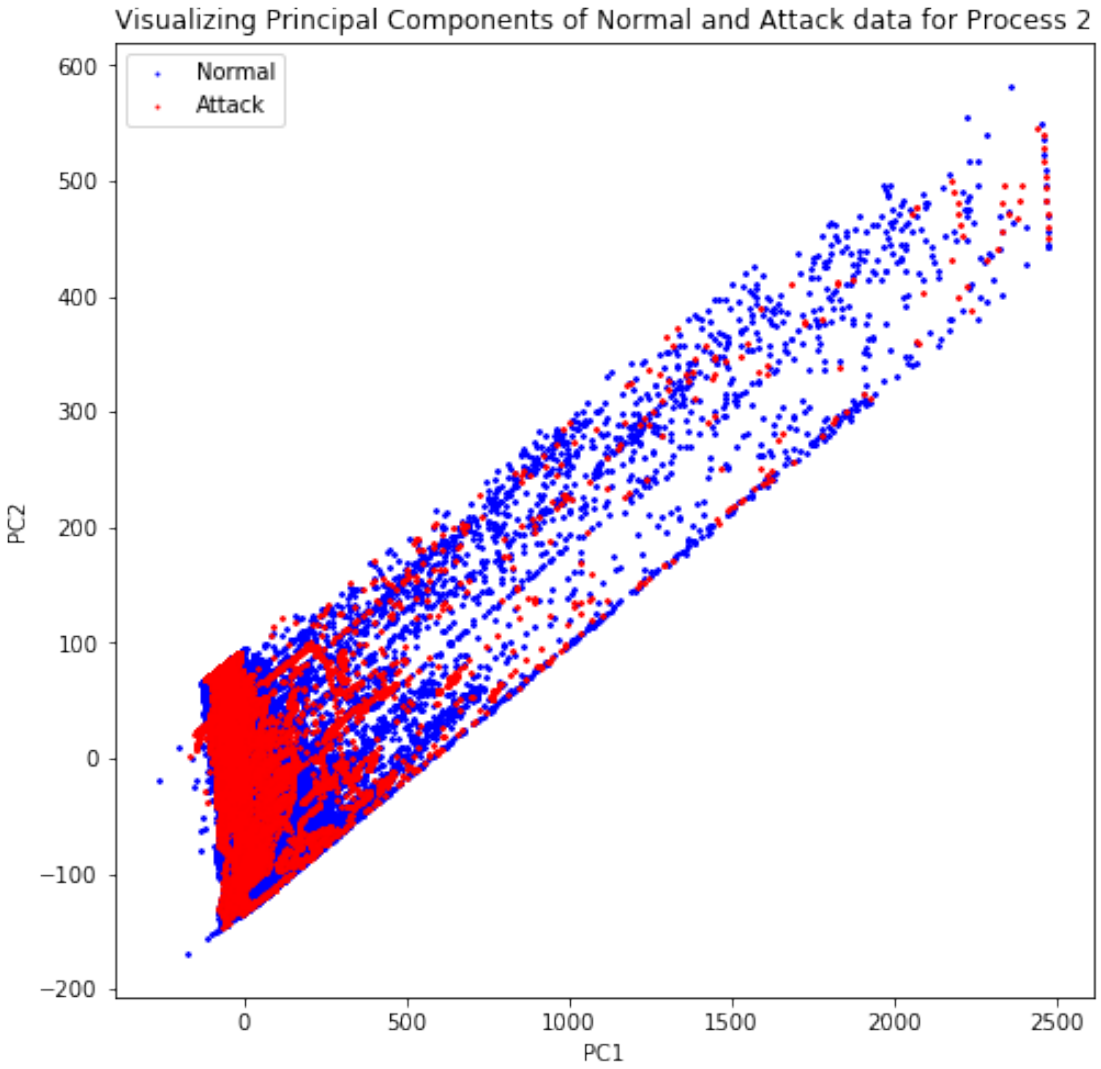
```
std      153.206     69.627     45.900     26.236     27.917     21.111
min     -167.941   -145.659    -81.305    -87.397    -62.582    -75.730
25%      -60.813    -59.318    -19.194    -11.783    -16.723     -4.044
50%      -26.393     17.275     -1.239     -1.312     -0.947      0.246
75%       21.898     63.571     29.194     10.913     12.236      3.412
max     2477.447    546.502    122.395     71.902    168.799     73.534
```

```
                PC7
count 172801.000
mean       -2.403
std        18.400
min       -90.155
25%        -7.949
50%         0.288
75%         3.896
max        54.902
```

