Part 0 : Import necessary packages, cleaning and separation - training and test set

In [302]:

```
#Common
import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

#Used for K Means
from sklearn.cluster import KMeans

#Plotting graph
import matplotlib.pyplot as plt
from pylab import subplot

#Feature selection package
from sklearn.feature_selection import SelectFromModel
from sklearn.ensemble import RandomForestClassifier

#PCA packages
from sklearn.decomposition import PCA

#ICA Packages
from sklearn.decomposition import FastICA

#Randomized project
from sklearn.random_projection import GaussianRandomProje
ction

from sklearn.mixture import GaussianMixture
from sklearn import mixture


from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
```

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


import time
```

In [303]:

```python
counter_strike_data = pd.read_csv("CSGOComplete.csv")
```

In [304]:

```python
#Data Cleaning
counter_strike_data  = counter_strike_data.drop(['Day','Month','Year','Date'] , axis = 1)
x_counter_strike_data = counter_strike_data.drop(labels = ['Result'],axis = 1)
y_counter_strike_data = counter_strike_data[['Result']]
```

In [305]:

```
y_counter_strike_data = y_counter_strike_data['Result']
len_yResult = len(y_counter_strike_data)
for i in range(0, len_yResult):
    if(y_counter_strike_data[i] == 'Win'):
        y_counter_strike_data[i] = '1'
    elif(y_counter_strike_data[i] == 'Lost'):
        y_counter_strike_data[i] = '0'
    elif(y_counter_strike_data[i]== 'Tie'):
        y_counter_strike_data[i] = '2'
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\IPython\c
ore\interactiveshell.py:3296: SettingWithCop
yWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy
  exec(code_obj, self.user_global_ns, self.u
ser_ns)
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy
  import sys
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy
  """
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame
```

```
See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy
  if __name__ == '__main__':
```

In [306]:

```
colNames = x_counter_strike_data.columns
scaler = MinMaxScaler()

for i in range(0, len(colNames)):
    if(colNames[i]!="Map"):
        x_counter_strike_data[colNames[i]] = scaler.fit_t
ransform(x_counter_strike_data[colNames[i]]

.values.reshape(-1,1))
```

In [307]:

```
mapColumn = x_counter_strike_data['Map']
mapColumnLen = len(x_counter_strike_data['Map'])

for i in range(0,mapColumnLen):
    if(mapColumn[i] == 'Mirage'):
        mapColumn[i] = 0
    elif(mapColumn[i] == 'Dust II'):
        mapColumn[i] = 1
    elif(mapColumn[i] == 'Cache'):
        mapColumn[i] = 2
    elif(mapColumn[i] == 'Overpass'):
        mapColumn[i] = 3
    elif(mapColumn[i] == 'Cobblestone'):
        mapColumn[i] = 4
    elif(mapColumn[i] == 'Inferno'):
        mapColumn[i] = 5
    elif(mapColumn[i] == 'Austria'):
        mapColumn[i] = 6
    elif(mapColumn[i] == 'Canals'):
        mapColumn[i] = 7
    elif(mapColumn[i] == 'Nuke'):
        mapColumn[i] = 8
    elif(mapColumn[i] == 'Italy'):
        mapColumn[i] = 9
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy
  # Remove the CWD from sys.path while we lo
ad stuff.
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:12: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame
```

```
See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy
  if sys.path[0] == '':
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:14: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy
  app.launch_new_instance()
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
```

```
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:22: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\ipykernel
_launcher.py:24: SettingWithCopyWarning:
A value is trying to be set on a copy of a s
lice from a DataFrame

See the caveats in the documentation: htt
p://pandas.pydata.org/pandas-docs/stable/ind
exing.html#indexing-view-versus-copy
```

Section 1 : K-Means

1(i) K-Means applied on all columns

In [308]:

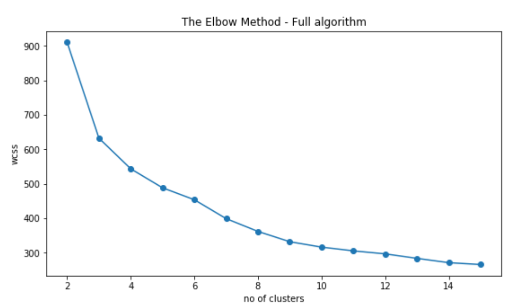
```
x_k_means_1 = x_counter_strike_data
alogrithm_list_k_means = ["full", "elkan"] #i
number_of_clusters_k_means = range(2, 16) #j
Within_Cluster_Sum_of_Squares = []
currentCluster_number = []

step = 0
for i in range (0,len(alogrithm_list_k_means)):
    for j in range (0,len(number_of_clusters_k_means)):
            kmeans = KMeans(algorithm = alogrithm_list_k_
means[i],
                            n_clusters=number_of_clusters
_k_means[j],
                            random_state=10
            ).fit(x_k_means_1)
            Within_Cluster_Sum_of_Squares.append(kmeans.i
nertia_)
            currentCluster_number.append(number_of_cluste
rs_k_means[j])
            print("Step:",(step+1),"/",len(alogrithm_list
_k_means) * len(number_of_clusters_k_means))
            step = step + 1
```

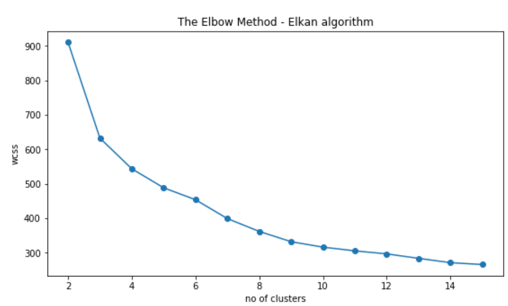
```
Step: 1 / 28
Step: 2 / 28
Step: 3 / 28
Step: 4 / 28
Step: 5 / 28
Step: 6 / 28
Step: 7 / 28
Step: 8 / 28
Step: 9 / 28
Step: 10 / 28
Step: 11 / 28
Step: 12 / 28
Step: 13 / 28
Step: 14 / 28
Step: 15 / 28
Step: 16 / 28
Step: 17 / 28
Step: 18 / 28
Step: 19 / 28
Step: 20 / 28
Step: 21 / 28
Step: 22 / 28
Step: 23 / 28
Step: 24 / 28
Step: 25 / 28
Step: 26 / 28
Step: 27 / 28
Step: 28 / 28
```

1(ii) K-Means applied on all columns - Plot

In [309]:

```
plt.figure(figsize=(20,5))


subplot(1,2,1)
plt.scatter(range(2,16), Within_Cluster_Sum_of_Squares[0:
14])
plt.plot(range(2,16), Within_Cluster_Sum_of_Squares[0:14
])
plt.title('The Elbow Method - Full algorithm')
plt.xlabel('no of clusters')
plt.ylabel('wcss')


subplot(1,2,2)
plt.scatter(range(2,16), Within_Cluster_Sum_of_Squares[14
:28])
plt.plot(range(2,16), Within_Cluster_Sum_of_Squares[14:28
])
plt.title('The Elbow Method - Elkan algorithm')
plt.xlabel('no of clusters')
plt.ylabel('wcss')

plt.show()
```

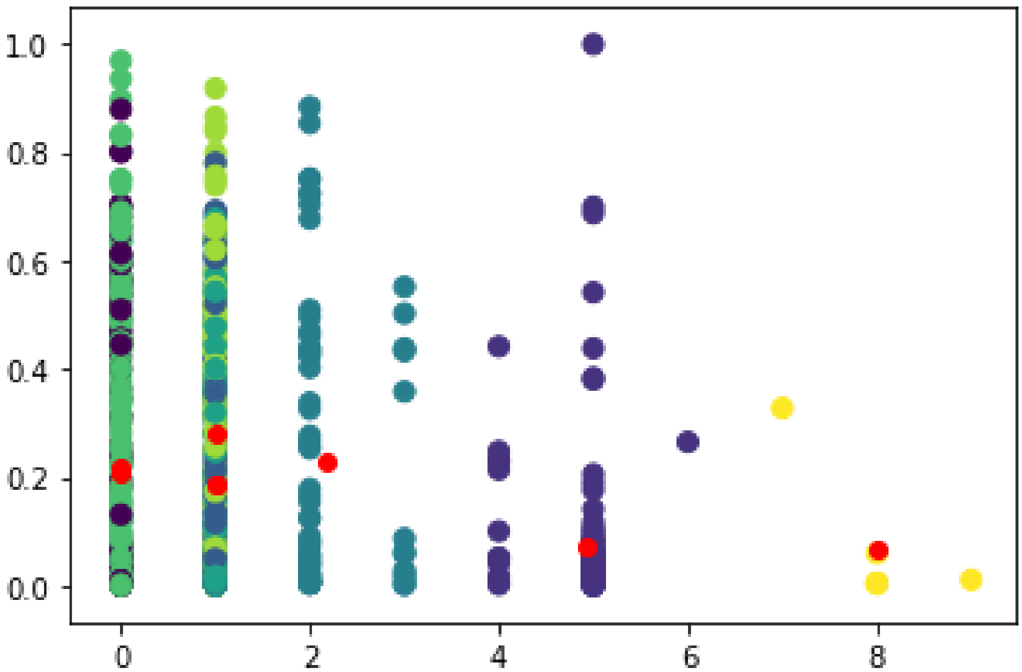




1 (iii) K-Means Plot

In [310]:

```
x_k_means_1 = np.array(x_k_means_1)
kmeans = KMeans(algorithm = "full",
                            n_clusters=8,
                            random_state=10).fit(x_k_mean
s_1)

y_kmeans = kmeans.predict(x_k_means_1)
plt.scatter(x_k_means_1[:, 0], x_k_means_1[:, 1], c=y_kme
ans, s=50, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_
centers_[:,1], color='red')

plt.show()
```



2(i) Feature Selection using Random Forest

In [311]:

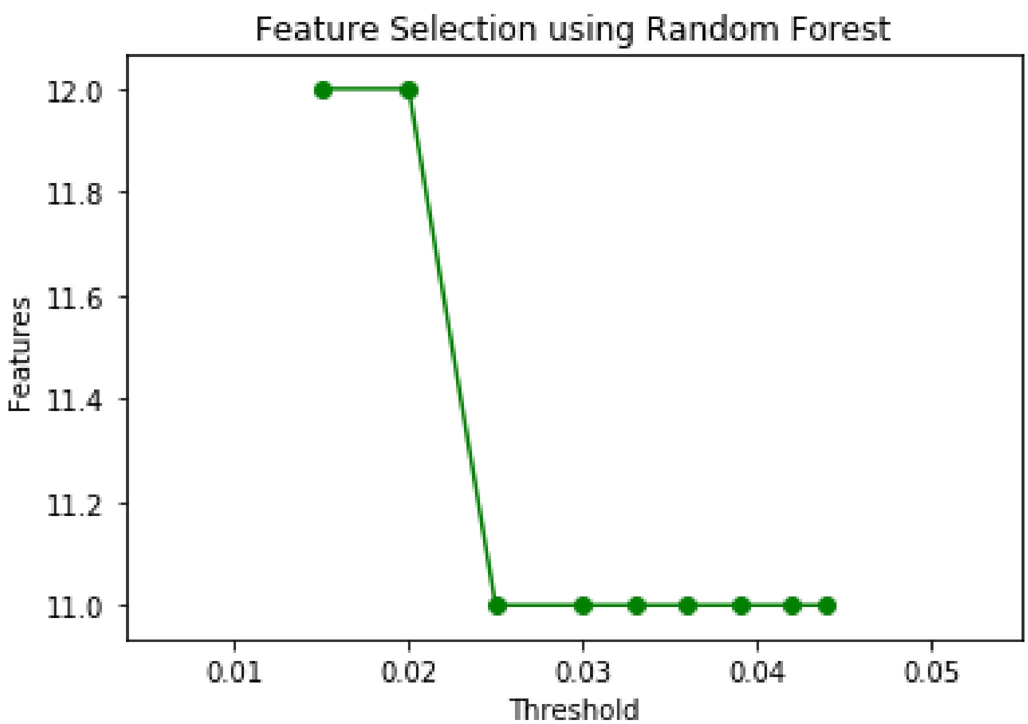
```
x_k_means_2 = x_counter_strike_data
#x_k_means_2 = np.array(x_k_means_2)

thresholdRange = [0.015, 0.02, 0.025, 0.030 ,0.033,0.036,
0.039,0.042,0.044]
final_selectedFeatures = []
final_threshold = []
step = 0
for i in range(0,len(thresholdRange)):
    sel = SelectFromModel(RandomForestClassifier(n_estima
tors = 100,random_state=10), threshold=thresholdRange[i])
    sel.fit(x_k_means_2, y_counter_strike_data)
    sel.get_support()
    selected_feat= x_k_means_2.columns[(sel.get_support
())]

    final_threshold.append(thresholdRange[i])
    final_selectedFeatures.append(len(selected_feat))
    step = step + 1
    print("Step:",step,"/",len(thresholdRange))
```

