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

In [72]:

```
#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
```

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


import time
```

In [73]:

```
electricity_data = pd.read_csv("energydata_complete.csv")
electricity_data_appliance  = electricity_data.drop(['dat
e','lights'] , axis = 1)
```

In [74]:

```
x_electricity = electricity_data_appliance.drop(labels =
['Appliances'],axis = 1)
y_electricity = electricity_data_appliance[['Appliances'
]]
```

In [75]:

```
scaler = MinMaxScaler()

x_electricity = scaler.fit_transform(x_electricity)
y_electricity = scaler.fit_transform(y_electricity)
```

In [76]:

```
pd.DataFrame(y_electricity).median()
y_electricity = np.where(y_electricity<0.04,0,1)
```

In [77]:

```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(x_ele
ctricity, y_electricity, test_size=0.3,
                                                     random
_state=1)
```

Section 1 : K-Means

1(i) K-Means applied on all columns

In [78]:

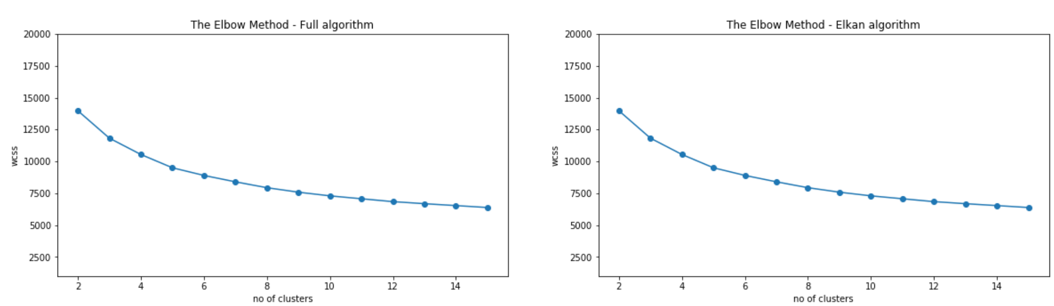
```
alogrithm_list_k_means = ["full", "elkan"] #i
number_of_clusters_k_means = range(2, 16) #j
Within_Cluster_Sum_of_Squares = []
currentCluster_number = []

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_electricity)
            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 [79]:

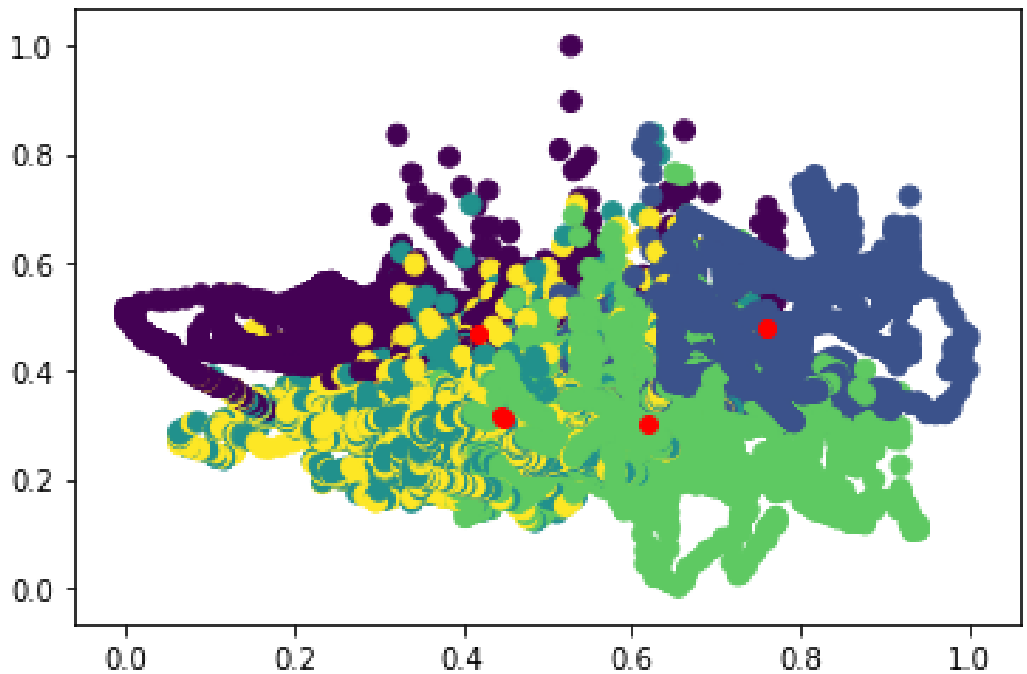
```
plt.figure(figsize=(20,5))


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')
plt.ylim(1000,20000,2000)

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.ylim(1000,20000,2000)
plt.show()
```



1 (iii) K-Means Plot

In [80]:

```
kmeans = KMeans(algorithm = "full",
                            n_clusters=5,
                            random_state=10).fit(x_electr
icity)

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

plt.show()
```



Part 2 : Feature selection using Random Forest

2(i) Feature Selection using Random Forest

In [81]:

```
x_electricity = electricity_data_appliance.drop(labels = 
['Appliances'],axis = 1)
y_electricity = electricity_data_appliance[['Appliances'
]]
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_electricity, y_electricity)
    sel.get_support()
    selected_feat= x_electricity.columns[(sel.get_support
())]

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

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\f
eature_selection\from_model.py:196: DataConv
ersionWarning: 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().
  self.estimator_.fit(X, y, **fit_params)


Step: 1 / 9

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\f
eature_selection\from_model.py:196: DataConv
ersionWarning: 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().
  self.estimator_.fit(X, y, **fit_params)

Step: 2 / 9

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\f
eature_selection\from_model.py:196: DataConv
ersionWarning: 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().
  self.estimator_.fit(X, y, **fit_params)

Step: 3 / 9
```

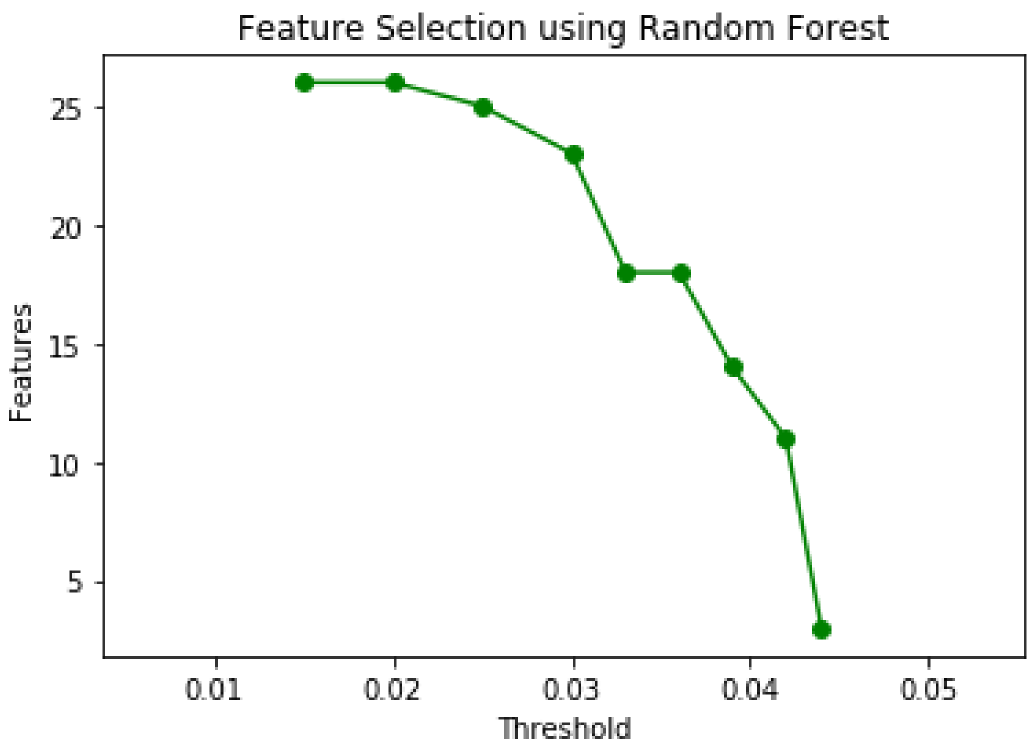
```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\f
eature_selection\from_model.py:196: DataConv
ersionWarning: A column-vector y was passed
when a 1d array was expected. Please change
the shape of y to (n_samples,), for example
using ravel().
  self.estimator_.fit(X, y, **fit_params)

Step: 4 / 9

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\f
eature_selection\from_model.py:196: DataConv
ersionWarning: 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().
  self.estimator_.fit(X, y, **fit_params)


Step: 5 / 9

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\f
eature_selection\from_model.py:196: DataConv
ersionWarning: 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().
  self.estimator_.fit(X, y, **fit_params)

Step: 6 / 9
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\f
eature_selection\from_model.py:196: DataConv
ersionWarning: 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().
  self.estimator_.fit(X, y, **fit_params)

Step: 7 / 9

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\f
eature_selection\from_model.py:196: DataConv
ersionWarning: 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().
  self.estimator_.fit(X, y, **fit_params)

Step: 8 / 9

c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\f
eature_selection\from_model.py:196: DataConv
ersionWarning: 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().
  self.estimator_.fit(X, y, **fit_params)

Step: 9 / 9
```