In [50]:
```
pca.explained_variance_ratio_
```

Out[50]:
```
array([0.68539998, 0.1398425 , 0.06227493, 0.02575497, 0.02229811,
       0.01504041])
```

In [23]:
```
# Now, let's try to see whether by plotting only first two PC, can we differentiate
# between the data points.
%matplotlib inline
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,8))
ax = plt.axes()
#sample_normal = normal_PC.sample(n=100000)
ax.scatter(P2_normal_PC['PC1'],P2_normal_PC['PC2'], s=1,c ='b');
ax.scatter(P2_attack_PC['PC1'],P2_attack_PC['PC2'], s=1,c ='r');
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.legend(['Normal','Attack'])
ax.set_title('Visualizing Principal Components of Normal and Attack data for Process 1
plt.show()
```




```
In [315]: # It is visible from above plot there is a huge need to remove some outliers from no
          # an effective model.
          P2_normal_PC = P2_normal_PC[(P2_normal_PC['PC1']>-120)&(P2_normal_PC['PC1']<100)]
          P2_normal_PC = P2_normal_PC[(P2_normal_PC['PC2']>-130)&(P2_normal_PC['PC2']<90)]

In [470]: P2_normal_PC.describe()

Out[470]:              PC1         PC2         PC3         PC4         PC5         PC6  \
          count 951653.000 951653.000 951653.000 951653.000 951653.000 951653.000
          mean     -27.288      0.414      -0.970      -0.029       1.326       0.340
          std       50.167     65.406      44.118      28.842      24.438      21.422
          min     -120.000   -156.798     -81.132     -87.557     -63.056     -93.621
          25%      -67.893    -65.740     -33.997      -9.809     -16.751      -5.880
          50%      -30.282     18.615      -8.896      -1.537      -0.762       0.222
```

```
75%        9.763     61.341     23.765     11.178     11.416      5.207
max       99.986    107.532     98.053    102.210    144.493     91.975

                PC7
count 951653.000
mean       -0.095
std        19.617
min       -82.254
25%        -7.738
50%         0.542
75%         6.051
max        94.782
```

In [474]:
```
P2_normal_PC[P2_normal_PC['PC2']<-130]
```

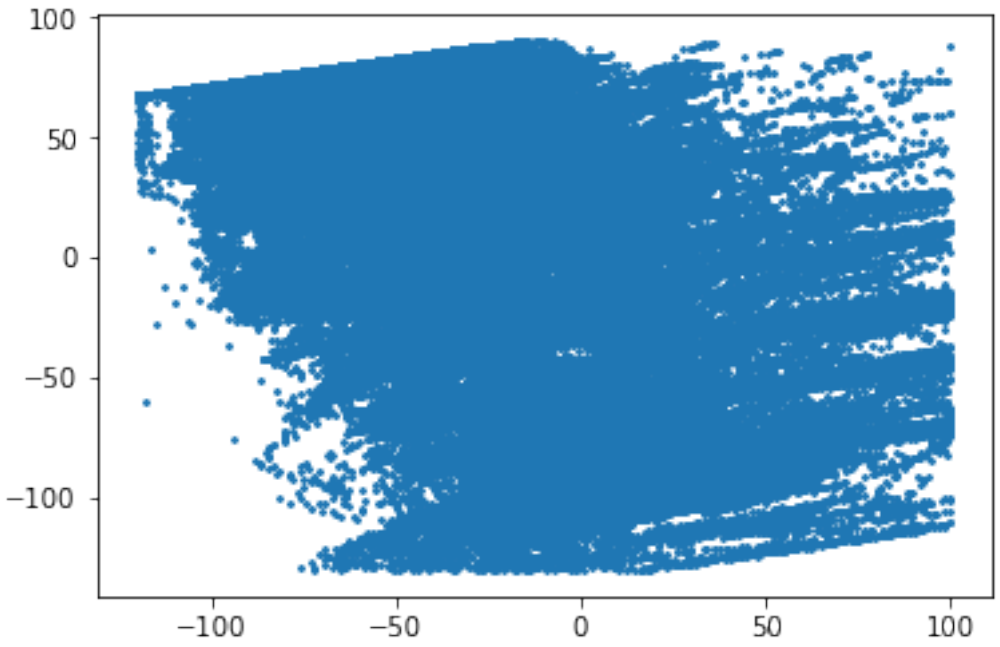
Out[474]:

	PC1	PC2	PC3	PC4	PC5	PC6	PC7
6999	-6.490	-132.957	-9.895	16.787	-23.381	-5.599	2.517
7000	-6.490	-132.957	-9.895	16.787	-23.381	-5.599	2.517
7001	-6.490	-132.957	-9.895	16.787	-23.381	-5.599	2.517
7002	-6.490	-132.957	-9.895	16.787	-23.381	-5.599	2.517
7003	-6.490	-132.957	-9.895	16.787	-23.381	-5.599	2.517
7004	-12.637	-133.070	-10.258	16.375	-21.999	-5.388	2.481
7005	-12.637	-133.070	-10.258	16.375	-21.999	-5.388	2.481
7006	-12.637	-133.070	-10.258	16.375	-21.999	-5.388	2.481
7007	-12.637	-133.070	-10.258	16.375	-21.999	-5.388	2.481
7008	-12.637	-133.070	-10.258	16.375	-21.999	-5.388	2.481
7009	-12.637	-133.070	-10.258	16.375	-21.999	-5.388	2.481
7010	-13.378	-131.698	-10.315	15.879	-20.209	-4.988	2.249
7011	-13.378	-131.698	-10.315	15.879	-20.209	-4.988	2.249
7012	-13.378	-131.698	-10.315	15.879	-20.209	-4.988	2.249
7013	-13.378	-131.698	-10.315	15.879	-20.209	-4.988	2.249
7014	-13.378	-131.698	-10.315	15.879	-20.209	-4.988	2.249
7015	-13.378	-131.698	-10.315	15.879	-20.209	-4.988	2.249
22481	2.396	-131.329	-9.136	16.555	-23.877	-4.993	1.074
22482	2.396	-131.329	-9.136	16.555	-23.877	-4.993	1.074
22483	2.396	-131.329	-9.136	16.555	-23.877	-4.993	1.074
22484	2.396	-131.329	-9.136	16.555	-23.877	-4.993	1.074
22485	2.396	-131.329	-9.136	16.555	-23.877	-4.993	1.074
22486	2.396	-131.329	-9.136	16.555	-23.877	-4.993	1.074
22487	0.851	-130.121	-9.303	16.044	-21.854	-4.393	0.861
22488	0.851	-130.121	-9.303	16.044	-21.854	-4.393	0.861
22489	0.851	-130.121	-9.303	16.044	-21.854	-4.393	0.861
22490	0.851	-130.121	-9.303	16.044	-21.854	-4.393	0.861
22491	0.851	-130.121	-9.303	16.044	-21.854	-4.393	0.861
23942	-24.422	-137.761	-9.466	16.735	-24.804	-6.523	1.676
23943	-24.422	-137.761	-9.466	16.735	-24.804	-6.523	1.676
...	...	...	...	...	...	...	...
1036802	-21.025	-133.053	-9.405	14.137	-18.293	-5.607	1.716