```
Step: 1 / 9
Step: 2 / 9
Step: 3 / 9
Step: 4 / 9
Step: 5 / 9
Step: 6 / 9
Step: 7 / 9
Step: 8 / 9
Step: 9 / 9
```

2(ii) Feature Selection using Random Forest - Plot

In [312]:

```
plt.scatter(final_threshold, final_selectedFeatures, colo
r = "green")
plt.plot(final_threshold, final_selectedFeatures, color =
"green")
plt.title('Feature Selection using Random Forest')
plt.xlabel('Threshold')
plt.ylabel('Features')

plt.show()
```



3(i) Feature Selection using Random Forest applied on dataset

In [313]:

```
y_k_means_2 = y_counter_strike_data

sel = SelectFromModel(RandomForestClassifier(n_estimators
= 100), threshold=0.025)
sel.fit(x_k_means_2, y_k_means_2)
sel.get_support()
selected_feat= x_k_means_2.columns[(sel.get_support())]
print(selected_feat)
```

```
Index(['Wait Time(s)', ' Match Time(s)', 'Te
am A Rounds', 'Team B Rounds',
       'Ping', 'Kills', 'Assists', 'Deaths',
'Mvp's', 'HS%', 'Points'],
      dtype='object')
```

3(ii) Feature Selection using Random Forest - Getting the features

In [314]:

```
x_k_means_3 = x_counter_strike_data
x_k_means_2_sel = x_k_means_3[['Wait Time(s)', ' Match Ti
me(s)', 'Team A Rounds', 'Team B Rounds',
       'Ping', 'Kills', 'Assists', 'Deaths', "Mvp's", "H
S%", 'Points']]
```

3(iii) Feature Selection using Random Forest - Min max scalar

3(iv) Feature Selection using Random Forest - Elbow cluster selectio

In [315]:

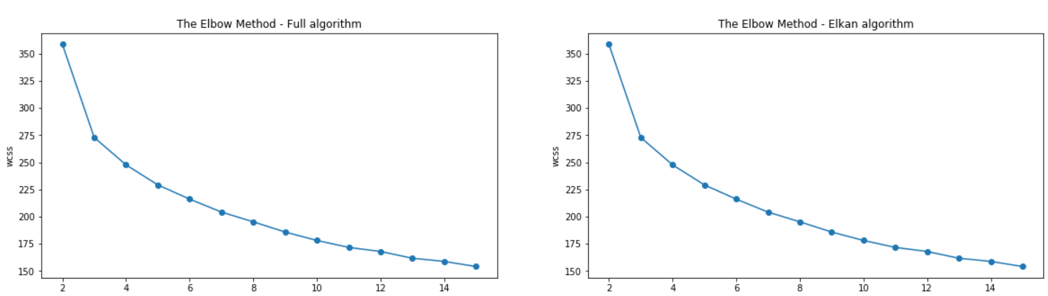
```
alogrithm_list_k_means = ["full", "elkan"] #i
number_of_clusters_k_means = range(2, 16) #j
Within_Cluster_Sum_of_Squares = []
currentCluster_number = []

step = 0
for i in range (0,len(alogrithm_list_k_means)):
    for j in range (0,len(number_of_clusters_k_means)):
            kmeans = KMeans(algorithm = alogrithm_list_k_
means[i],
                            n_clusters=number_of_clusters
_k_means[j],
                            random_state=10
            ).fit(x_k_means_2_sel)
            Within_Cluster_Sum_of_Squares.append(kmeans.i
nertia_)
            currentCluster_number.append(number_of_cluste
rs_k_means[j])
            print("Step:",(step+1),"/",len(alogrithm_list
_k_means) * len(number_of_clusters_k_means))
            step = step + 1
```

```
Step: 1 / 28
Step: 2 / 28
Step: 3 / 28
Step: 4 / 28
Step: 5 / 28
Step: 6 / 28
Step: 7 / 28
Step: 8 / 28
Step: 9 / 28
Step: 10 / 28
Step: 11 / 28
Step: 12 / 28
Step: 13 / 28
Step: 14 / 28
Step: 15 / 28
Step: 16 / 28
Step: 17 / 28
Step: 18 / 28
Step: 19 / 28
Step: 20 / 28
Step: 21 / 28
Step: 22 / 28
Step: 23 / 28
Step: 24 / 28
Step: 25 / 28
Step: 26 / 28
Step: 27 / 28
Step: 28 / 28
```

3(v) Feature Selection using Random Forest - Elbow cluster selection - Plot

In [316]:

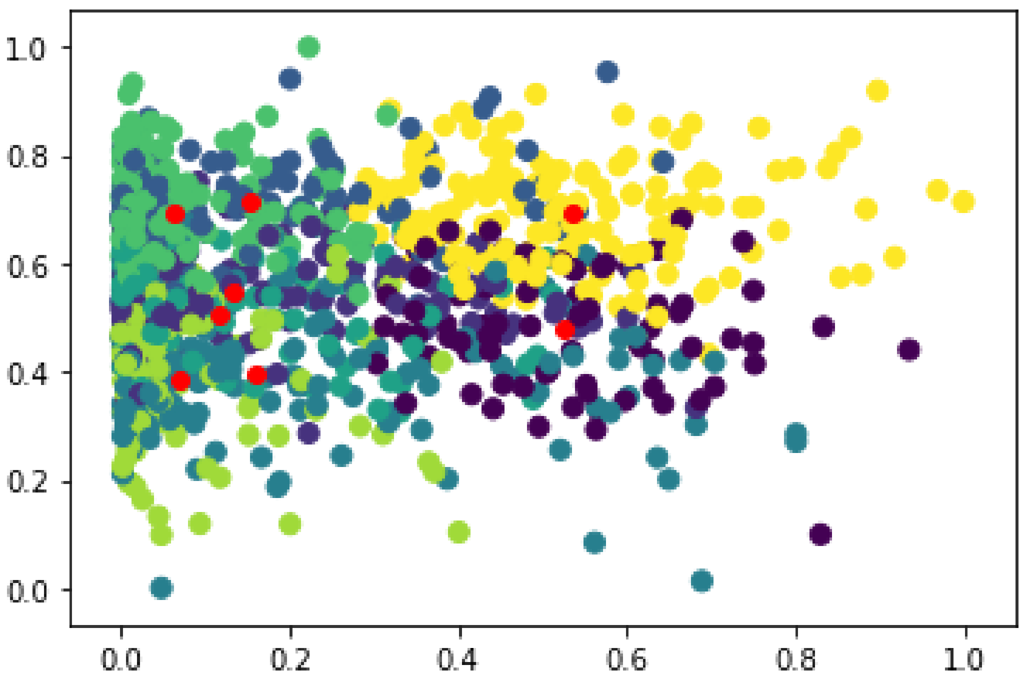
```
plt.figure(figsize=(20,5))


subplot(1,2,1)
plt.scatter(range(2,16), Within_Cluster_Sum_of_Squares[0:14])
plt.plot(range(2,16), Within_Cluster_Sum_of_Squares[0:14])
plt.title('The Elbow Method - Full algorithm')
plt.xlabel('no of clusters')
plt.ylabel('wcss')


subplot(1,2,2)
plt.scatter(range(2,16), Within_Cluster_Sum_of_Squares[14:28])
plt.plot(range(2,16), Within_Cluster_Sum_of_Squares[14:28])
plt.title('The Elbow Method - Elkan algorithm')
plt.xlabel('no of clusters')
plt.ylabel('wcss')


plt.show()
```



3(vi) Feature Selection using Random Forest - K Means clustering

In [317]:

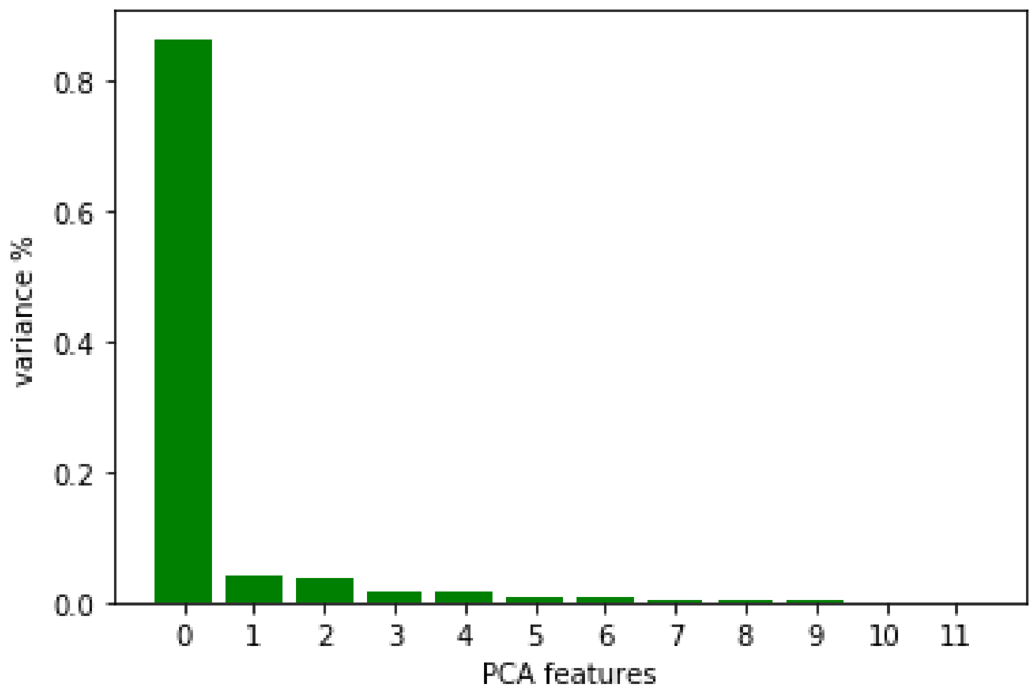
```
x_k_means_2_sel = np.array(x_k_means_2_sel)
kmeans = KMeans(algorithm = "full",
                            n_clusters=8,
                            random_state=10).fit(x_k_mean
s_2_sel)

y_kmeans = kmeans.predict(x_k_means_2_sel)
plt.scatter(x_k_means_2_sel[:, 0], x_k_means_2_sel[:, 1],
c=y_kmeans, s=50, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_
centers_[:,1], color='red')

plt.show()
```



Part 4(i) PCA performed

In [318]:

```
x_k_means_3_sel = x_counter_strike_data

pca = PCA(n_components=12)
principalComponents = pca.fit_transform(x_k_means_3_sel)
# Plot the explained variances
features = range(pca.n_components_)
plt.bar(features, pca.explained_variance_ratio_, color='g
reen')
plt.xlabel('PCA features')
plt.ylabel('variance %')
plt.xticks(features)
# Save components to a DataFrame
PCA_components = pd.DataFrame(principalComponents)
```



Part 4(ii) PCA to build K-Means

In [319]:

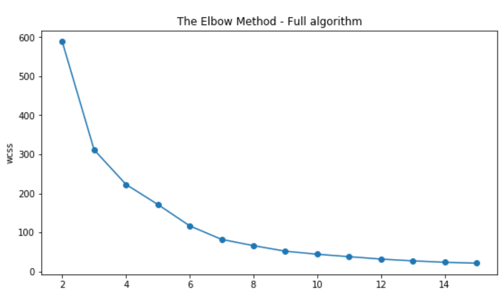
```
PCA_components = pd.DataFrame(principalComponents)

alogrithm_list_k_means = ["full", "elkan"] #i
number_of_clusters_k_means = range(2, 16) #j
Within_Cluster_Sum_of_Squares = []
currentCluster_number = []

step = 0
for i in range (0,len(alogrithm_list_k_means)):
    for j in range (0,len(number_of_clusters_k_means)):
            kmeans = KMeans(algorithm = alogrithm_list_k_
means[i],
                            n_clusters=number_of_clusters
_k_means[j],
                            random_state=10
            ).fit(PCA_components.iloc[:,:2])
            Within_Cluster_Sum_of_Squares.append(kmeans.i
nertia_)
            currentCluster_number.append(number_of_cluste
rs_k_means[j])
            print("Step:",(step+1),"/",len(alogrithm_list
_k_means) * len(number_of_clusters_k_means))
            step = step + 1
```

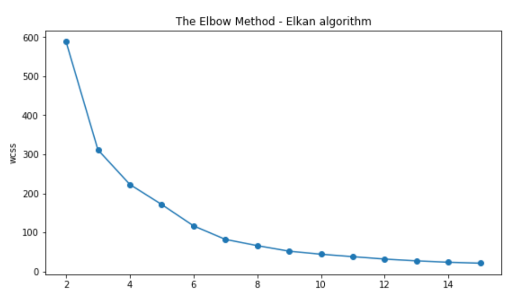
```
Step: 1 / 28
Step: 2 / 28
Step: 3 / 28
Step: 4 / 28
Step: 5 / 28
Step: 6 / 28
Step: 7 / 28
Step: 8 / 28
Step: 9 / 28
Step: 10 / 28
Step: 11 / 28
Step: 12 / 28
Step: 13 / 28
Step: 14 / 28
Step: 15 / 28
Step: 16 / 28
Step: 17 / 28
Step: 18 / 28
Step: 19 / 28
Step: 20 / 28
Step: 21 / 28
Step: 22 / 28
Step: 23 / 28
Step: 24 / 28
Step: 25 / 28
Step: 26 / 28
Step: 27 / 28
Step: 28 / 28
```

Part 4(iii) PCA to build K-Means - Plot

In [320]:

```
plt.figure(figsize=(20,5))


subplot(1,2,1)
plt.scatter(range(2,16), Within_Cluster_Sum_of_Squares[0:14])
plt.plot(range(2,16), Within_Cluster_Sum_of_Squares[0:14])
plt.title('The Elbow Method - Full algorithm')
plt.xlabel('no of clusters')
plt.ylabel('wcss')


subplot(1,2,2)
plt.scatter(range(2,16), Within_Cluster_Sum_of_Squares[14:28])
plt.plot(range(2,16), Within_Cluster_Sum_of_Squares[14:28])
plt.title('The Elbow Method - Elkan algorithm')
plt.xlabel('no of clusters')
plt.ylabel('wcss')

plt.show()
```

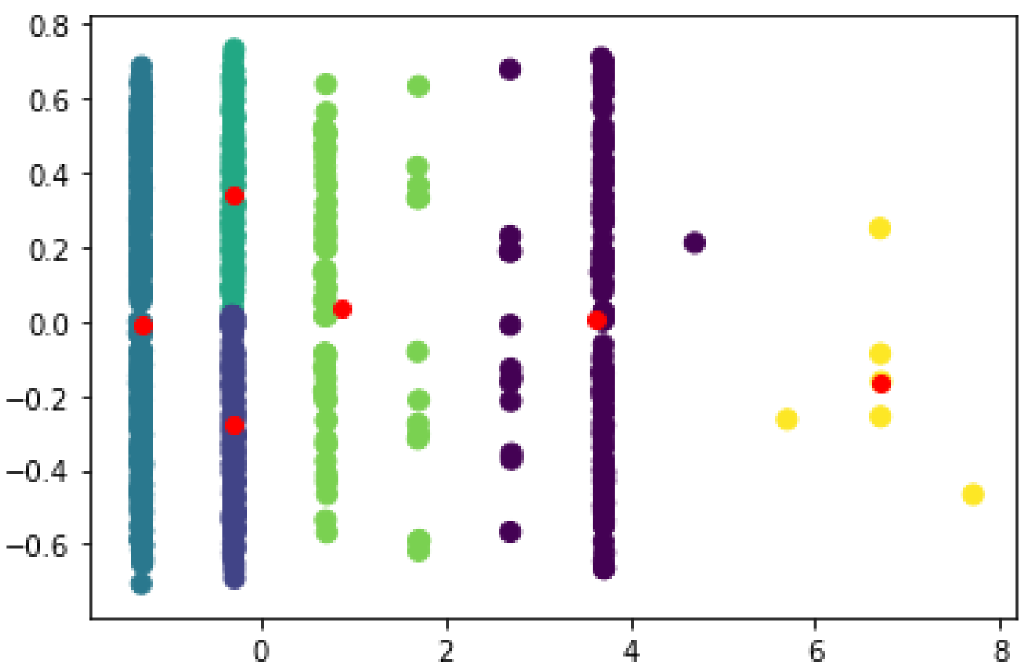




4(iv) Plotting K-means for PCA

In [321]:

```
kmeans = KMeans(algorithm = "full",
                            n_clusters=6,
                            random_state=10).fit(PCA_comp
onents.iloc[:,:2])

pca_map = PCA_components.iloc[:,:2]
y_kmeans = kmeans.predict(pca_map)
plt.scatter(pca_map.iloc[:, 0], pca_map.iloc[:, 1], c=y_k
means, s=50, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_
centers_[:,1], color='red')

plt.show()
```



In [322]:

```
ica = FastICA(n_components=12, random_state=10)

x_k_means_4 = x_counter_strike_data
x_k_means_4_ica = ica.fit_transform(x_k_means_4)
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\d
ecomposition\fastica_.py:119: ConvergenceWar
ning: FastICA did not converge. Consider inc
reasing tolerance or the maximum number of i
terations.
  ConvergenceWarning)
```

Part 5(ii) ICA to build K-Means

In [323]:

```
#ICA_components = pd.DataFrame(x_electricity_ica)



number_of_clusters_k_means = range(2, 16) #j
Within_Cluster_Sum_of_Squares = []
currentCluster_number = []
tol_List_values = []

tol_list = [0.001, 0.01, 0.1]
step = 0
for j in range (0,len(number_of_clusters_k_means)):
    for k in range(0, len(tol_list)):
        ica = FastICA(random_state=10, tol=tol_list[k])
        tol_List_values.append(tol_list[k])
        kmeans = KMeans(algorithm = 'full',
                                    n_clusters=number_of_clusters
_k_means[j],
                                    random_state=10
                               ).fit(x_k_means_4_ica)
        Within_Cluster_Sum_of_Squares.append(kmeans.inert
ia_)
        currentCluster_number.append(number_of_clusters_k
_means[j])
        print("Step:",(step+1),"/",len(tol_list) * len(nu
mber_of_clusters_k_means))
        step = step + 1
```

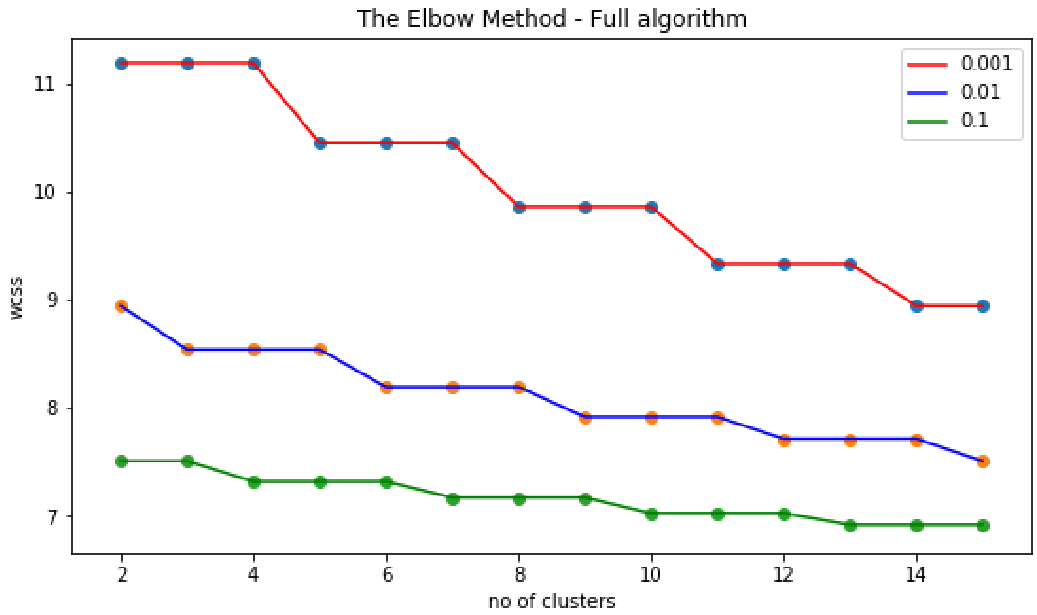
```
Step: 1 / 42
Step: 2 / 42
Step: 3 / 42
Step: 4 / 42
Step: 5 / 42
Step: 6 / 42
Step: 7 / 42
Step: 8 / 42
Step: 9 / 42
Step: 10 / 42
Step: 11 / 42
Step: 12 / 42
Step: 13 / 42
Step: 14 / 42
Step: 15 / 42
Step: 16 / 42
Step: 17 / 42
Step: 18 / 42
Step: 19 / 42
Step: 20 / 42
Step: 21 / 42
Step: 22 / 42
Step: 23 / 42
Step: 24 / 42
Step: 25 / 42
Step: 26 / 42
Step: 27 / 42
Step: 28 / 42
Step: 29 / 42
Step: 30 / 42
Step: 31 / 42
Step: 32 / 42
Step: 33 / 42
Step: 34 / 42
Step: 35 / 42
Step: 36 / 42
Step: 37 / 42
```

```
Step: 38 / 42
Step: 39 / 42
Step: 40 / 42
Step: 41 / 42
Step: 42 / 42
```

Part 5(iii) ICA to build K-Means - Plot

In [324]:

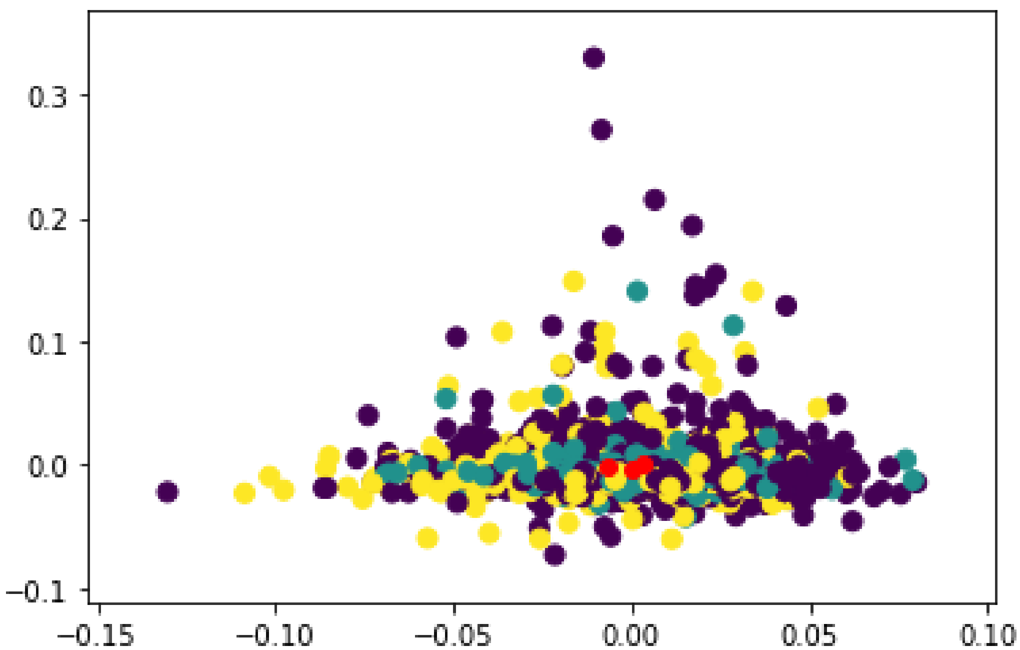
```
plt.figure(figsize=(20,5))

subplot(1,2,1)
plt.scatter(range(2,16), Within_Cluster_Sum_of_Squares[0:
14])
plt.plot(range(2,16), Within_Cluster_Sum_of_Squares[0:14
],color='r',label ='0.001')
plt.scatter(range(2,16), Within_Cluster_Sum_of_Squares[14
:28])
plt.plot(range(2,16), Within_Cluster_Sum_of_Squares[14:28
],color='b',label ='0.01')
plt.scatter(range(2,16), Within_Cluster_Sum_of_Squares[28
:42])
plt.plot(range(2,16), Within_Cluster_Sum_of_Squares[28:42
],color='g',label ='0.1')

plt.title('The Elbow Method - Full algorithm')
plt.xlabel('no of clusters')
plt.ylabel('wcss')
plt.legend(loc='upper right')

plt.show()
```



5(iv) Plotting K-means for ICA

In [325]:

```
kmeans = KMeans(algorithm = "full",
                            n_clusters=3,
                            random_state=10).fit(x_k_mean
s_4_ica)

y_kmeans = kmeans.predict(x_k_means_4_ica)
plt.scatter(x_k_means_4_ica[:, 0], x_k_means_4_ica[:, 1],
c=y_kmeans, s=50, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_
centers_[:,1], color='red')

plt.show()
```



Part 6 - Randomized projects

Part 6(i) Randomized projections performed to build K-Means

In [326]:

```
x_k_means_5 = x_counter_strike_data

transformer = GaussianRandomProjection(random_state=10, n
_components=26)
x_k_means_5_randomized_projects = transformer.fit_transfo
rm(x_k_means_5)

alogrithm_list_k_means = ["full", "elkan"] #i
number_of_clusters_k_means = range(2, 16) #j
Within_Cluster_Sum_of_Squares = []
currentCluster_number = []

step = 0
for i in range (0,len(alogrithm_list_k_means)):
    for j in range (0,len(number_of_clusters_k_means)):
            kmeans = KMeans(algorithm = alogrithm_list_k_
means[i],
                            n_clusters=number_of_clusters
_k_means[j],
                            random_state=10
            ).fit(x_k_means_5_randomized_projects)
            Within_Cluster_Sum_of_Squares.append(kmeans.i
nertia_)
            currentCluster_number.append(number_of_cluste
rs_k_means[j])
            print("Step:",(step+1),"/",len(alogrithm_list
_k_means) * len(number_of_clusters_k_means))
            step = step + 1
```

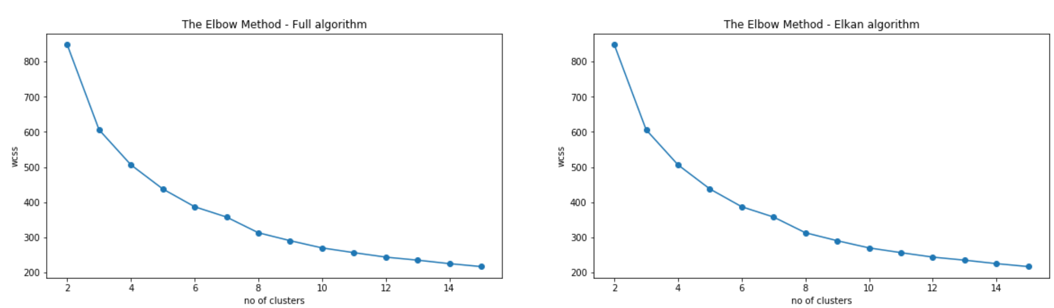
```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\r
andom_projection.py:376: DataDimensionalityW
arning: The number of components is higher t
han the number of features: n_features < n_c
omponents (12 < 26).The dimensionality of th
e problem will not be reduced.
  DataDimensionalityWarning)
```

```
Step: 1 / 28
Step: 2 / 28
Step: 3 / 28
Step: 4 / 28
Step: 5 / 28
Step: 6 / 28
Step: 7 / 28
Step: 8 / 28
Step: 9 / 28
Step: 10 / 28
Step: 11 / 28
Step: 12 / 28
Step: 13 / 28
Step: 14 / 28
Step: 15 / 28
Step: 16 / 28
Step: 17 / 28
Step: 18 / 28
Step: 19 / 28
Step: 20 / 28
Step: 21 / 28
Step: 22 / 28
Step: 23 / 28
Step: 24 / 28
Step: 25 / 28
Step: 26 / 28
Step: 27 / 28
Step: 28 / 28
```

Part 6(ii) Randomized projections performed to build K-Means - Plot

In [327]:

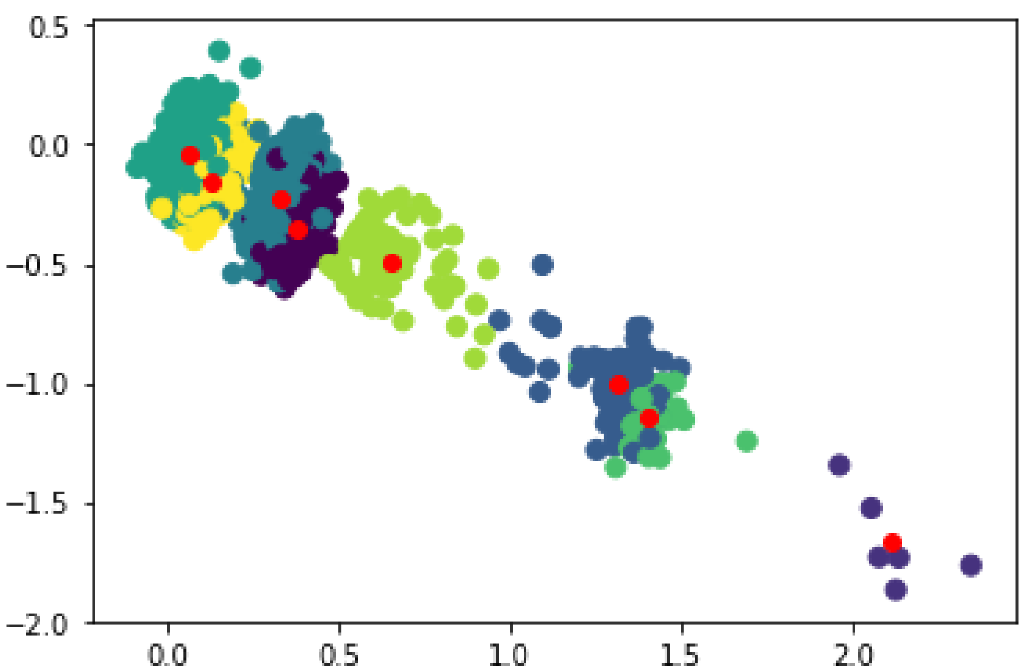
```
plt.figure(figsize=(20,5))

subplot(1,2,1)
plt.scatter(range(2,16), Within_Cluster_Sum_of_Squares[0:
14])
plt.plot(range(2,16), Within_Cluster_Sum_of_Squares[0:14
])
plt.title('The Elbow Method - Full algorithm')
plt.xlabel('no of clusters')
plt.ylabel('wcss')


subplot(1,2,2)
plt.scatter(range(2,16), Within_Cluster_Sum_of_Squares[14
:28])
plt.plot(range(2,16), Within_Cluster_Sum_of_Squares[14:28
])
plt.title('The Elbow Method - Elkan algorithm')
plt.xlabel('no of clusters')
plt.ylabel('wcss')


plt.show()
```



6(iii) Plotting K-means for Randomized projections

In [328]:

```
kmeans = KMeans(algorithm = "full",
                            n_clusters=8,
                            random_state=10).fit(x_k_mean
s_5_randomized_projects)

y_kmeans = kmeans.predict(x_k_means_5_randomized_projects
)
plt.scatter(x_k_means_5_randomized_projects[:, 0], x_k_me
ans_5_randomized_projects[:, 1], c=y_kmeans, s=50, cmap=
'viridis')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_
centers_[:,1], color='red')

plt.show()
```



Section 2 : Expectation Maximization

7(i) Expectation Maximization on the complete dataset

In [329]:

```
x_counter_strike_data_rand_1 = x_counter_strike_data

cov_type = ['tied', 'diag', 'full','spherical']
number_of_components = range(1,13,1)

appendAicValues = []
step = 0

for i in range(0, len(cov_type)):
    for j in range(0, len(number_of_components)):
        g = mixture.GaussianMixture(covariance_type = cov
_type[i],
                                    n_components = number
_of_components[j],
                                    random_state = 10).fi
t(x_counter_strike_data_rand_1)
        appendAicValues.append(g.bic(x_counter_strike_dat
a_rand_1))
        step = step + 1
        print("Step : ", step , "/", len(cov_type) * len(
number_of_components))
```

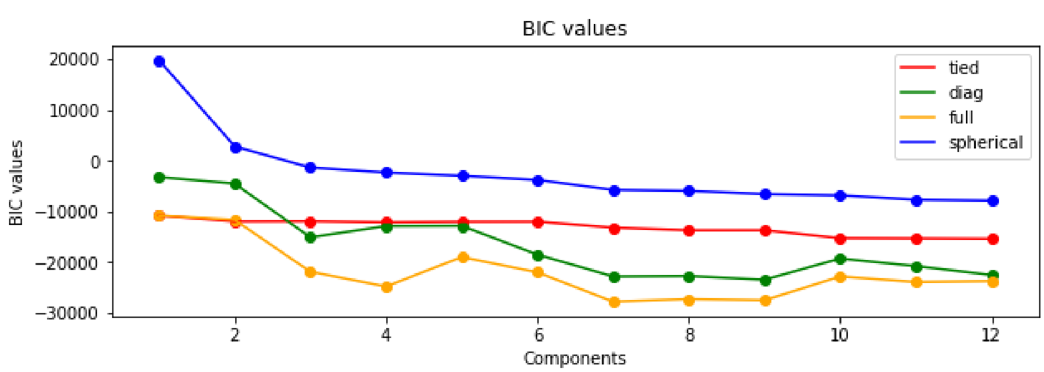
```
Step :  1 / 48
Step :  2 / 48
Step :  3 / 48
Step :  4 / 48
Step :  5 / 48
Step :  6 / 48
Step :  7 / 48
Step :  8 / 48
Step :  9 / 48
Step :  10 / 48
Step :  11 / 48
Step :  12 / 48
Step :  13 / 48
Step :  14 / 48
Step :  15 / 48
Step :  16 / 48
Step :  17 / 48
Step :  18 / 48
Step :  19 / 48
Step :  20 / 48
Step :  21 / 48
Step :  22 / 48
Step :  23 / 48
Step :  24 / 48
Step :  25 / 48
Step :  26 / 48
Step :  27 / 48
Step :  28 / 48
Step :  29 / 48
Step :  30 / 48
Step :  31 / 48
Step :  32 / 48
Step :  33 / 48
Step :  34 / 48
Step :  35 / 48
Step :  36 / 48
Step :  37 / 48
```