2(ii) Feature Selection using Random Forest - Plot

In [82]:

```
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()
```



Part 3 : Selected features are used to build K-Means

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

In [83]:

```
sel = SelectFromModel(RandomForestClassifier(n_estimators
= 100), threshold=0.033)
sel.fit(x_electricity, y_electricity)
sel.get_support()
selected_feat= x_electricity.columns[(sel.get_support())]
print(selected_feat)
```

```
c:\users\siddharth\appdata\local\programs\py
thon\python37-32\lib\site-packages\sklearn\f
eature_selection\from_model.py:196: DataConv
ersionWarning: 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().
  self.estimator_.fit(X, y, **fit_params)

Index(['RH_1', 'T2', 'RH_2', 'RH_3', 'RH_4',
'RH_5', 'T6', 'RH_6', 'RH_7',
       'T8', 'RH_8', 'RH_9', 'T_out', 'Press
_mm_hg', 'RH_out', 'Tdewpoint',
       'rv1', 'rv2'],
      dtype='object')
```

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

In [84]:

```
x_electricity_transformed_feature_sel = electricity_data_appliance[['RH_1', 'T2', 'RH_2', 'RH_3', 'RH_4', 'RH_5', 'T6', 'RH_6', 'RH_7',
       'T8', 'RH_8', 'RH_9', 'T_out', 'Press_mm_hg', 'RH_out', 'Tdewpoint',
       'rv1', 'rv2']]
```

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

In [85]:

```
scaler = MinMaxScaler()
x_electricity_transformed_feature_sel = scaler.fit_transform(x_electricity_transformed_feature_sel)
```

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

In [86]:

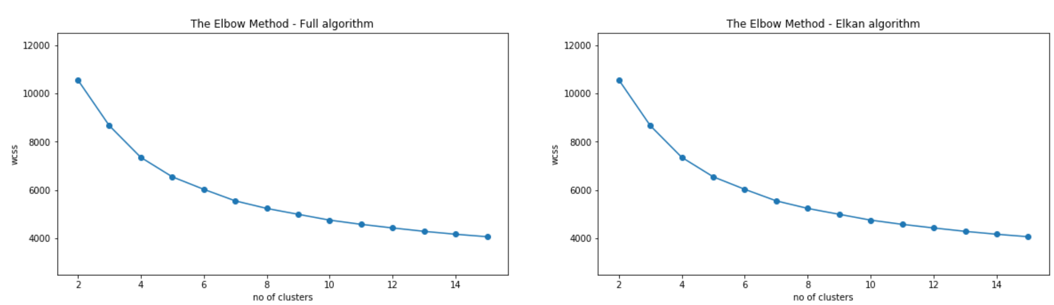
```
alogrithm_list_k_means = ["full", "elkan"] #i
number_of_clusters_k_means = range(2, 16) #j
Within_Cluster_Sum_of_Squares = []
currentCluster_number = []

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_electricity_transformed_feature_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 [87]:

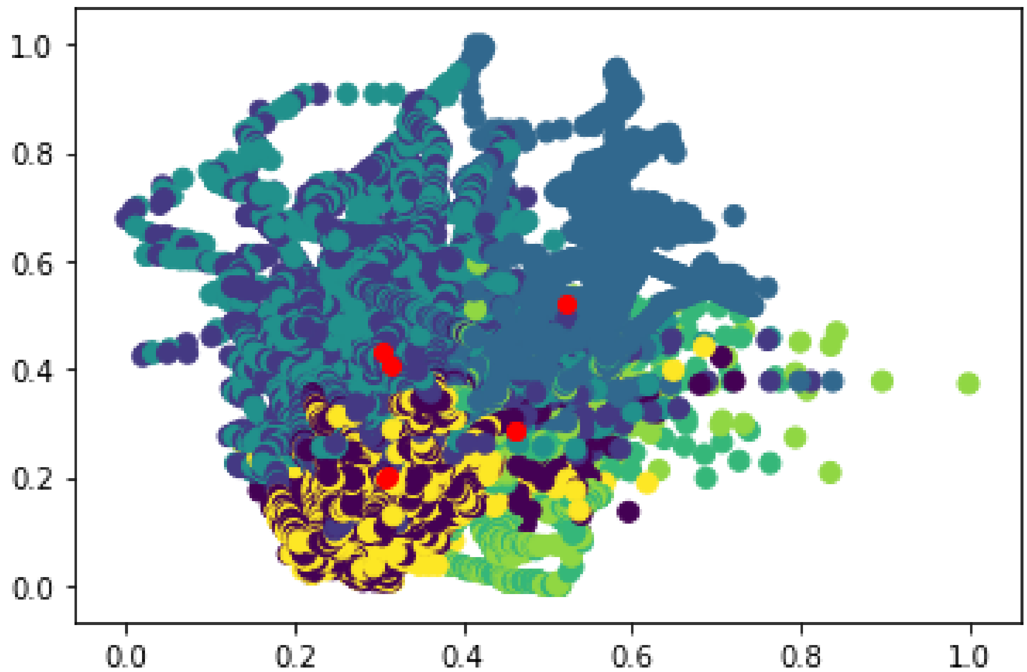
```
plt.figure(figsize=(20,5))



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')
plt.ylim(2500,12500,2000)

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.ylim(2500,12500,2000)

plt.show()
```



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

In [88]:

```
kmeans = KMeans(algorithm = "full",
                            n_clusters=7,
                            random_state=10).fit(x_electr
icity_transformed_feature_sel)

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

plt.show()
```



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

In [89]:

```
kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1
]
```

Out[89]:

```
(array([0.30941571, 0.31333952, 0.5207726 ,
0.30472535, 0.46236505,
        0.46123147, 0.30627252]),
 array([0.20357192, 0.40584126, 0.52070434,
0.43462151, 0.28530735,
        0.28542335, 0.19539997]))
```

Part 4 - PCA

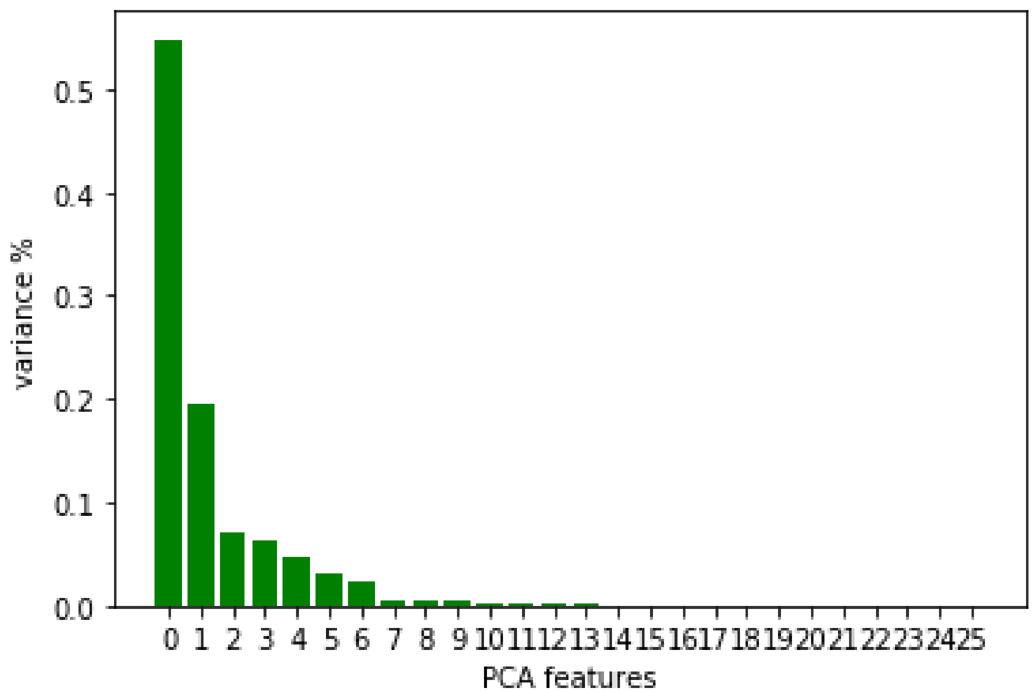
In [90]:

```
x_electricity = electricity_data_appliance.drop(labels =
['Appliances'],axis = 1)
```

Part 4(i) PCA performed

In [91]:

```
pca = PCA(n_components=26)
principalComponents = pca.fit_transform(x_electricity)
# 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 [92]:

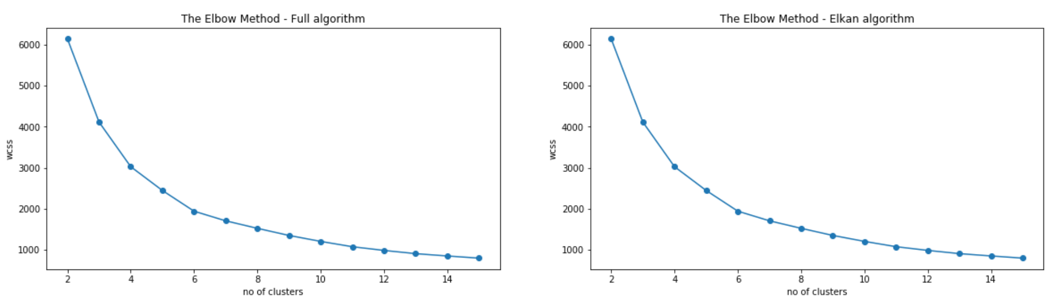
```
x_electricity = scaler.fit_transform(x_electricity)
principalComponents = pca.fit_transform(x_electricity)
PCA_components = pd.DataFrame(principalComponents)



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 [93]:

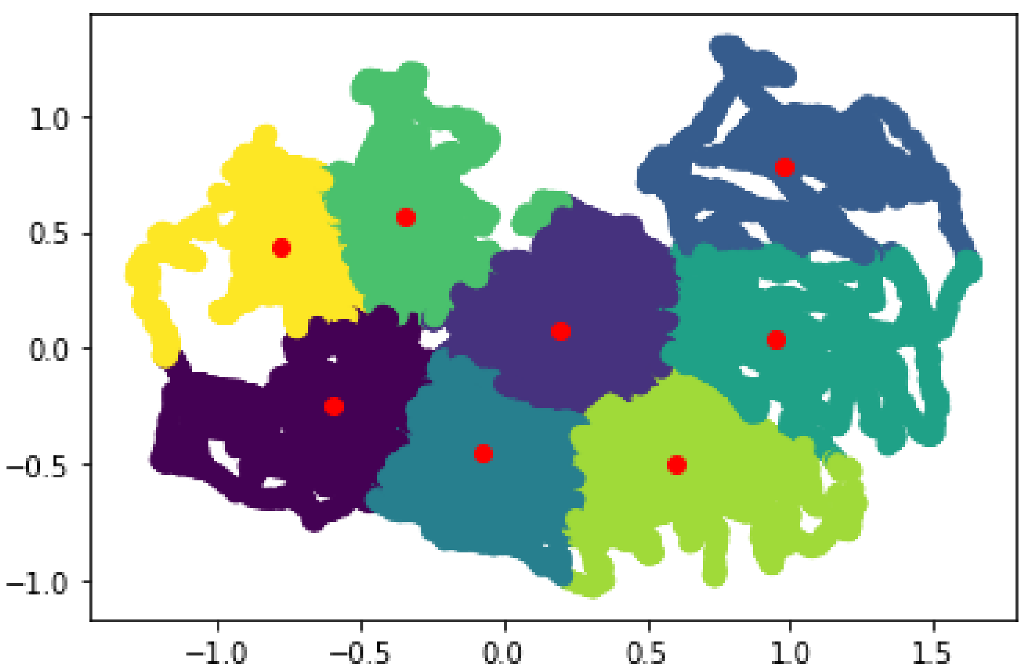
```
plt.figure(figsize=(20,5))


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 [94]:

```
kmeans = KMeans(algorithm = "full",
                            n_clusters=8,
                            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()
```



4(v) Plotting K-means for PCA - Centroids

In [95]:

```
kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1
]
```

Out[95]:

```
(array([-0.59281473,  0.1977377 ,  0.9760464
4, -0.0744117 ,  0.94794543,
        -0.33939894,  0.59860742, -0.7738532
7]),
 array([-0.24624601,  0.07421771,  0.7841190
3, -0.4456552 ,  0.03485481,
         0.56320204, -0.49661227,  0.4316189
7]))
```

Part 5 - ICA

In [96]:

```
x_electricity = electricity_data_appliance.drop(labels =
['Appliances'],axis = 1)
```

Part 5(i) ICA performed

In [97]:

```
ica = FastICA(n_components=26, random_state=10)
x_electricity_ica = ica.fit_transform(x_electricity)
```

```
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 [98]:

```
#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_electricity_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
```

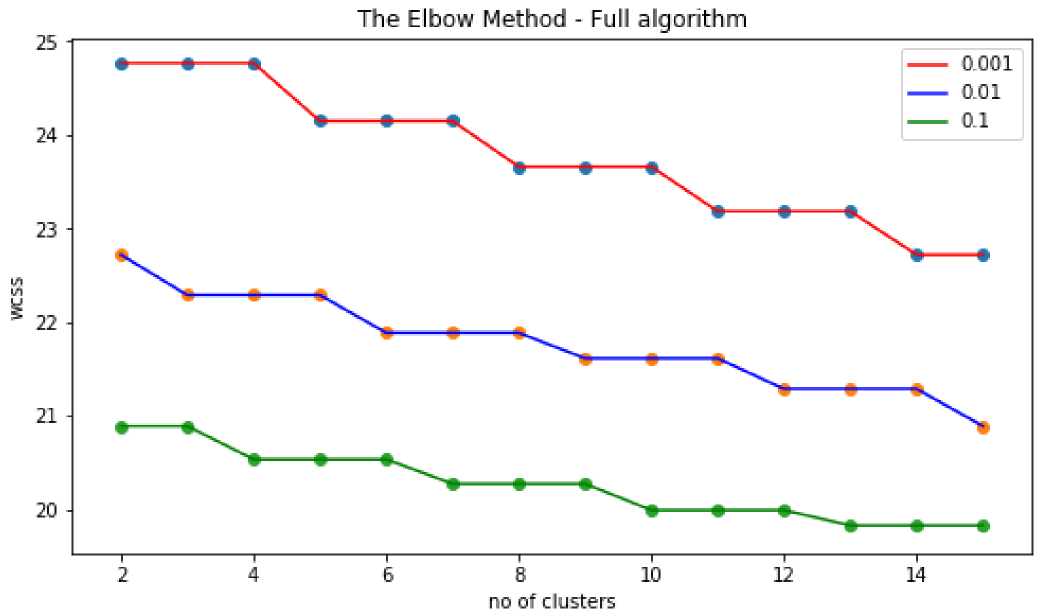
```
Step: 1 / 42
Step: 2 / 42
Step: 3 / 42
Step: 4 / 42
Step: 5 / 42
Step: 6 / 42
Step: 7 / 42
Step: 8 / 42
Step: 9 / 42
Step: 10 / 42
Step: 11 / 42
Step: 12 / 42
Step: 13 / 42
Step: 14 / 42
Step: 15 / 42
Step: 16 / 42
Step: 17 / 42
Step: 18 / 42
Step: 19 / 42
Step: 20 / 42
Step: 21 / 42
Step: 22 / 42
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 [99]:

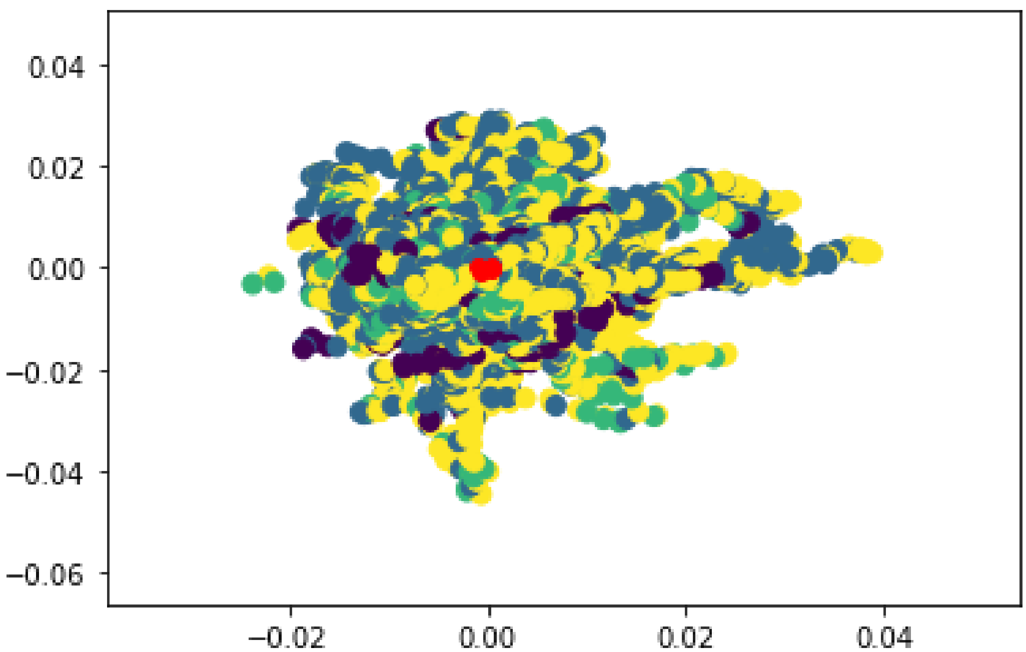
```
plt.figure(figsize=(20,5))

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 [100]:

```
kmeans = KMeans(algorithm = "full",
                            n_clusters=4,
                            random_state=10, tol = 0.1).f
it(x_electricity_ica)

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

plt.show()
```



5(v) Plotting K-means for ICA - Centroids

In [101]:

```
kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1
]
```

Out[101]:

```
(array([-0.00057949,  0.00039886, -0.000877
,  0.00027398]),
 array([-0.00061377,  0.00028974,  0.0004983
2, -0.00044796]))
```

Part 6 - Randomized projects

In [102]:

```
x_electricity = electricity_data_appliance.drop(labels =
['Appliances'],axis = 1)
x_electricity = scaler.fit_transform(x_electricity)
```

Part 6(i) Randomized projections performed

In [103]:

```
transformer = GaussianRandomProjection(random_state=10, n
_components=26)
x_electricity_randomized_projects = transformer.fit_trans
form(x_electricity)
```

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

In [104]:

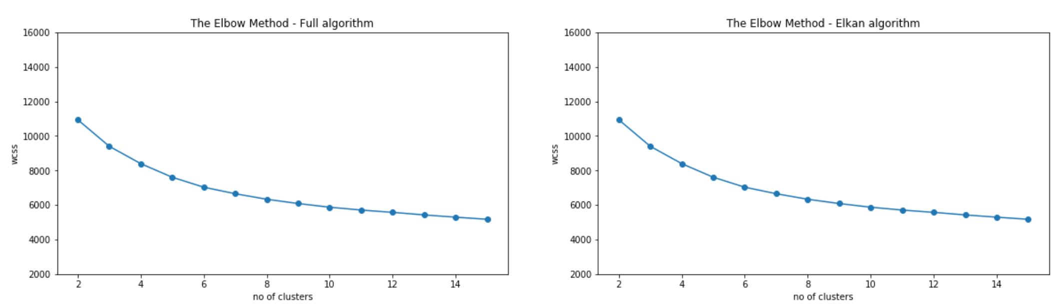
```
alogrithm_list_k_means = ["full", "elkan"] #i
number_of_clusters_k_means = range(2, 16) #j
Within_Cluster_Sum_of_Squares = []
currentCluster_number = []

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_electricity_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
```

```
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(iii) Randomized projections performed to build K-Means - Plot

In [105]:

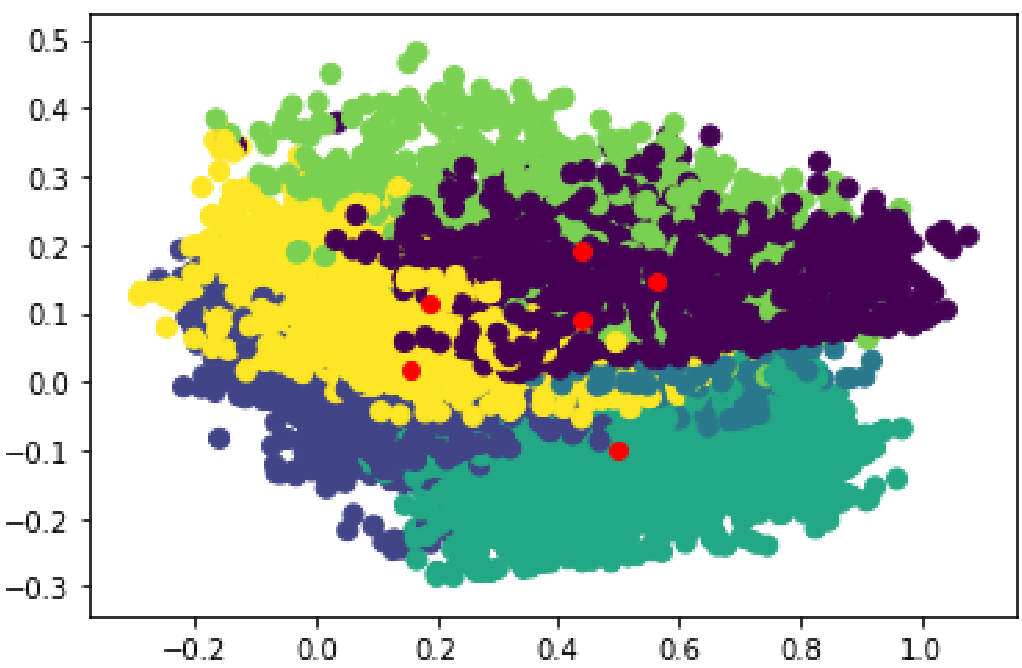
```
plt.figure(figsize=(20,5))

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')
plt.ylim(2000,16000)

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.ylim(2000,16000)

plt.show()
```



6(iv) Plotting K-means for Randomized projections

In [106]:

```
kmeans = KMeans(algorithm = "full",
                            n_clusters=6,
                            random_state=10).fit(x_electr
icity_randomized_projects)

y_kmeans = kmeans.predict(x_electricity_randomized_projec
ts)
plt.scatter(x_electricity_randomized_projects[:, 0], x_el
ectricity_randomized_projects[:, 1], c=y_kmeans, s=50, cm
ap='viridis')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_
centers_[:,1], color='red')

plt.show()
```



6(v) Plotting K-means for Randomized projections - Centroids

In [107]:

```
kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1
]
```

Out[107]:

```
(array([0.56381508, 0.15647184, 0.44116684,
0.49946479, 0.43879097,
        0.18834999]),
 array([ 0.14731753,  0.01802941,  0.0919015
1, -0.09857605,  0.19071245,
         0.11342978]))
```

Section 2 : Expectation Maximization

7(i) Expectation Maximization on the complete dataset

In [108]:

```
x_electricity = electricity_data_appliance.drop(labels = 
['Appliances'],axis = 1)
scaler = MinMaxScaler()
x_electricity_transformed_exp_max = scaler.fit_transform(
x_electricity)

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(x_electricity_transformed_exp_max)
        appendAicValues.append(g.bic(x_electricity_transf
ormed_exp_max))
        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
```

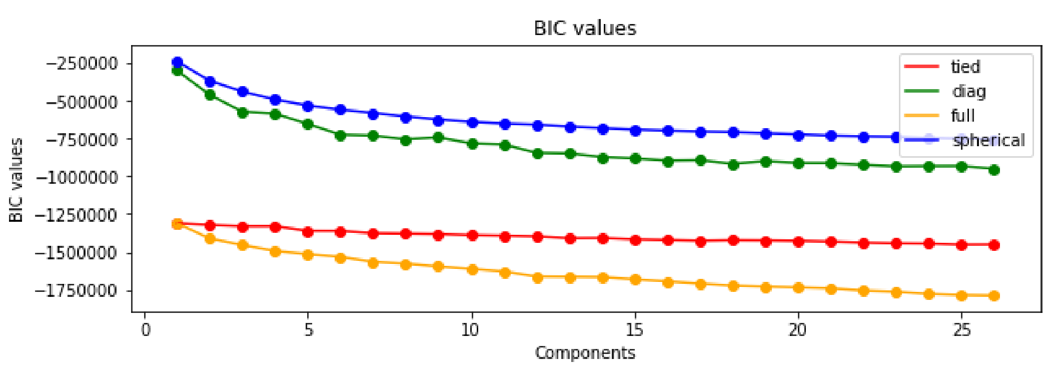
```
Step :  38 / 104
Step :  39 / 104
Step :  40 / 104
Step :  41 / 104
Step :  42 / 104
Step :  43 / 104
Step :  44 / 104
Step :  45 / 104
Step :  46 / 104
Step :  47 / 104
Step :  48 / 104
Step :  49 / 104
Step :  50 / 104
Step :  51 / 104
Step :  52 / 104
Step :  53 / 104
Step :  54 / 104
Step :  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
```

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

In [109]:

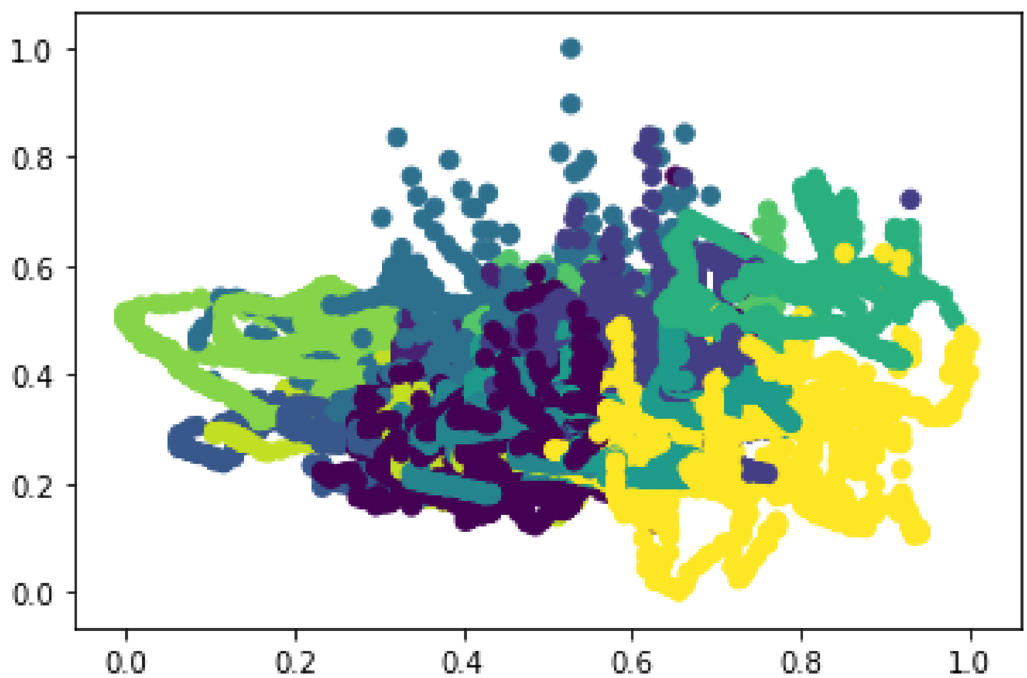
```
steps = np.arange(1,27,1)
plt.figure(figsize=(10,3))

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()
```



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

In [110]:

```
g = mixture.GaussianMixture(covariance_type = 'full',n_co
mponents=12).fit(x_electricity_transformed_exp_max)
labels = g.predict(x_electricity_transformed_exp_max)
plt.scatter(x_electricity_transformed_exp_max[:, 0], x_el
ectricity_transformed_exp_max[:, 1],
            c=labels, s=40, cmap='viridis');
```



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

In [111]:

```
x_electricity_transformed_feature_sel = electricity_data_appliance[['RH_1', 'T2', 'RH_2', 'RH_3', 'RH_4', 'RH_5', 'T6', 'RH_6', 'RH_7',
       'T8', 'RH_8', 'RH_9', 'T_out', 'Press_mm_hg', 'RH_out', 'Tdewpoint',
       'rv1', 'rv2']]

scaler = MinMaxScaler()
x_electricity_transformed_feature_sel = scaler.fit_transform(x_electricity_transformed_feature_sel)
```

8 (ii) Feature selection dataset - Random forest

In [112]:

```
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_electricity_transformed_feature_sel)
        appendAicValues.append(g.bic(x_electricity_transf
ormed_feature_sel))
        step = step + 1
        print("Step : ", step , "/", len(cov_type) * len(
number_of_components))
```

```
Step :  1 / 100
Step :  2 / 100
Step :  3 / 100
Step :  4 / 100
Step :  5 / 100
Step :  6 / 100
Step :  7 / 100
Step :  8 / 100
Step :  9 / 100
Step :  10 / 100
Step :  11 / 100
Step :  12 / 100
Step :  13 / 100
Step :  14 / 100
Step :  15 / 100
Step :  16 / 100
Step :  17 / 100
Step :  18 / 100
Step :  19 / 100
Step :  20 / 100
Step :  21 / 100
Step :  22 / 100
Step :  23 / 100
Step :  24 / 100
Step :  25 / 100
Step :  26 / 100
Step :  27 / 100
Step :  28 / 100
Step :  29 / 100
Step :  30 / 100
Step :  31 / 100
Step :  32 / 100
Step :  33 / 100
Step :  34 / 100
Step :  35 / 100
Step :  36 / 100
Step :  37 / 100
```

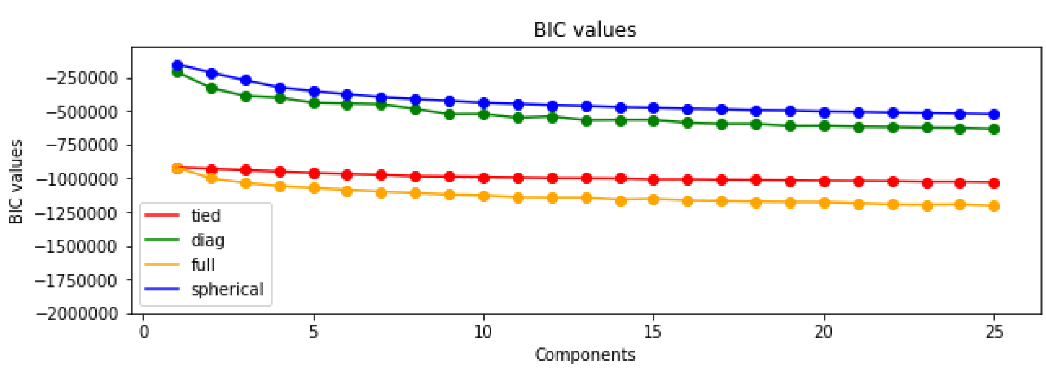
```
Step :  38 / 100
Step :  39 / 100
Step :  40 / 100
Step :  41 / 100
Step :  42 / 100
Step :  43 / 100
Step :  44 / 100
Step :  45 / 100
Step :  46 / 100
Step :  47 / 100
Step :  48 / 100
Step :  49 / 100
Step :  50 / 100
Step :  51 / 100
Step :  52 / 100
Step :  53 / 100
Step :  54 / 100
Step :  55 / 100
Step :  56 / 100
Step :  57 / 100
Step :  58 / 100
Step :  59 / 100
Step :  60 / 100
Step :  61 / 100
Step :  62 / 100
Step :  63 / 100
Step :  64 / 100
Step :  65 / 100
Step :  66 / 100
Step :  67 / 100
Step :  68 / 100
Step :  69 / 100
Step :  70 / 100
Step :  71 / 100
Step :  72 / 100
Step :  73 / 100
Step :  74 / 100
```

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

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

In [113]:

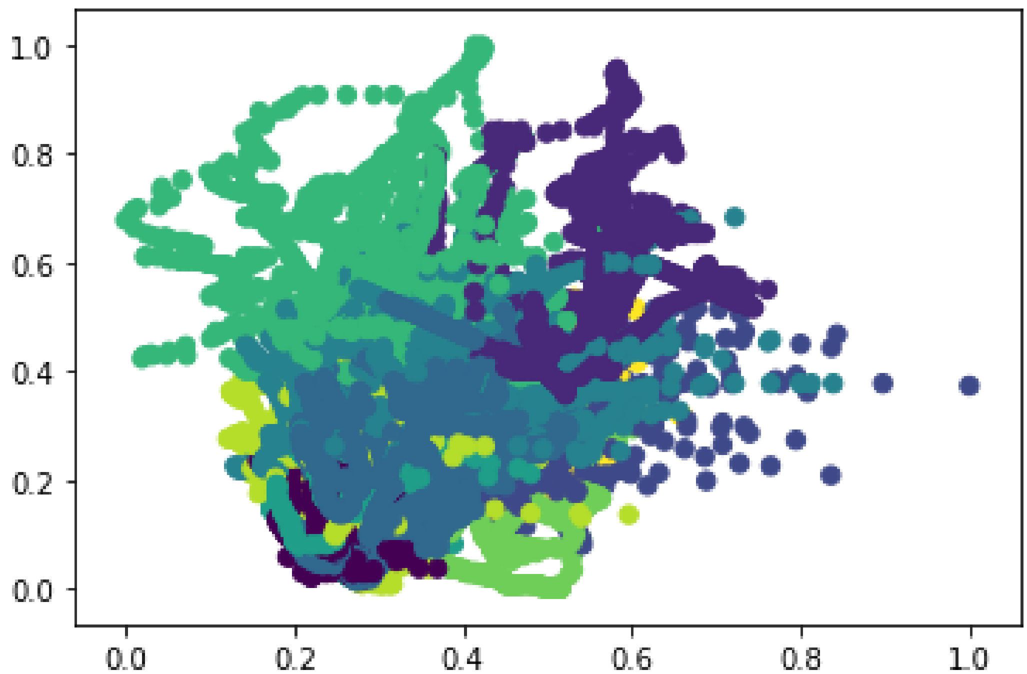
```
steps = np.arange(1,26,1)
plt.figure(figsize=(10,3))

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', la
bel='diag')

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

plt.scatter(steps, appendAicValues[75:100], color='blue')
plt.plot(steps, appendAicValues[75:100], color='blue', la
bel='spherical')
plt.ylim(-2000000,-20000)

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 [114]:

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



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

In [115]:

```
x_electricity = electricity_data_appliance.drop(labels =
['Appliances'],axis = 1)
scaler = MinMaxScaler()
x_electricity_pca = scaler.fit_transform(x_electricity)

principalComponents = pca.fit_transform(x_electricity_pca
)
PCA_components = pd.DataFrame(principalComponents)

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
```

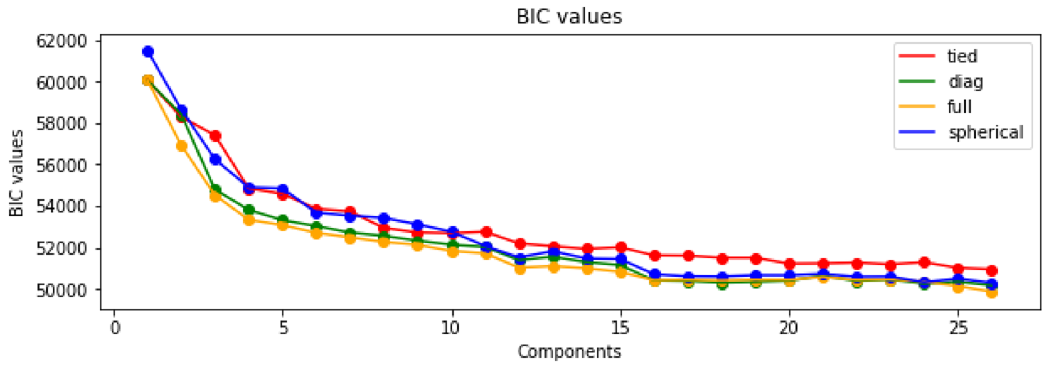
```
Step :  38 / 104
Step :  39 / 104
Step :  40 / 104
Step :  41 / 104
Step :  42 / 104
Step :  43 / 104
Step :  44 / 104
Step :  45 / 104
Step :  46 / 104
Step :  47 / 104
Step :  48 / 104
Step :  49 / 104
Step :  50 / 104
Step :  51 / 104
Step :  52 / 104
Step :  53 / 104
Step :  54 / 104
Step :  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 [116]:

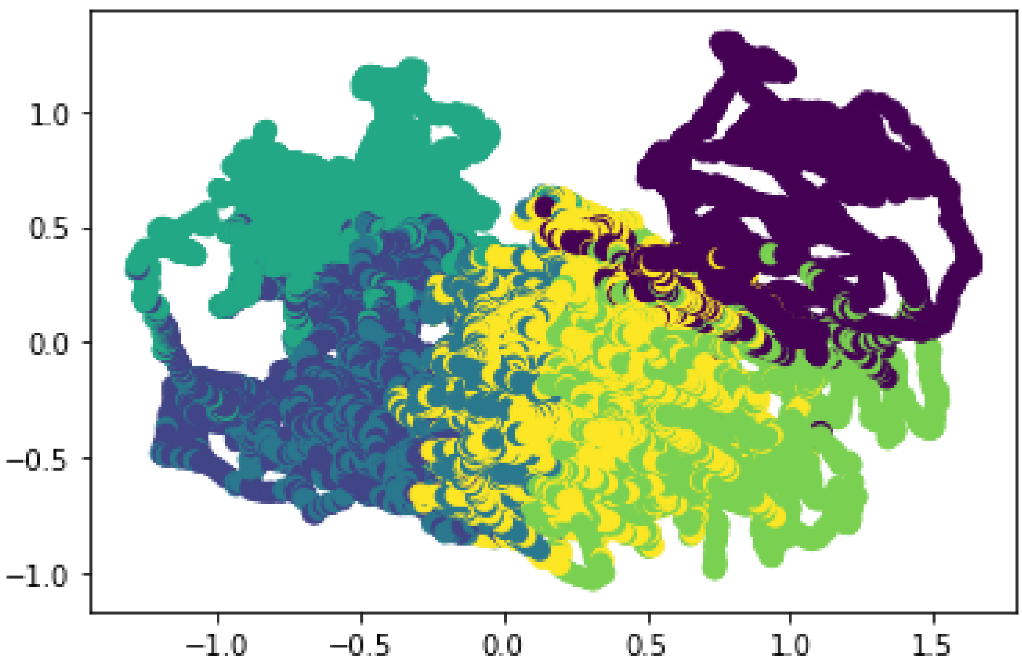
```
steps = np.arange(1,27,1)
plt.figure(figsize=(10,3))

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 [117]:

```
g = mixture.GaussianMixture(covariance_type = 'full',n_co
mponents=10, 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 [118]:

```
x_electricity = electricity_data_appliance.drop(labels =
['Appliances'],axis = 1)
scaler = MinMaxScaler()
x_electricity = scaler.fit_transform(x_electricity)

ica = FastICA(n_components=26, random_state=10)
x_electricity_ica = scaler.fit_transform(x_electricity)


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(x_electricity_ica)
        appendAicValues.append(g.bic(x_electricity_ica))
        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
```

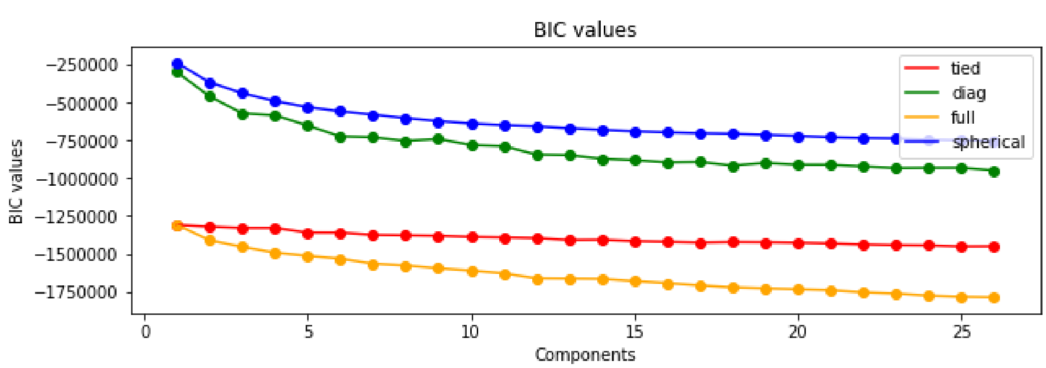
```
Step :  38 / 104
Step :  39 / 104
Step :  40 / 104
Step :  41 / 104
Step :  42 / 104
Step :  43 / 104
Step :  44 / 104
Step :  45 / 104
Step :  46 / 104
Step :  47 / 104
Step :  48 / 104
Step :  49 / 104
Step :  50 / 104
Step :  51 / 104
Step :  52 / 104
Step :  53 / 104
Step :  54 / 104
Step :  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
```

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

In [119]:

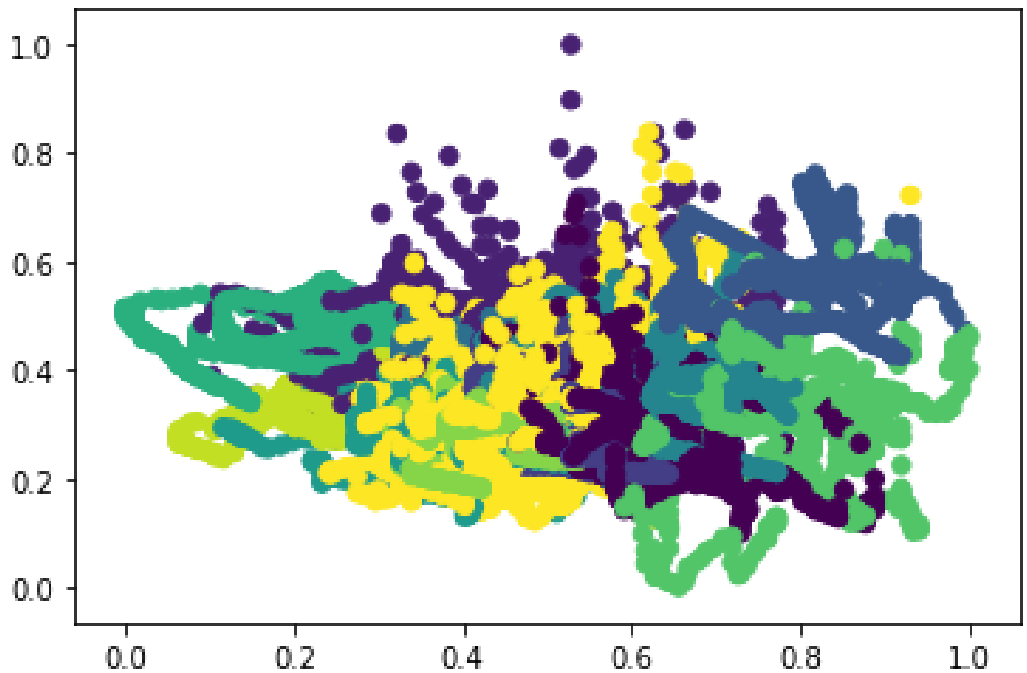
```
steps = np.arange(1,27,1)
plt.figure(figsize=(10,3))

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()
```