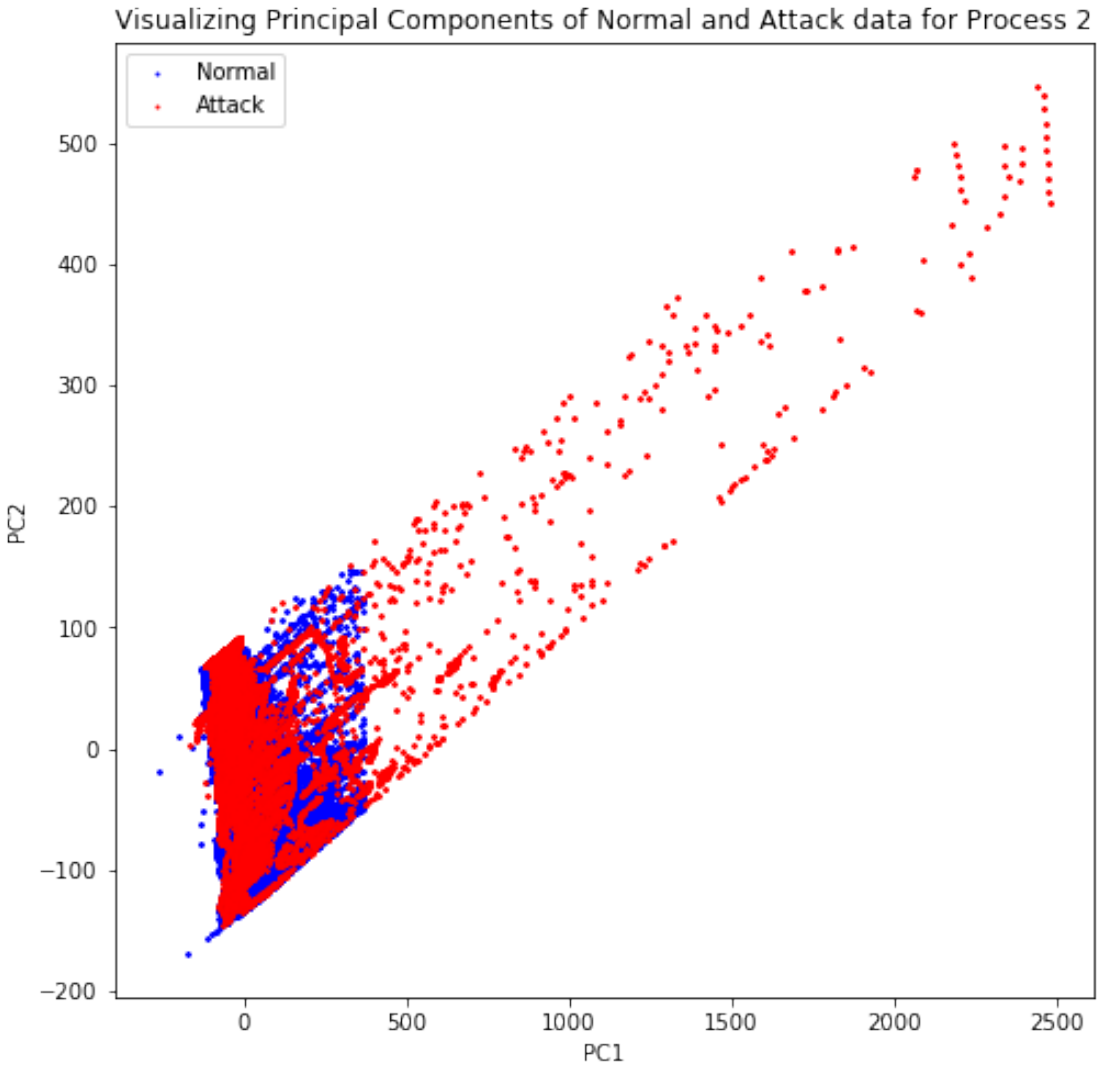
```
1036803 -21.025 -133.053  -9.405 14.137 -18.293 -5.607 1.716
1036804 -26.832 -132.850  -9.601 13.282 -16.391 -5.494 1.902
1036805 -26.832 -132.850  -9.601 13.282 -16.391 -5.494 1.902
1036806 -26.832 -132.850  -9.601 13.282 -16.391 -5.494 1.902
1036807 -26.832 -132.850  -9.601 13.282 -16.391 -5.494 1.902
1036808 -26.832 -132.850  -9.601 13.282 -16.391 -5.494 1.902
1036809 -25.018 -130.845  -9.535 12.545 -14.479 -5.333 2.142
1036810 -25.018 -130.845  -9.535 12.545 -14.479 -5.333 2.142
1036811 -25.018 -130.845  -9.535 12.545 -14.479 -5.333 2.142
1036812 -25.018 -130.845  -9.535 12.545 -14.479 -5.333 2.142
1036813 -25.018 -130.845  -9.535 12.545 -14.479 -5.333 2.142
1036814 -25.018 -130.845  -9.535 12.545 -14.479 -5.333 2.142
1037462 -29.175 -136.828 -10.316 14.879 -20.934 -5.821 2.887
1037463 -29.175 -136.828 -10.316 14.879 -20.934 -5.821 2.887
1037464 -29.175 -136.828 -10.316 14.879 -20.934 -5.821 2.887
1037465 -29.175 -136.828 -10.316 14.879 -20.934 -5.821 2.887
1037466 -29.175 -136.828 -10.316 14.879 -20.934 -5.821 2.887
1037467 -29.175 -136.828 -10.316 14.879 -20.934 -5.821 2.887
1037468 -12.292 -131.194  -9.906 14.426 -17.788 -5.151 3.213
1037469 -12.292 -131.194  -9.906 14.426 -17.788 -5.151 3.213
1037470 -12.292 -131.194  -9.906 14.426 -17.788 -5.151 3.213
1037471 -12.292 -131.194  -9.906 14.426 -17.788 -5.151 3.213
1037472 -12.292 -131.194  -9.906 14.426 -17.788 -5.151 3.213
1037473 -13.852 -130.021  -9.974 13.636 -15.886 -4.985 3.435
1037474 -13.852 -130.021  -9.974 13.636 -15.886 -4.985 3.435
1037475 -13.852 -130.021  -9.974 13.636 -15.886 -4.985 3.435
1037476 -13.852 -130.021  -9.974 13.636 -15.886 -4.985 3.435
1037477 -13.852 -130.021  -9.974 13.636 -15.886 -4.985 3.435
1037478 -13.852 -130.021  -9.974 13.636 -15.886 -4.985 3.435

[2116 rows x 7 columns]
```