```
Step :  38 / 48
Step :  39 / 48
Step :  40 / 48
Step :  41 / 48
Step :  42 / 48
Step :  43 / 48
Step :  44 / 48
Step :  45 / 48
Step :  46 / 48
Step :  47 / 48
Step :  48 / 48
```

7(ii) Expectation Maximization on the complete dataset - Plot

In [330]:

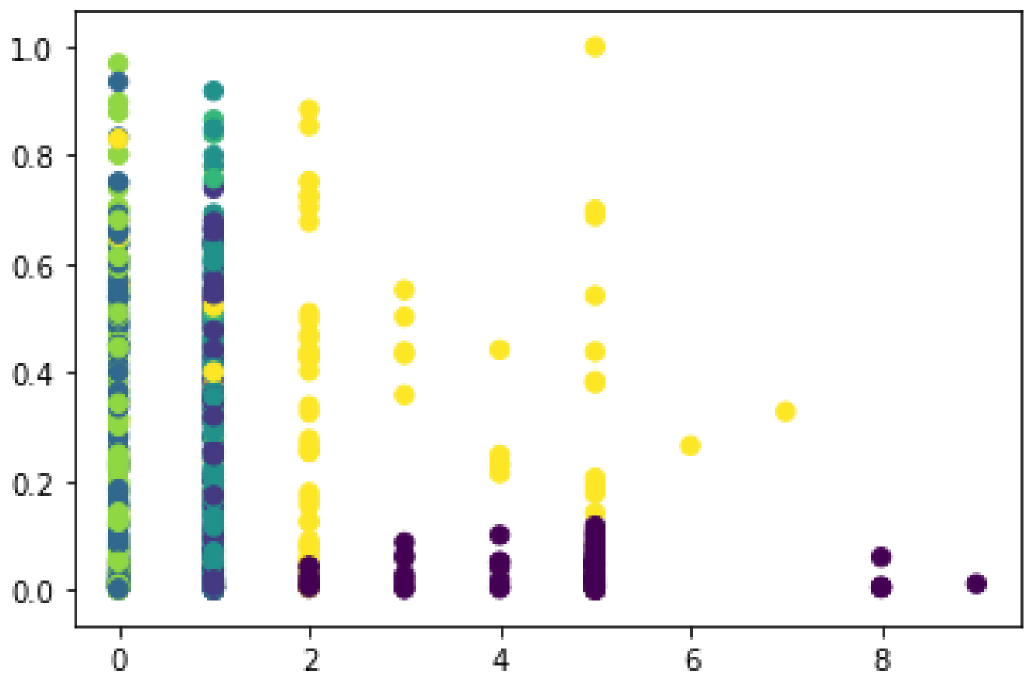
```
steps = np.arange(1,13,1)
plt.figure(figsize=(10,3))

plt.scatter(steps, appendAicValues[0:12], color='red')
plt.plot(steps, appendAicValues[0:12], color='red', label
='tied')

plt.scatter(steps, appendAicValues[12:24], color='green')
plt.plot(steps, appendAicValues[12:24], color='green', la
bel='diag')

plt.scatter(steps, appendAicValues[24:36], color='orange'
)
plt.plot(steps, appendAicValues[24:36], color='orange', l
abel='full')

plt.scatter(steps, appendAicValues[36:48], color='blue')
plt.plot(steps, appendAicValues[36:48], color='blue', lab
el='spherical')


plt.xlabel('Components')
plt.ylabel('BIC values')
plt.legend()
plt.title('BIC values')
plt.show()
```



7(iii) Expectation Maximization on the complete dataset - Separation plot

In [331]:

```
x_counter_strike_data_rand_1 = np.array(x_counter_strike_data_rand_1)
g = mixture.GaussianMixture(covariance_type = 'full',n_components=7).fit(x_counter_strike_data_rand_1)
labels = g.predict(x_counter_strike_data_rand_1)
plt.scatter(x_counter_strike_data_rand_1[:, 0], x_counter_strike_data_rand_1[:, 1],
            c=labels, s=40, cmap='viridis');
```



In [332]:

```
#8 (i) Feature Selection using Random Forest - Based on 2
(ii)

x_counter_strike_data_rand_2 = x_counter_strike_data[['Wa
it Time(s)', ' Match Time(s)', 'Team A Rounds', 'Team B R
ounds',
       'Ping', 'Kills', 'Assists', 'Deaths', "Mvp's", "H
S%", 'Points']]

cov_type = ['tied', 'diag', 'full','spherical']
number_of_components = range(1,26,1)

appendAicValues = []
step = 0

for i in range(0, len(cov_type)):
    for j in range(0, len(number_of_components)):
        g = mixture.GaussianMixture(covariance_type = cov
_type[i],
                                    n_components = number
_of_components[j],
                                    random_state = 10).fi
t(x_counter_strike_data_rand_2)
        appendAicValues.append(g.bic(x_counter_strike_dat
a_rand_2))
        step = step + 1
        print("Step : ", step , "/", len(cov_type) * len(
number_of_components))
        print("length:", len(appendAicValues))
```

```
Step :  1 / 100
length: 1
Step :  2 / 100
length: 2
Step :  3 / 100
length: 3
Step :  4 / 100
length: 4
Step :  5 / 100
length: 5
Step :  6 / 100
length: 6
Step :  7 / 100
length: 7
Step :  8 / 100
length: 8
Step :  9 / 100
length: 9
Step :  10 / 100
length: 10
Step :  11 / 100
length: 11
Step :  12 / 100
length: 12
Step :  13 / 100
length: 13
Step :  14 / 100
length: 14
Step :  15 / 100
length: 15
Step :  16 / 100
length: 16
Step :  17 / 100
length: 17
Step :  18 / 100
length: 18
Step :  19 / 100
```

```
length: 19
Step :  20 / 100
length: 20
Step :  21 / 100
length: 21
Step :  22 / 100
length: 22
Step :  23 / 100
length: 23
Step :  24 / 100
length: 24
Step :  25 / 100
length: 25
Step :  26 / 100
length: 26
Step :  27 / 100
length: 27
Step :  28 / 100
length: 28
Step :  29 / 100
length: 29
Step :  30 / 100
length: 30
Step :  31 / 100
length: 31
Step :  32 / 100
length: 32
Step :  33 / 100
length: 33
Step :  34 / 100
length: 34
Step :  35 / 100
length: 35
Step :  36 / 100
length: 36
Step :  37 / 100
length: 37
```

```
Step :  38 / 100
length: 38
Step :  39 / 100
length: 39
Step :  40 / 100
length: 40
Step :  41 / 100
length: 41
Step :  42 / 100
length: 42
Step :  43 / 100
length: 43
Step :  44 / 100
length: 44
Step :  45 / 100
length: 45
Step :  46 / 100
length: 46
Step :  47 / 100
length: 47
Step :  48 / 100
length: 48
Step :  49 / 100
length: 49
Step :  50 / 100
length: 50
Step :  51 / 100
length: 51
Step :  52 / 100
length: 52
Step :  53 / 100
length: 53
Step :  54 / 100
length: 54
Step :  55 / 100
length: 55
Step :  56 / 100
```

```
length: 56
Step :  57 / 100
length: 57
Step :  58 / 100
length: 58
Step :  59 / 100
length: 59
Step :  60 / 100
length: 60
Step :  61 / 100
length: 61
Step :  62 / 100
length: 62
Step :  63 / 100
length: 63
Step :  64 / 100
length: 64
Step :  65 / 100
length: 65
Step :  66 / 100
length: 66
Step :  67 / 100
length: 67
Step :  68 / 100
length: 68
Step :  69 / 100
length: 69
Step :  70 / 100
length: 70
Step :  71 / 100
length: 71
Step :  72 / 100
length: 72
Step :  73 / 100
length: 73
Step :  74 / 100
length: 74
```

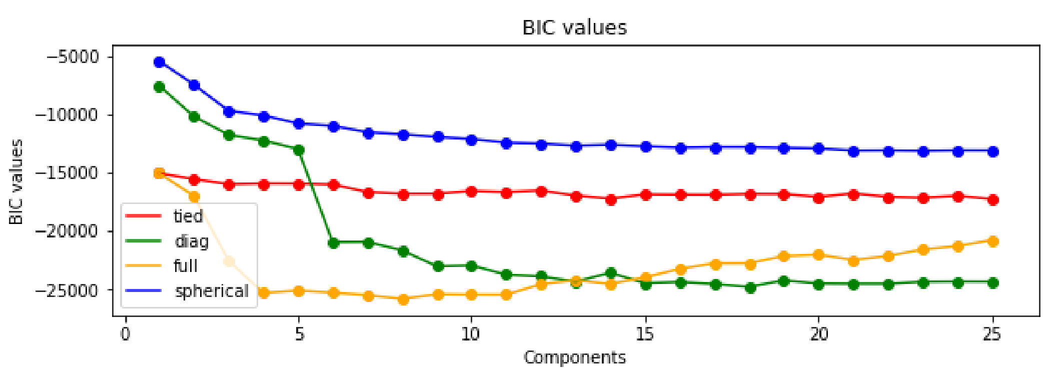
```
Step :  75 / 100
length: 75
Step :  76 / 100
length: 76
Step :  77 / 100
length: 77
Step :  78 / 100
length: 78
Step :  79 / 100
length: 79
Step :  80 / 100
length: 80
Step :  81 / 100
length: 81
Step :  82 / 100
length: 82
Step :  83 / 100
length: 83
Step :  84 / 100
length: 84
Step :  85 / 100
length: 85
Step :  86 / 100
length: 86
Step :  87 / 100
length: 87
Step :  88 / 100
length: 88
Step :  89 / 100
length: 89
Step :  90 / 100
length: 90
Step :  91 / 100
length: 91
Step :  92 / 100
length: 92
Step :  93 / 100
```

```
length: 93
Step :  94 / 100
length: 94
Step :  95 / 100
length: 95
Step :  96 / 100
length: 96
Step :  97 / 100
length: 97
Step :  98 / 100
length: 98
Step :  99 / 100
length: 99
Step :  100 / 100
length: 100
```

8 (ii) Feature selection dataset - Random forest - Plot

In [333]:

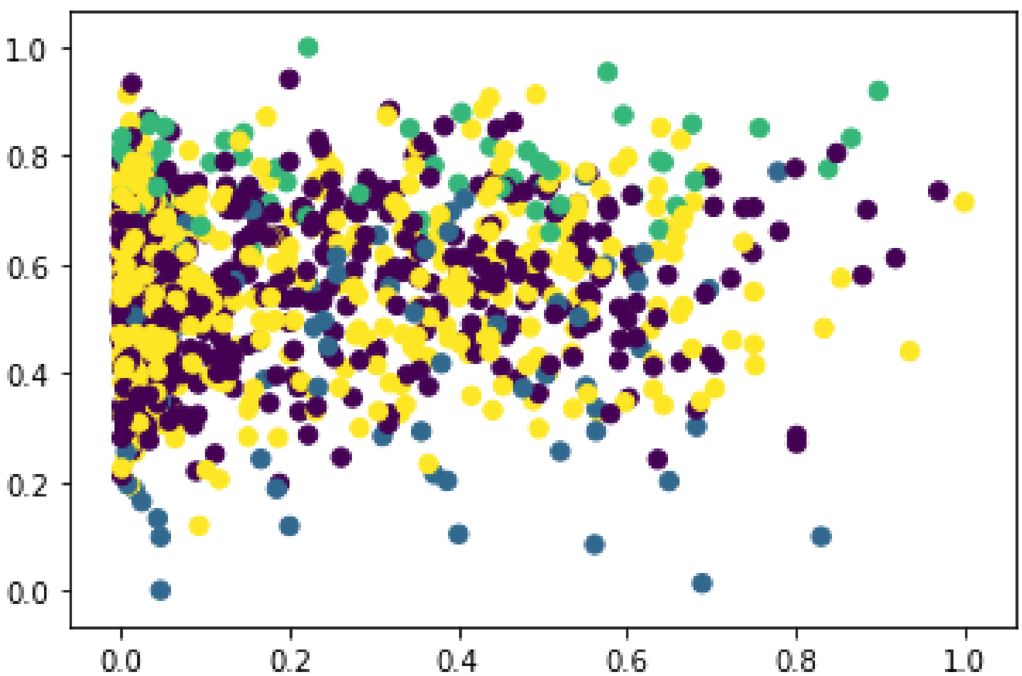
```
steps = np.arange(1,26,1)
plt.figure(figsize=(10,3))

plt.scatter(steps, appendAicValues[0:25], color='red')
plt.plot(steps, appendAicValues[0:25], color='red', label='tied')

plt.scatter(steps, appendAicValues[25:50], color='green')
plt.plot(steps, appendAicValues[25:50], color='green', label='diag')

plt.scatter(steps, appendAicValues[50:75], color='orange')
plt.plot(steps, appendAicValues[50:75], color='orange', label='full')

plt.scatter(steps, appendAicValues[75:100], color='blue')
plt.plot(steps, appendAicValues[75:100], color='blue', label='spherical')


plt.xlabel('Components')
plt.ylabel('BIC values')
plt.legend()
plt.title('BIC values')
plt.show()
```



9(i) Expectation Maximization on the random forest selection set - Separation plot

In [334]:

```
x_counter_strike_data_rand_2 = np.array(x_counter_strike_data_rand_2)

g = mixture.GaussianMixture(covariance_type = 'full',n_components=4).fit(x_counter_strike_data_rand_2)
labels = g.predict(x_counter_strike_data_rand_2)
plt.scatter(x_counter_strike_data_rand_2[:, 0], x_counter_strike_data_rand_2[:, 1], c=labels, s=40, cmap='viridis');
```



10 (i) PCA - Based on the data apply it to Guassian Mixture

In [335]:

```
cov_type = ['tied', 'diag', 'full','spherical']
number_of_components = range(1,27,1)

appendAicValues = []
step = 0

for i in range(0, len(cov_type)):
    for j in range(0, len(number_of_components)):
        g = mixture.GaussianMixture(covariance_type = cov
_type[i],
                                    n_components = number
_of_components[j],
                                    random_state = 10).fi
t(PCA_components.iloc[:,:2])
        appendAicValues.append(g.bic(PCA_components.iloc
[:,:2]))
        step = step + 1
        print("Step : ", step , "/", len(cov_type) * len(
number_of_components))
```

```
Step :  1 / 104
Step :  2 / 104
Step :  3 / 104
Step :  4 / 104
Step :  5 / 104
Step :  6 / 104
Step :  7 / 104
Step :  8 / 104
Step :  9 / 104
Step :  10 / 104
Step :  11 / 104
Step :  12 / 104
Step :  13 / 104
Step :  14 / 104
Step :  15 / 104
Step :  16 / 104
Step :  17 / 104
Step :  18 / 104
Step :  19 / 104
Step :  20 / 104
Step :  21 / 104
Step :  22 / 104
Step :  23 / 104
Step :  24 / 104
Step :  25 / 104
Step :  26 / 104
Step :  27 / 104
Step :  28 / 104
Step :  29 / 104
Step :  30 / 104
Step :  31 / 104
Step :  32 / 104
Step :  33 / 104
Step :  34 / 104
Step :  35 / 104
Step :  36 / 104
Step :  37 / 104
```

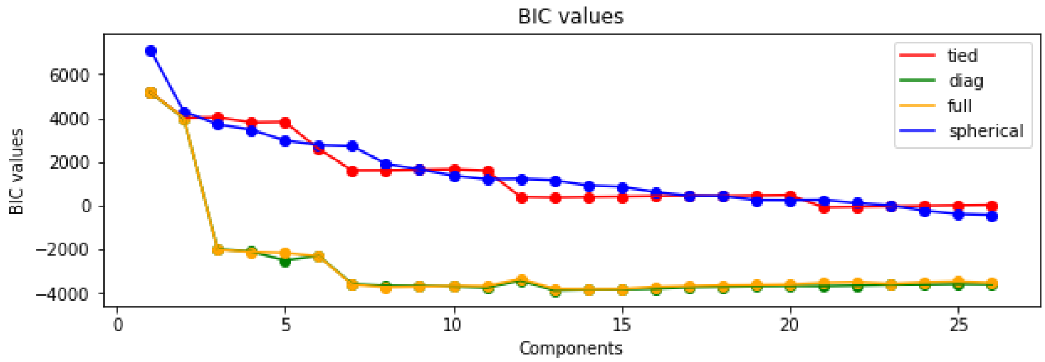
```
Step :  38 / 104
Step :  39 / 104
Step :  40 / 104
Step :  41 / 104
Step :  42 / 104
Step :  43 / 104
Step :  44 / 104
Step :  45 / 104
Step :  46 / 104
Step :  47 / 104
Step :  48 / 104
Step :  49 / 104
Step :  50 / 104
Step :  51 / 104
Step :  52 / 104
Step :  53 / 104
Step :  54 / 104
Step :  55 / 104
Step :  56 / 104
Step :  57 / 104
Step :  58 / 104
Step :  59 / 104
Step :  60 / 104
Step :  61 / 104
Step :  62 / 104
Step :  63 / 104
Step :  64 / 104
Step :  65 / 104
Step :  66 / 104
Step :  67 / 104
Step :  68 / 104
Step :  69 / 104
Step :  70 / 104
Step :  71 / 104
Step :  72 / 104
Step :  73 / 104
Step :  74 / 104
```

```
Step :  75 / 104
Step :  76 / 104
Step :  77 / 104
Step :  78 / 104
Step :  79 / 104
Step :  80 / 104
Step :  81 / 104
Step :  82 / 104
Step :  83 / 104
Step :  84 / 104
Step :  85 / 104
Step :  86 / 104
Step :  87 / 104
Step :  88 / 104
Step :  89 / 104
Step :  90 / 104
Step :  91 / 104
Step :  92 / 104
Step :  93 / 104
Step :  94 / 104
Step :  95 / 104
Step :  96 / 104
Step :  97 / 104
Step :  98 / 104
Step :  99 / 104
Step :  100 / 104
Step :  101 / 104
Step :  102 / 104
Step :  103 / 104
Step :  104 / 104
```

10 (ii) PCA - Based on the data apply it to Guassian Mixture - Plot

In [336]:

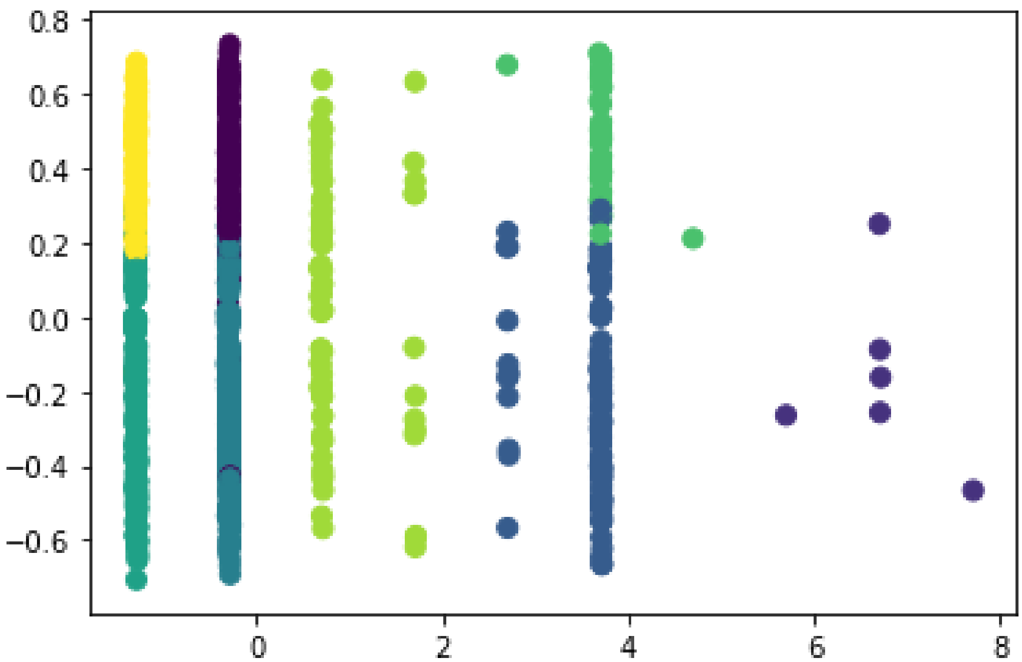
```
steps = np.arange(1,27,1)
plt.figure(figsize=(10,3))

plt.scatter(steps, appendAicValues[0:26], color='red')
plt.plot(steps, appendAicValues[0:26], color='red', label
='tied')

plt.scatter(steps, appendAicValues[26:52], color='green')
plt.plot(steps, appendAicValues[26:52], color='green', la
bel='diag')

plt.scatter(steps, appendAicValues[52:78], color='orange'
)
plt.plot(steps, appendAicValues[52:78], color='orange', l
abel='full')

plt.scatter(steps, appendAicValues[78:104], color='blue')
plt.plot(steps, appendAicValues[78:104], color='blue', la
bel='spherical')


plt.xlabel('Components')
plt.ylabel('BIC values')
plt.legend()
plt.title('BIC values')
plt.show()
```



10 (iii) Expectation Maximization on PCA - Separation plot

In [337]:

```
g = mixture.GaussianMixture(covariance_type = 'full',n_co
mponents=8, random_state=10).fit(PCA_components.iloc[:,:2
])
pca_map = PCA_components.iloc[:,:2]
labels = g.predict(pca_map)
plt.scatter(pca_map.iloc[:, 0], pca_map.iloc[:, 1], c=y_k
means, s=50, cmap='viridis')
plt.show()
```



11 (i) ICA - Based on the data apply it to Guassian Mixture

In [338]:

```
cov_type = ['tied', 'diag', 'full','spherical']
number_of_components = range(1,13,1)

appendAicValues = []
step = 0

for i in range(0, len(cov_type)):
    for j in range(0, len(number_of_components)):
        g = mixture.GaussianMixture(covariance_type = cov
_type[i],
                                    n_components = number
_of_components[j],
                                    random_state = 10).fi
t(x_k_means_4_ica)
        appendAicValues.append(g.bic(x_k_means_4_ica))
        step = step + 1
        print("Step : ", step , "/", len(cov_type) * len(
number_of_components))
```

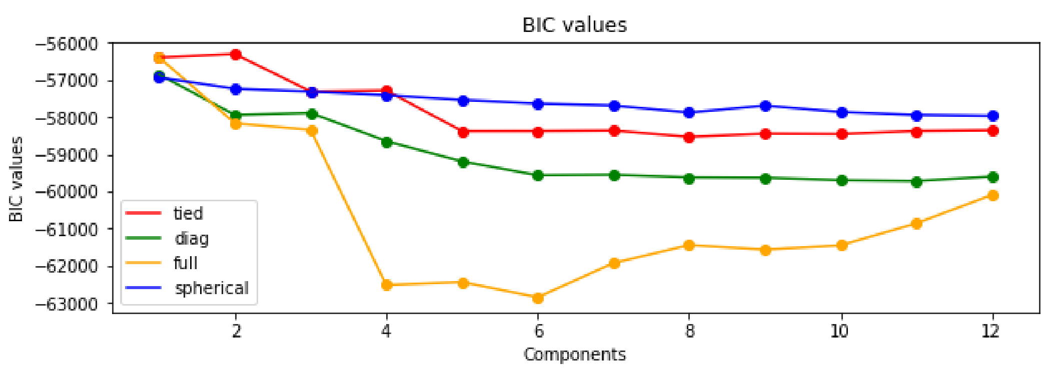
```
Step :  1 / 48
Step :  2 / 48
Step :  3 / 48
Step :  4 / 48
Step :  5 / 48
Step :  6 / 48
Step :  7 / 48
Step :  8 / 48
Step :  9 / 48
Step :  10 / 48
Step :  11 / 48
Step :  12 / 48
Step :  13 / 48
Step :  14 / 48
Step :  15 / 48
Step :  16 / 48
Step :  17 / 48
Step :  18 / 48
Step :  19 / 48
Step :  20 / 48
Step :  21 / 48
Step :  22 / 48
Step :  23 / 48
Step :  24 / 48
Step :  25 / 48
Step :  26 / 48
Step :  27 / 48
Step :  28 / 48
Step :  29 / 48
Step :  30 / 48
Step :  31 / 48
Step :  32 / 48
Step :  33 / 48
Step :  34 / 48
Step :  35 / 48
Step :  36 / 48
Step :  37 / 48
```