11 (iii) Expectation Maximization on ICA - Separation plot

In [120]:

```
g = mixture.GaussianMixture(covariance_type = 'full',n_co
mponents=12).fit(x_electricity_ica)
labels = g.predict(x_electricity)
plt.scatter(x_electricity_ica[:, 0], x_electricity_ica[:,
1], c=labels, s=40, cmap='viridis');
```



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

In [121]:

```
x_electricity = electricity_data_appliance.drop(labels =
['Appliances'],axis = 1)
x_electricity = scaler.fit_transform(x_electricity)

transformer = GaussianRandomProjection(random_state=10, n
_components=26)
x_electricity_randomized_projects = transformer.fit_trans
form(x_electricity)


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(x_electricity_randomized_projects)
        appendAicValues.append(g.bic(x_electricity_random
ized_projects))
        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
```

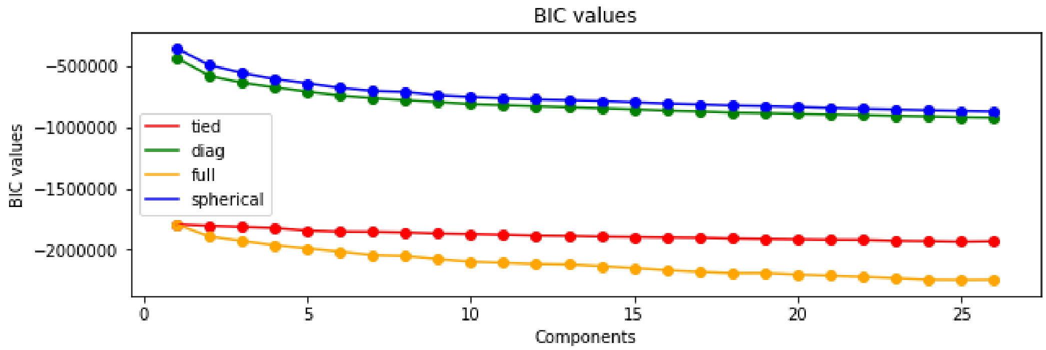
```
Step :  38 / 104
Step :  39 / 104
Step :  40 / 104
Step :  41 / 104
Step :  42 / 104
Step :  43 / 104
Step :  44 / 104
Step :  45 / 104
Step :  46 / 104
Step :  47 / 104
Step :  48 / 104
Step :  49 / 104
Step :  50 / 104
Step :  51 / 104
Step :  52 / 104
Step :  53 / 104
Step :  54 / 104
Step :  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
```

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

In [122]:

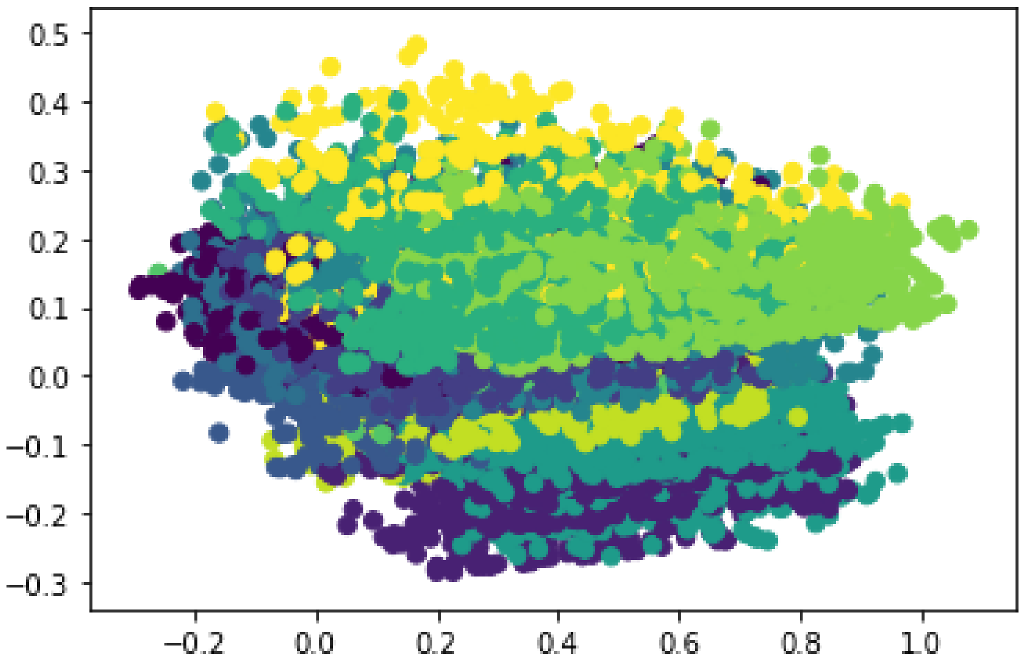
```
steps = np.arange(1,27,1)
plt.figure(figsize=(10,3))

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()
```



12 (iii) Randomized projection - Separation plot

In [123]:

```
g = mixture.GaussianMixture(covariance_type = 'full',n_co
mponents=12).fit(x_electricity_randomized_projects)
labels = g.predict(x_electricity_randomized_projects)
plt.scatter(x_electricity_randomized_projects[:, 0], x_el
ectricity_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 [124]:

```
x_electricity = electricity_data_appliance.drop(labels = ['Appliances'],axis = 1)
y_electricity = electricity_data_appliance[['Appliances']]

scaler = MinMaxScaler()

x_electricity = scaler.fit_transform(x_electricity)
y_electricity = scaler.fit_transform(y_electricity)

pd.DataFrame(y_electricity).median()
y_electricity = np.where(y_electricity<0.04,0,1)


X_Train, X_Test, Y_Train, Y_Test = train_test_split(x_electricity, y_electricity, test_size=0.3,
                                                    random_state=1)
```

13(i) Neural networks on Entire data

In [125]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_size
s=(5,3), random_state=1,max_iter = 1000)
clf.fit(X_Train, Y_Train)
predicted_classes = clf.predict(X_Test)

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

```
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)

--- 14.332423686981201 seconds ---
```

13(ii) Neural networks on Entire data - parameters

In [126]:

```
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] ))
```

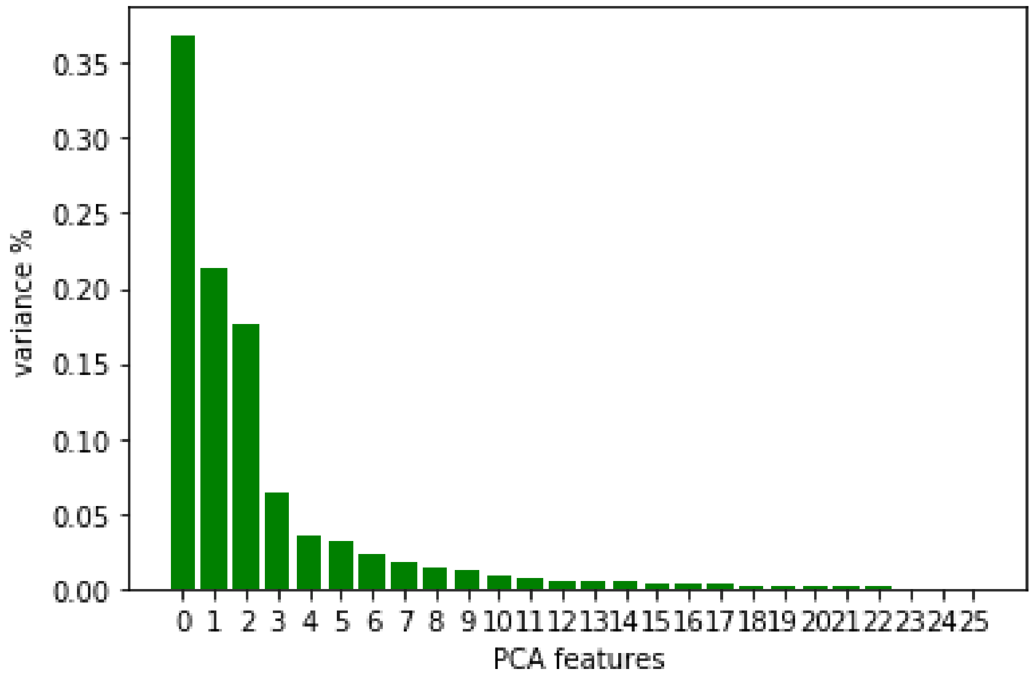
```
[[1376  835]
 [ 657 3053]]
Accuracy 0.7480155379158926
Sensitivity 0.8229110512129381
Specificity 0.6223428312980552
Precision 0.785236625514403
```

14 Neural networks : Based on PCA

14(i) Neural networks - PCA build

In [127]:

```
pca = PCA(n_components=26)
principalComponents = pca.fit_transform(x_electricity)
# 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)

x_electricity = electricity_data_appliance.drop(labels =
['Appliances'],axis = 1)
x_electricity_pca = scaler.fit_transform(x_electricity)
principalComponents = pca.fit_transform(x_electricity_pca
)
PCA_components = pd.DataFrame(principalComponents)


y_electricity = electricity_data_appliance[['Appliances'
]]
y_electricity = scaler.fit_transform(y_electricity)
scaler = MinMaxScaler()
y_electricity = np.where(y_electricity<0.04,0,1)
```



14(ii) Neural networks - PCA - Split training and test data

In [128]:

```
X_Train, X_Test, Y_Train, Y_Test = train_test_split(PCA_c
omponents.iloc[:,:2], y_electricity, test_size=0.3,
                                                    random
_state=1)
```

14(iii) Neural networks - PCA - Experiment

In [129]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_size
s=(5,3), random_state=1,max_iter = 1000)
clf.fit(X_Train, Y_Train)
predicted_classes = clf.predict(X_Test)

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

```
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)

--- 0.4903254508972168 seconds ---
```

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

In [130]:

```
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] ))
```

```
[[ 754 1457]
 [ 494 3216]]
Accuracy 0.6704948488431008
Sensitivity 0.8668463611859838
Specificity 0.3410221619176843
Precision 0.6882088594050931
```

15 Neural networks : Based on ICA

15 (i) Neural networks : Based on ICA Build

In [131]:

```
x_electricity = electricity_data_appliance.drop(labels =
['Appliances'],axis = 1)
x_electricity = scaler.fit_transform(x_electricity)
ica = FastICA(n_components=26, random_state=10)
x_electricity_ica = ica.fit_transform(x_electricity)

y_electricity = electricity_data_appliance[['Appliances'
]]
y_electricity = scaler.fit_transform(y_electricity)
scaler = MinMaxScaler()
y_electricity = np.where(y_electricity<0.04,0,1)




X_Train, X_Test, Y_Train, Y_Test = train_test_split(x_ele
ctricity_ica, y_electricity, test_size=0.3,
                                                    random
_state=1)
```

```
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)
```

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

In [132]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_size
s=(5,3), random_state=1,max_iter = 1000)
clf.fit(X_Train, Y_Train)
predicted_classes = clf.predict(X_Test)

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

```
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)

--- 2.9586353302001953 seconds ---
```

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

In [133]:

```
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] ))
```

```
[[1292  919]
 [ 688 3022]]
Accuracy 0.7285931430501604
Sensitivity 0.8145552560646
Specificity 0.5843509724106739
Precision 0.766810454199417
```

16 Neural networks : Based on Randomized projections

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

In [134]:

```
x_electricity = electricity_data_appliance.drop(labels = ['Appliances'],axis = 1)
x_electricity = scaler.fit_transform(x_electricity)

transformer = GaussianRandomProjection(random_state=10, n_components=26)
x_electricity_randomized_projects = transformer.fit_transform(x_electricity)

y_electricity = electricity_data_appliance[['Appliances']]
y_electricity = scaler.fit_transform(y_electricity)
scaler = MinMaxScaler()
y_electricity = np.where(y_electricity<0.04,0,1)



X_Train, X_Test, Y_Train, Y_Test = train_test_split(x_electricity_randomized_projects, y_electricity, test_size=0.3,
                                                    random_state=1)
```

16(ii) Neural networks : Based on randomized projections - Experiment

In [135]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_size
s=(5,3), random_state=1,max_iter = 1000)
clf.fit(X_Train, Y_Train)
predicted_classes = clf.predict(X_Test)

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

```
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)

--- 8.854084491729736 seconds ---
```

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

In [136]:

```
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] ))
```

```
[[1200 1011]
 [ 504 3206]]
Accuracy 0.7441310589427461
Sensitivity 0.8641509433962264
Specificity 0.5427408412483039
Precision 0.760256106236612
```

17 Neural networks : Based on Feature Selection

17(i) Neural networks : Feature Selection

In [137]:

```
x_electricity_transformed_feature_sel = electricity_data_
appliance[['RH_1', 'T2', 'RH_2', 'RH_3', 'RH_4', 'RH_5',
'T6', 'RH_6', 'RH_7',
       'T8', 'RH_8', 'RH_9', 'T_out', 'Press_mm_hg', 'RH_
out', 'Tdewpoint',
       'rv1', 'rv2']]
y_electricity = electricity_data_appliance[['Appliances'
]]


scaler = MinMaxScaler()
x_electricity_new = scaler.fit_transform(x_electricity_tr
ansformed_feature_sel)
y_electricity_new = scaler.fit_transform(y_electricity)
y_electricity_new = np.where(y_electricity_new<0.04,0,1)

X_Train, X_Test, Y_Train, Y_Test = train_test_split(x_ele
ctricity_new, y_electricity_new, test_size=0.3,
                                                   random
_state=1)
```

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

In [138]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_sizes=(5,3), random_state=1,max_iter = 1000)
clf.fit(X_Train, Y_Train)
predicted_classes = clf.predict(X_Test)

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

```
c:\users\siddharth\appdata\local\programs\python\python37-32\lib\site-packages\sklearn\neural_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)

--- 13.810113191604614 seconds ---
```

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

In [139]:

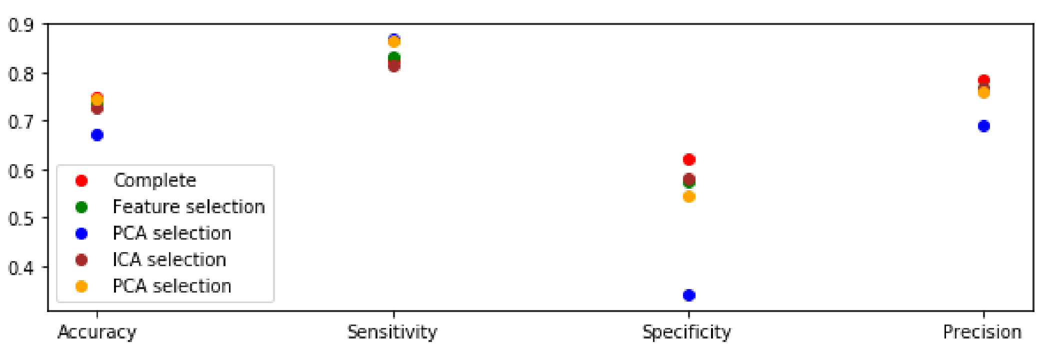
```
cm = confusion_matrix(Y_Test, predicted_classes)
print(cm)
print("Accuracy", (cm[1][1] + cm[0][0]) / np.sum(cm) )
print("Sensitivity", cm[1][1] / (cm[1][1] + cm[1][0] ))
print("Specificity", cm[0][0] / (cm[0][0] + cm[0][1] ))
print("Precision", cm[1][1] / (cm[1][1] + cm[0][1] ))
```

```
[[1270  941]
 [ 623 3087]]
Accuracy 0.7358554298260429
Sensitivity 0.8320754716981132
Specificity 0.574400723654455
Precision 0.7663853028798411
```

18 plot confusion matrix

In [140]:

```
complete_values_plot = [0.7480,0.8229,0.6223,0.7852]
feature_Selection_plot =  [0.7359,.8321,0.5744,0.7664]
pca_selection_plot = [0.6705,0.8668,0.3410,0.6882]
ica_selection_plot = [0.7277,0.8154,0.5807,0.7654]
rand_selection_plot = [0.7441,0.8642,0.5427,0.7603]




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


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

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



Question 5

In [148]:

```
scaler = MinMaxScaler()

x_electricity = scaler.fit_transform(x_electricity)
y_electricity = scaler.fit_transform(y_electricity)

pd.DataFrame(y_electricity).median()
y_electricity = np.where(y_electricity<0.04,0,1)

X_Train, X_Test, Y_Train, Y_Test = train_test_split(x_ele
ctricity, y_electricity, test_size=0.3,
                                                   random
_state=1)

x_k_means_1 = x_electricity
kmeans = KMeans(algorithm = "full",
                            n_clusters=7,
                            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_electricity),

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 [149]:

```
start_time = time.time()
clf = MLPClassifier(activation = 'tanh',hidden_layer_size
s=(5,3), 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 [150]:

```
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] ))
```

```
[[ 458 1753]
 [ 508 3202]]
Accuracy 0.6181388279006924
Sensitivity 0.8630727762803234
Specificity 0.20714608774310267
Precision 0.6462159434914228
```

Exp maximization

In [152]:

```
x_exp_neural = x_k_means_1
g = mixture.GaussianMixture(covariance_type = 'full',n_co
mponents=12).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_electricity),

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 [153]:

```
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] ))
```

```
[[ 899 1312]
 [ 651 3059]]
Accuracy 0.6684681641614593
Sensitivity 0.8245283018867925
Specificity 0.40660334690185435
Precision 0.6998398535804163
```

In [ ]:

```
```