In [476]:
```
# We had removed some outliers from normal data. Let's see how it looks now.
plt.scatter(P2_normal_PC['PC1'],P2_normal_PC['PC2'],s=1)
```

Out[476]:
```
<matplotlib.collections.PathCollection at 0x7fe3c11fb748>
```



In [178]:
```
# Now, let's try to see whether by plotting only first two PC, can we differentiate
# between the data points.
%matplotlib inline
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(8,8))
ax = plt.axes()
#sample_normal = normal_PC.sample(n=100000)
ax.scatter(P2_normal_PC['PC1'],P2_normal_PC['PC2'], s=1,c ='b');
ax.scatter(P2_attack_PC['PC1'],P2_attack_PC['PC2'], s=1,c ='r');
ax.set_xlabel('PC1')
ax.set_ylabel('PC2')
ax.legend(['Normal','Attack'])
ax.set_title('Visualizing Principal Components of Normal and Attack data for Process
plt.show()
```



3.1 Before training the model, let's first arrange all the actual attacks in a dataframe.

```
In [63]: actual_attacks = attack_data[['Date','Time']]
         actual_attacks['DateTime'] = actual_attacks['Date']+' '+actual_attacks['Time']
```

```
/data/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
In [64]: actual_attacks['datetime'] = list(pd.to_datetime(actual_attacks['DateTime']))
```

```
/data/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```

```
In [65]: actual_attacks['state'] = 1
```

```
/data/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
  """Entry point for launching an IPython kernel.
```