```
Step :  38 / 48
Step :  39 / 48
Step :  40 / 48
Step :  41 / 48
Step :  42 / 48
Step :  43 / 48
Step :  44 / 48
Step :  45 / 48
Step :  46 / 48
Step :  47 / 48
Step :  48 / 48
```

11 (ii) ICA - Based on the data apply it to Guassian Mixture - Plot

In [339]:

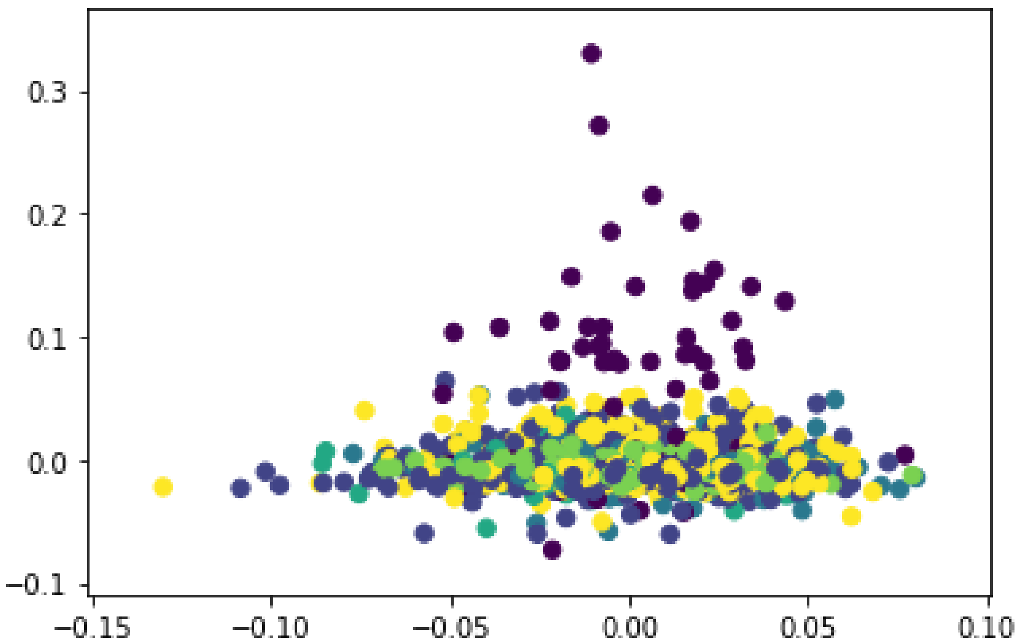
```
steps = np.arange(1,13,1)
plt.figure(figsize=(10,3))

plt.scatter(steps, appendAicValues[0:12], color='red')
plt.plot(steps, appendAicValues[0:12], color='red', label
='tied')

plt.scatter(steps, appendAicValues[12:24], color='green')
plt.plot(steps, appendAicValues[12:24], color='green', la
bel='diag')

plt.scatter(steps, appendAicValues[24:36], color='orange'
)
plt.plot(steps, appendAicValues[24:36], color='orange', l
abel='full')

plt.scatter(steps, appendAicValues[36:48], color='blue')
plt.plot(steps, appendAicValues[36:48], color='blue', lab
el='spherical')



plt.xlabel('Components')
plt.ylabel('BIC values')
plt.legend()
plt.title('BIC values')
plt.show()
```



11 (iii) Expectation Maximization on ICA - Separation plot

In [340]:

```
g = mixture.GaussianMixture(covariance_type = 'full',n_co
mponents=6, random_state = 10).fit(x_k_means_4_ica)
labels = g.predict(x_k_means_4_ica)
plt.scatter(x_k_means_4_ica[:, 0], x_k_means_4_ica[:, 1],
c=labels, s=40, cmap='viridis');
```



12 (i) Randomized projection - Based on the data apply it to Expectation Max

In [341]:

```
cov_type = ['tied', 'diag', 'full','spherical']
number_of_components = range(1,13,1)

appendAicValues = []
step = 0

for i in range(0, len(cov_type)):
    for j in range(0, len(number_of_components)):
        g = mixture.GaussianMixture(covariance_type = cov
_type[i],
                                    n_components = number
_of_components[j],
                                    random_state = 10).fi
t(x_k_means_5_randomized_projects)
        appendAicValues.append(g.bic(x_k_means_5_randomiz
ed_projects))
        step = step + 1
        print("Step : ", step , "/", len(cov_type) * len(
number_of_components))
```

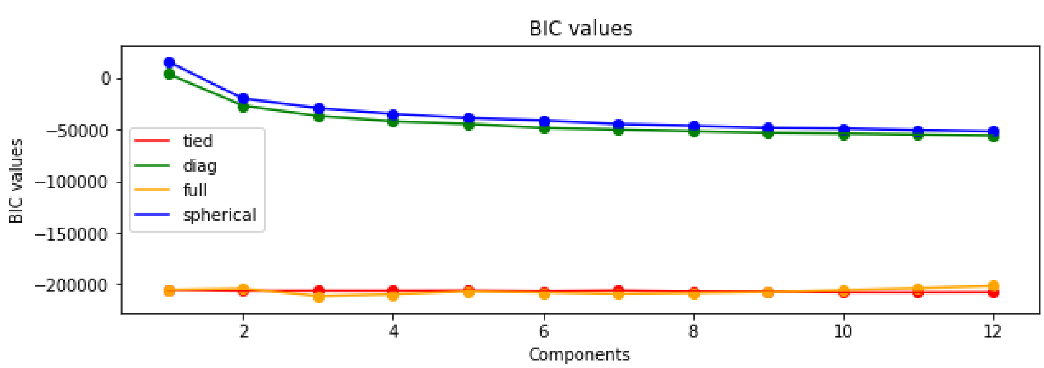
```
Step :  1 / 48
Step :  2 / 48
Step :  3 / 48
Step :  4 / 48
Step :  5 / 48
Step :  6 / 48
Step :  7 / 48
Step :  8 / 48
Step :  9 / 48
Step :  10 / 48
Step :  11 / 48
Step :  12 / 48
Step :  13 / 48
Step :  14 / 48
Step :  15 / 48
Step :  16 / 48
Step :  17 / 48
Step :  18 / 48
Step :  19 / 48
Step :  20 / 48
Step :  21 / 48
Step :  22 / 48
Step :  23 / 48
Step :  24 / 48
Step :  25 / 48
Step :  26 / 48
Step :  27 / 48
Step :  28 / 48
Step :  29 / 48
Step :  30 / 48
Step :  31 / 48
Step :  32 / 48
Step :  33 / 48
Step :  34 / 48
Step :  35 / 48
Step :  36 / 48
Step :  37 / 48
```

```
Step :  38 / 48
Step :  39 / 48
Step :  40 / 48
Step :  41 / 48
Step :  42 / 48
Step :  43 / 48
Step :  44 / 48
Step :  45 / 48
Step :  46 / 48
Step :  47 / 48
Step :  48 / 48
```

12 (ii) Randomized projection - Based on the data apply it to Expectation Max - Plot

In [342]:

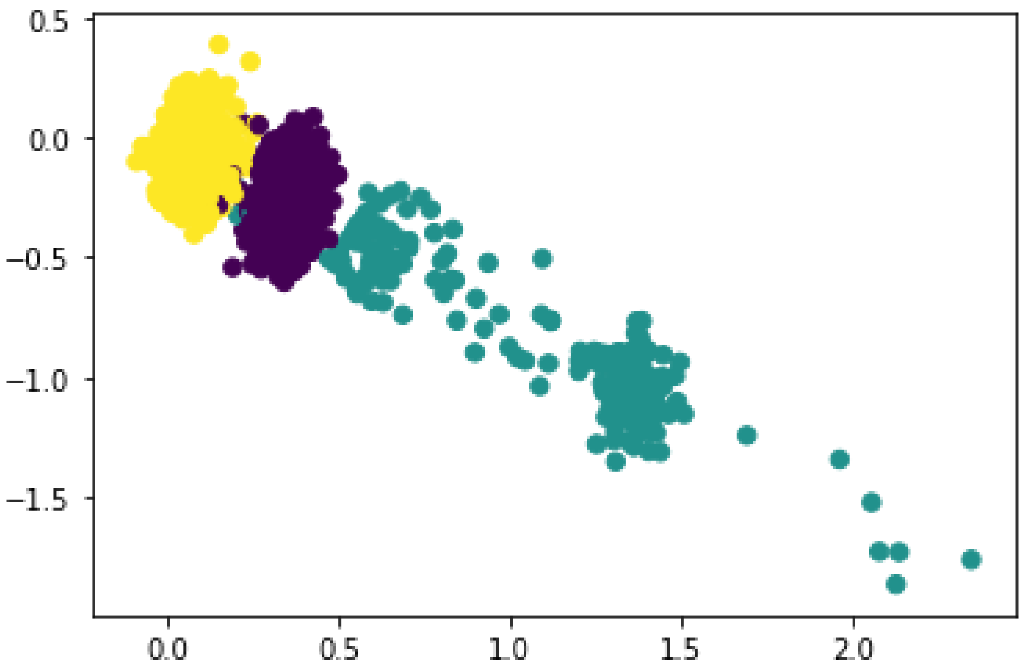
```
steps = np.arange(1,13,1)
plt.figure(figsize=(10,3))

plt.scatter(steps, appendAicValues[0:12], color='red')
plt.plot(steps, appendAicValues[0:12], color='red', label
='tied')

plt.scatter(steps, appendAicValues[12:24], color='green')
plt.plot(steps, appendAicValues[12:24], color='green', la
bel='diag')

plt.scatter(steps, appendAicValues[24:36], color='orange'
)
plt.plot(steps, appendAicValues[24:36], color='orange', l
abel='full')

plt.scatter(steps, appendAicValues[36:48], color='blue')
plt.plot(steps, appendAicValues[36:48], color='blue', lab
el='spherical')



plt.xlabel('Components')
plt.ylabel('BIC values')
plt.legend()
plt.title('BIC values')
plt.show()
```



12 (iii) Randomized projection - Separation plot

In [343]:

```
g = mixture.GaussianMixture(covariance_type = 'full',n_components=3).fit(x_k_means_5_randomized_projects)
labels = g.predict(x_k_means_5_randomized_projects)
plt.scatter(x_k_means_5_randomized_projects[:, 0], x_k_means_5_randomized_projects[:, 1], c=labels, s=40, cmap='viridis');
```



Section 3 : Neural NETS

13 Neural networks : Based on Feature selection

Scaling and test-train split

In [344]:

```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(x_cou
nter_strike_data, y_counter_strike_data, test_size=0.3,
                                                    random
_state=1)
```

13(i) Neural networks on Entire data

In [345]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_size
s=(4,3), random_state=1,max_iter = 3000)
clf.fit(X_Train, Y_Train)
predicted_classes = clf.predict(X_Test)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 5.771490812301636 seconds ---
```

13(ii) Neural networks on Entire data - parameters

In [346]:

```
cm = confusion_matrix(Y_Test, predicted_classes)
print(cm)
print("Accuracy", (cm[1][1] + cm[0][0]) / np.sum(cm) )
print("Sensitivity", cm[1][1] / (cm[1][1] + cm[1][0] ))
print("Specificity", cm[0][0] / (cm[0][0] + cm[0][1] ))
print("Precision", cm[1][1] / (cm[1][1] + cm[0][1] ))
```

```
[[129  36  12]
 [ 20 113   6]
 [  7   0  17]]
Accuracy 0.711764705882353
Sensitivity 0.849624060150376
Specificity 0.7818181818181819
Precision 0.7583892617449665
```