```
In [66]: # Now we'll set the datetime column as index for better accessibility of the datafram
         actual_attacks = actual_attacks.set_index('datetime')

In [67]: # Let's take out the datetime index of actual attacks by slicing the data based on at
         # in the attack details file for all the 15 attacks.
         timestamps_attack = pd.concat([actual_attacks.loc['2017-10-09 19:25:00':'2017-10-09 19
                                        actual_attacks.loc['2017-10-10 10:24:10':'2017-10-10 10:34
                                        actual_attacks.loc['2017-10-10 10:55:00':'2017-10-10 11:24
                                        actual_attacks.loc['2017-10-10 11:30:40':'2017-10-10 11:44
                                        actual_attacks.loc['2017-10-10 13:39:30':'2017-10-10 13:50
                                        actual_attacks.loc['2017-10-10 14:48:17':'2017-10-10 14:59
                                        actual_attacks.loc['2017-10-10 17:40:00':'2017-10-10 17:49
                                        actual_attacks.loc['2017-10-11 10:55:00':'2017-10-11 10:56
                                        actual_attacks.loc['2017-10-11 11:17:54':'2017-10-11 11:31
                                        actual_attacks.loc['2017-10-11 11:36:31':'2017-10-11 11:47
                                        actual_attacks.loc['2017-10-11 11:59:00':'2017-10-11 12:05
                                        actual_attacks.loc['2017-10-11 12:07:30':'2017-10-11 12:10
                                        actual_attacks.loc['2017-10-11 12:16:00':'2017-10-11 12:25
                                        actual_attacks.loc['2017-10-11 15:26:30':'2017-10-11 15:37

In [68]: # Let's change the state of above timestamps to -1 (which mean attack) in the actual_
         actual_attacks.loc[timestamps_attack,'state'] = -1

In [69]: actual_attacks['state'].value_counts()

Out[69]:  1    162853
         -1      9948
         Name: state, dtype: int64
```

3.1.1 Now, it's time to model the PC reduced normal data using different algorithms.

3.2 ---> One Class SVM on Principal Component reconstructed Process 2 normal data

```
In [24]: from sklearn.svm import OneClassSVM

In [511]: # defining the model with its parameters. One thing to note here is varying the para
          # impact on the outcome of model predictions on attack data. Thus it needs to be adj
          OC_SVM = OneClassSVM(gamma='scale',nu=0.0001)

In [514]: OC_SVM.fit(P2_normal_PC.sample(500000))

Out[514]: OneClassSVM(cache_size=200, coef0=0.0, degree=3, gamma='scale', kernel='rbf',
                max_iter=-1, nu=0.0001, random_state=None, shrinking=True, tol=0.001,
                verbose=False)

In [515]: # Let's predict how many data points has been identified as normal in attack data.
          print(np.count_nonzero(OC_SVM.predict(P2_attack_PC)==1),'data points has been identi

157865 data points has been identified as normal


In [457]: # Let's predict how many data points has been identified as normal in attack data.
          print(np.count_nonzero(OC_SVM.predict(P2_attack_PC)==1),'data points has been identi

158073 data points has been identified as normal
```

Time to compute the Confusion Matrix to check the performance of our model.

```
In [70]: from sklearn.metrics import confusion_matrix
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import classification_report

In [516]: predictions = OC_SVM.predict(P2_attack_PC)

In [459]: print(confusion_matrix(actual_attacks['state'],predictions,labels=[1,-1]))

[[152852  10001]
 [  5221   4727]]


In [460]: print('Accuracy Score:', accuracy_score(actual_attacks['state'],predictions))

Accuracy Score: 0.9119102320009722


In [461]: print(classification_report(actual_attacks['state'],predictions))
```

```
              precision    recall  f1-score   support

          -1       0.32      0.48      0.38      9948
           1       0.97      0.94      0.95    162853

   micro avg       0.91      0.91      0.91    172801
   macro avg       0.64      0.71      0.67    172801
weighted avg       0.93      0.91      0.92    172801
```

3.2.1 Time to check ROC and precision-recall curve for the model

```
In [509]: from sklearn.metrics import roc_curve
          from sklearn.metrics import roc_auc_score

In [520]: roc_auc_score(actual_attacks['state'],predictions)

Out[520]: 0.7065078839679017

In [530]: from sklearn.metrics import precision_recall_curve
          from sklearn.metrics import f1_score
          from sklearn.metrics import auc

In [541]: f1_score(actual_attacks['state'],predictions)

Out[541]: 0.9519203786504032

In [538]: auc(precision_recall_curve(actual_attacks['state'],OC_SVM.decision_function(P2_attac

Out[538]: 0.9677410877758246
```

3.3 ---> Isolation Forest on Principal Component reconstructed Process 2 normal data

```
In [42]: from sklearn.ensemble import IsolationForest

In [490]: # Here again, we'll tune the parameters in accordance to provide the best outputs.
          Isolation_Forest = IsolationForest(contamination=0.01, n_estimators=100, behaviour='

In [491]: Isolation_Forest.fit(P2_normal_PC.sample(100000))

Out[491]: IsolationForest(behaviour='new', bootstrap=False, contamination=0.01,
                  max_features=1.0, max_samples='auto', n_estimators=100,
                  n_jobs=None, random_state=None, verbose=0)

In [492]: # Let's predict how many data points has been identified as normal in attack data.
          print(np.count_nonzero(Isolation_Forest.predict(P2_attack_PC)==1),'data points has b

168578 data points has been identified as normal
```

Time to compute the Confusion Matrix to check the performance of our model.

```
In [70]: from sklearn.metrics import confusion_matrix
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import classification_report

In [353]: predictions = Isolation_Forest.predict(P2_attack_PC)

In [177]: print(confusion_matrix(actual_attacks['state'],predictions,labels=[1,-1]))

[[157752   5101]
 [  7335   2613]]


In [172]: print('Accuracy Score:', accuracy_score(actual_attacks['state'],predictions))

Accuracy Score: 0.928032823884121


In [173]: print(classification_report(actual_attacks['state'],predictions))

              precision    recall  f1-score   support

          -1       0.34      0.26      0.30      9948
           1       0.96      0.97      0.96    162853

   micro avg       0.93      0.93      0.93    172801
   macro avg       0.65      0.62      0.63    172801
weighted avg       0.92      0.93      0.92    172801
```

3.4 ---> Elliptic Envelope on Principal Component reconstructed Process 1 normal data

```
In [255]: from sklearn.covariance import EllipticEnvelope

In [502]: # Here again, we'll tune the parameters in accordance to provide the best outputs.
          Elliptic_Envelope = EllipticEnvelope(contamination=0.1)

In [503]: Elliptic_Envelope.fit(P2_normal_PC.sample(100000))

Out[503]: EllipticEnvelope(assume_centered=False, contamination=0.1, random_state=None,
                  store_precision=True, support_fraction=None)

In [504]: # Let's predict how many data points has been identified as normal in attack data.
          print(np.count_nonzero(Elliptic_Envelope.predict(P2_attack_PC)==1),'data points has b

151996 data points has been identified as normal
```

Time to compute the Confusion Matrix to check the performance of our model.

```
In [70]: from sklearn.metrics import confusion_matrix
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import classification_report

In [505]: predictions = Elliptic_Envelope.predict(P2_attack_PC)

In [312]: print(confusion_matrix(actual_attacks['state'],predictions,labels=[1,-1]))

[[144189  18664]
 [  8608   1340]]


In [306]: print('Accuracy Score:', accuracy_score(actual_attacks['state'],predictions))

Accuracy Score: 0.8421768392544025


In [303]: print(classification_report(actual_attacks['state'],predictions))

              precision    recall  f1-score   support

          -1       0.07      0.13      0.09      9948
           1       0.94      0.89      0.91    162853

   micro avg       0.84      0.84      0.84    172801
   macro avg       0.51      0.51      0.50    172801
weighted avg       0.89      0.84      0.87    172801
```