14 Neural networks : Based on PCA

14 (i) Neural networks : Based on PCA Build - Experiment

In [347]:

```
scaler = MinMaxScaler()
x_pca = scaler.fit_transform(x_counter_strike_data)
principalComponents = pca.fit_transform(x_pca)
PCA_components = pd.DataFrame(principalComponents)
```

In [348]:

```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(PCA_components.iloc[:,:2], y_counter_strike_data, test_size=0.3,
                                                                       random_state=1)
```

14(ii) Neural networks - PCA - Experiment

In [349]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_sizes=(4,3), random_state=1,max_iter = 3000)
clf.fit(X_Train, Y_Train)
predicted_classes = clf.predict(X_Test)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 0.7881829738616943 seconds ---
```

14(iii) Neural networks - PCA - Experiment - Parameters

In [350]:

```
cm = confusion_matrix(Y_Test, predicted_classes)
print(cm)
print("Accuracy", (cm[1][1] + cm[0][0]) / np.sum(cm) )
print("Sensitivity", cm[1][1] / (cm[1][1] + cm[1][0] ))
print("Specificity", cm[0][0] / (cm[0][0] + cm[0][1] ))
print("Precision", cm[1][1] / (cm[1][1] + cm[0][1] ))
```

```
[[114  61   2]
 [ 89  45   5]
 [  3  16   5]]
Accuracy 0.4676470588235294
Sensitivity 0.3358208955223881
Specificity 0.6514285714285715
Precision 0.42452830188679247
```

15 Neural networks : Based on ICA

15 (i) Neural networks : Based on ICA Build

In [351]:

```
scaler = MinMaxScaler()
x_ica = scaler.fit_transform(x_counter_strike_data)

ica = FastICA(n_components=12, random_state=10)
x_ica = ica.fit_transform(x_ica)
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\d
ecomposition\fastica_.py:119: ConvergenceWar
ning: FastICA did not converge. Consider inc
reasing tolerance or the maximum number of i
terations.
  ConvergenceWarning)
```

In [352]:

```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(x_ica
, y_counter_strike_data, test_size=0.3,
                                                    random
_state=1)
```

15 (ii) Neural networks : Based on ICA Build - Experiment

In [353]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_size
s=(4,3), random_state=1,max_iter = 3000)
clf.fit(X_Train, Y_Train)
predicted_classes = clf.predict(X_Test)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 3.494811773300171 seconds ---
```

15(iii) Neural networks - ICA - Experiment - Parameters

In [354]:

```
cm = confusion_matrix(Y_Test, predicted_classes)
print(cm)
print("Accuracy", (cm[1][1] + cm[0][0]) / np.sum(cm) )
print("Sensitivity", cm[1][1] / (cm[1][1] + cm[1][0] ))
print("Specificity", cm[0][0] / (cm[0][0] + cm[0][1] ))
print("Precision", cm[1][1] / (cm[1][1] + cm[0][1] ))
```

```
[[119  40  18]
 [ 23 110   6]
 [  2   3  19]]
Accuracy 0.6735294117647059
Sensitivity 0.8270676691729323
Specificity 0.7484276729559748
Precision 0.7333333333333333
```

16 Neural networks : Based on Randomized projections

16(i) Take randomized projections : Get and split data

In [355]:

```
scaler = MinMaxScaler()
x_rand = scaler.fit_transform(x_counter_strike_data)

transformer = GaussianRandomProjection(random_state=10, n
_components=12)
x_rand = transformer.fit_transform(x_rand)

X_Train, X_Test, Y_Train, Y_Test = train_test_split(x_ran
d, y_counter_strike_data, test_size=0.3,
                                                    random
_state=1)
```

16 (ii) Neural networks : Based on Randomized projects Build - Experiment

In [356]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_size
s=(4,3), random_state=1,max_iter = 3000)
clf.fit(X_Train, Y_Train)
predicted_classes = clf.predict(X_Test)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 3.4338009357452393 seconds ---
```

16(iii) Neural networks - RAndomized projections - Experiment - Parameters

In [357]:

```
cm = confusion_matrix(Y_Test, predicted_classes)
print(cm)
print("Accuracy", (cm[1][1] + cm[0][0]) / np.sum(cm) )
print("Sensitivity", cm[1][1] / (cm[1][1] + cm[1][0] ))
print("Specificity", cm[0][0] / (cm[0][0] + cm[0][1] ))
print("Precision", cm[1][1] / (cm[1][1] + cm[0][1] ))
```

```
[[132  31  14]
 [ 30 104   5]
 [  8   1  15]]
Accuracy 0.6941176470588235
Sensitivity 0.7761194029850746
Specificity 0.8098159509202454
Precision 0.7703703703703704
```

17 Neural networks : Based on Feature Selection

17(i) Neural networks : Feature Selection

In [358]:

```
x_feature_select = x_counter_strike_data[['Wait Time(s)',
' Match Time(s)', 'Team A Rounds', 'Team B Rounds',
       'Ping', 'Kills', 'Assists', 'Deaths', "Mvp's", "H
S%", 'Points']]

scaler = MinMaxScaler()
x_feature_select = scaler.fit_transform(x_feature_select)

X_Train, X_Test, Y_Train, Y_Test = train_test_split(x_fea
ture_select, y_counter_strike_data, test_size=0.3,
                                                    random
_state=1)
```

17(ii) Neural networks : Run neural networks based on feature selection

In [359]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_size
s=(4,3), random_state=1,max_iter = 3000)
clf.fit(X_Train, Y_Train)
predicted_classes = clf.predict(X_Test)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 3.6976749897003174 seconds ---
```

17(iii) Neural networks : Run neural networks based on neural networks - Parameters

In [360]:

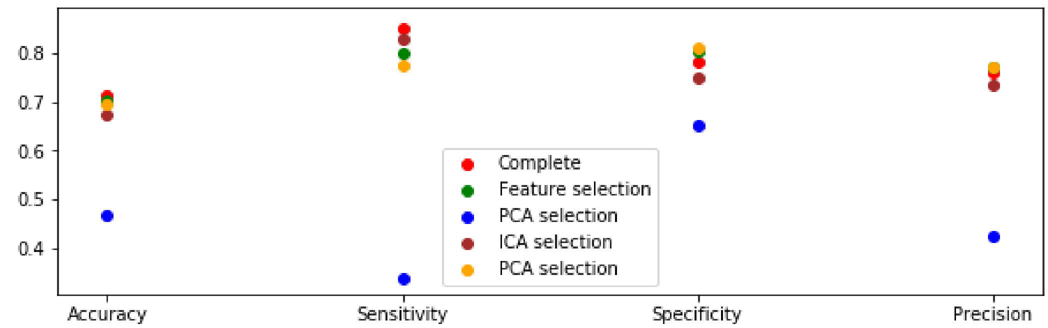
```
cm = confusion_matrix(Y_Test, predicted_classes)
print(cm)
print("Accuracy", (cm[1][1] + cm[0][0]) / np.sum(cm) )
print("Sensitivity", cm[1][1] / (cm[1][1] + cm[1][0] ))
print("Specificity", cm[0][0] / (cm[0][0] + cm[0][1] ))
print("Precision", cm[1][1] / (cm[1][1] + cm[0][1] ))
```

```
[[131  32  14]
 [ 27 108   4]
 [  8   2  14]]
Accuracy 0.7029411764705882
Sensitivity 0.8
Specificity 0.803680981595092
Precision 0.7714285714285715
```

18 Plot confusion matrix plot

In [361]:

```
complete_values_plot =[0.7118,0.8496,0.7818,0.7584]
feature_Selection_plot = [0.7029, 0.8, 0.8037, 0.7714]
pca_selection_plot = [0.4676, 0.3358, 0.6514, 0.4245]
ica_selection_plot = [0.6735, 0.8271, 0.7484, 0.7333]
rand_selection_plot = [0.6941,0.7761,0.8098,0.7704]

steps = ['Accuracy','Sensitivity','Specificity','Precision']
plt.figure(figsize=(10,3))



#Complete
plt.scatter(steps, complete_values_plot, color='red', label='Complete')
plt.scatter(steps, feature_Selection_plot, color='green', label='Feature selection')
plt.scatter(steps, pca_selection_plot, color='blue', label='PCA selection')
plt.scatter(steps, ica_selection_plot, color='brown', label='ICA selection')
plt.scatter(steps, rand_selection_plot, color='orange', label='PCA selection')

plt.legend()
plt.show()
```


0.8
0.7
0.6
0.5
0.4
Complete
Feature selection
PCA selection
ICA selection
PCA selection
Accuracy
Sensitivity
Specificity
Precision

Question 5

In [380]:

```
x_k_means_1 = x_counter_strike_data
kmeans = KMeans(algorithm = "full",
                            n_clusters=10,
                            random_state=10).fit(x_k_mean
s_1)
y_kmeans = kmeans.predict(x_k_means_1)

X_Train, X_Test, Y_Train, Y_Test = train_test_split(np.ar
ray(y_kmeans)
                                                    , np.
array(y_counter_strike_data),

test_size=0.3, random_state=1)

X_Train = X_Train.reshape(-1,1)
Y_Train = Y_Train.reshape(-1,1)

X_Test = X_Test.reshape(-1,1)
Y_Test = Y_Test.reshape(-1,1)
```

In [381]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_size
s=(4,3), random_state=1,max_iter = 3000)
clf.fit(X_Train,Y_Train)
predicted_classes = clf.predict(X_Test)
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\n
eural_network\multilayer_perceptron.py:921:
DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
```

In [382]:

```
cm = confusion_matrix(Y_Test, predicted_classes)
print(cm)
print("Accuracy", (cm[1][1] + cm[0][0]) / (cm[1][1] + cm[
1][0] + cm[0][1] + cm[0][0]) )
print("Sensitivity", cm[1][1] / (cm[1][1] + cm[1][0] ))
print("Specificity", cm[0][0] / (cm[0][0] + cm[0][1] ))
print("Precision", cm[1][1] / (cm[1][1] + cm[0][1] ))
```

```
[[134  43   0]
 [106  33   0]
 [ 10  14   0]]
Accuracy 0.5284810126582279
Sensitivity 0.23741007194244604
Specificity 0.7570621468926554
Precision 0.4342105263157895
```

Exp max - neural nets

In [385]:

```
x_exp_neural = x_k_means_1
g = mixture.GaussianMixture(covariance_type = 'full',n_co
mponents=17).fit(x_exp_neural)
labels = g.predict(x_exp_neural)

X_Train, X_Test, Y_Train, Y_Test = train_test_split(np.ar
ray(labels)
                                                      , np.
array(y_counter_strike_data),

test_size=0.3, random_state=1)

X_Train = X_Train.reshape(-1,1)
Y_Train = Y_Train.reshape(-1,1)

X_Test = X_Test.reshape(-1,1)
Y_Test = Y_Test.reshape(-1,1)

start_time = time.time()
clf = MLPClassifier(activation = 'tanh',random_state=1,hi
dden_layer_sizes=(4,3),max_iter = 3000)
clf.fit(X_Train,Y_Train)
predicted_classes = clf.predict(X_Test)
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\n
eural_network\multilayer_perceptron.py:921:
DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please
change the shape of y to (n_samples, ), for
example using ravel().
  y = column_or_1d(y, warn=True)
```

In [386]:

```
cm = confusion_matrix(Y_Test, predicted_classes)
print(cm)
print("Accuracy", (cm[1][1] + cm[0][0]) / (cm[1][1] + cm[
1][0] + cm[0][1] + cm[0][0]) )
print("Sensitivity", cm[1][1] / (cm[1][1] + cm[1][0] ))
print("Specificity", cm[0][0] / (cm[0][0] + cm[0][1] ))
print("Precision", cm[1][1] / (cm[1][1] + cm[0][1] ))
```

```
[[151  26   0]
 [114  25   0]
 [  7   8   9]]
Accuracy 0.5569620253164557
Sensitivity 0.17985611510791366
Specificity 0.8531073446327684
Precision 0.49019607843137253
```