Machine_learining_EEG

September 9, 2022

0.1 Machine learning with the extracted EEG parameters

0.1.1 Importing the packagery:

[1]:
```
#Python3.7
#Numpy version:
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_samples, silhouette_score

%matplotlib inline
```

Now we will try different combinations of parameters and evaluate the accuracy of a cluster-based analysis. Let's remember that we have two main types of parameters: the ones extracted from the bipolar derivations and the ones extracted from the original channels.

0.2 Standard deviation_ vertical_bipolar

This first derivation contains the standard deviation of the vertical bipolar chains obtained from the double banana. Each row of the CSV data represents a segment of the recording. The length of each segment depends on the duration of the sleep stage. Each segment is then subdivided into sub-segments of 10 seconds.

Then a PSD plot is obtained from each sub-segment and divided into 5 frequency bands: Delta waves(0.2-4 Hz), theta waves(4-8 Hz), alpha(8-12 Hz), beta(12-30 Hz) and gamma(30-90 Hz).

Finally, a standard deviation of the relative power of a frequency band is calculated. The standard deviations that correspond to the same segment and the same chain are averaged and recorded. This means that for a single segment we have 7 chains, each with 5 bands. This results in an array of 35 dependent variables. The last variable in the array is the independent variable: the sleep stage.

Importing the data

[3]:
```
data=pd.read_csv("Extracted_data/Data_medians_vert_STD.csv")
```

Adapting the data

[4]:
```
del data['Unnamed: 0']
```

[5]:
```
# 0: Awake
# 1: N1
# 2: N2
# 3: N3
# 4: REM

for i in range(0,len(data)):
    if data.iloc[i,-1]=='Awake':
        data.iloc[i,-1]=0
    elif data.iloc[i,-1]=='N1':
        data.iloc[i,-1]=1
    elif data.iloc[i,-1]=='N2':
        data.iloc[i,-1]=2
    elif data.iloc[i,-1]=='N3':
        data.iloc[i,-1]=3
    elif data.iloc[i,-1]=='REM':
        data.iloc[i,-1]=4
    else:
        data.iloc[i,-1]=np.nan
```

[6]:
```
data_2=data.dropna()
```

[7]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_2.iloc[:,:35])
```

[8]:
```
Y=data_2.iloc[:,-1].to_numpy()
```

[9]:
```
# Train-test split
X_train=X[:int(len(X)*0.9)]
X_test=X[int(len(X)*0.9):]
```

[10]:
```
Y_train=Y[:int(len(X)*0.9)]
Y_test=Y[int(len(X)*0.9):]
```

Defining a Kmeans model The KMeans algorithm clusters the data by attempting to divide the samples into n groups of equal variance. The mean is commonly called the "centroid" of the cluster. Note that these are typically not points from the dataset.

The k-means algorithm aims to choose centroids that minimize the in-cluster inertia or sum of squares criterion. The first step is to select the first centroid. The K-means algorithm consists of 3 basic steps. First, several points of the dataset are selected as the first centroids. Then the rest of the points are classified according to the nearest centroid. The new centroid is the mean value of the points in each class. The difference between the old and new centroids is calculated and the

algorithm repeats these last two steps until this value is below the threshold. For more information visit the documentation site.

[11]:
```
# Run local implementation of kmeans
model = KMeans(n_clusters=6, max_iter=100, init='random',n_init=10)
```

[12]:
```
model.fit(X_train)
```

[12]:
```
KMeans(init='random', max_iter=100, n_clusters=6)
```

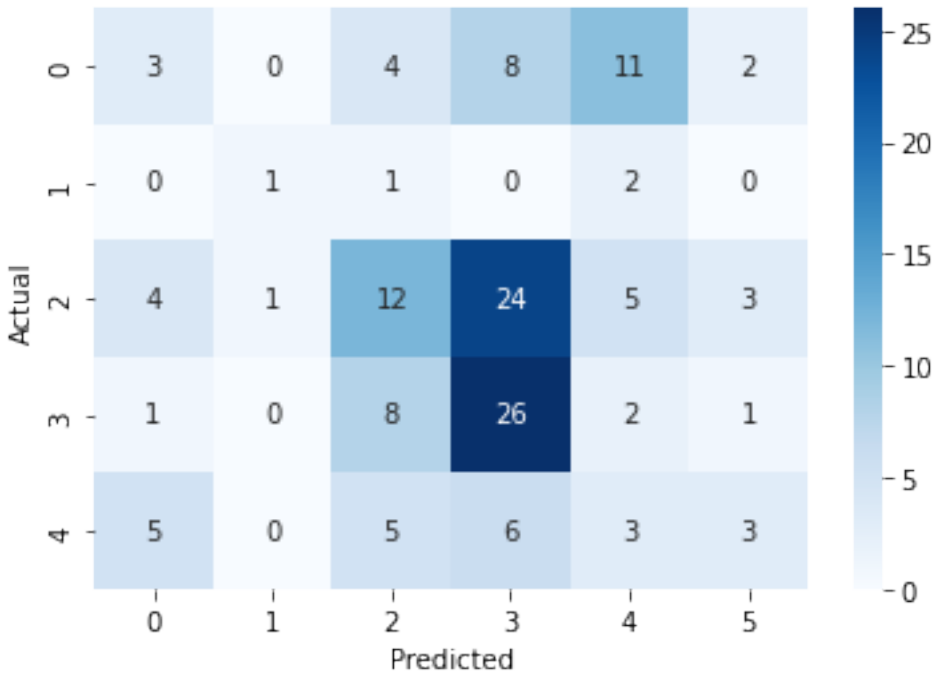
[13]:
```
#Obtaining clusters centroid
centroids = model.cluster_centers_

#To obtain the labels of each cluster
labels = model.labels_
```

Train values

[14]:
```
confusion_matrix = pd.crosstab(Y_train, labels, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```

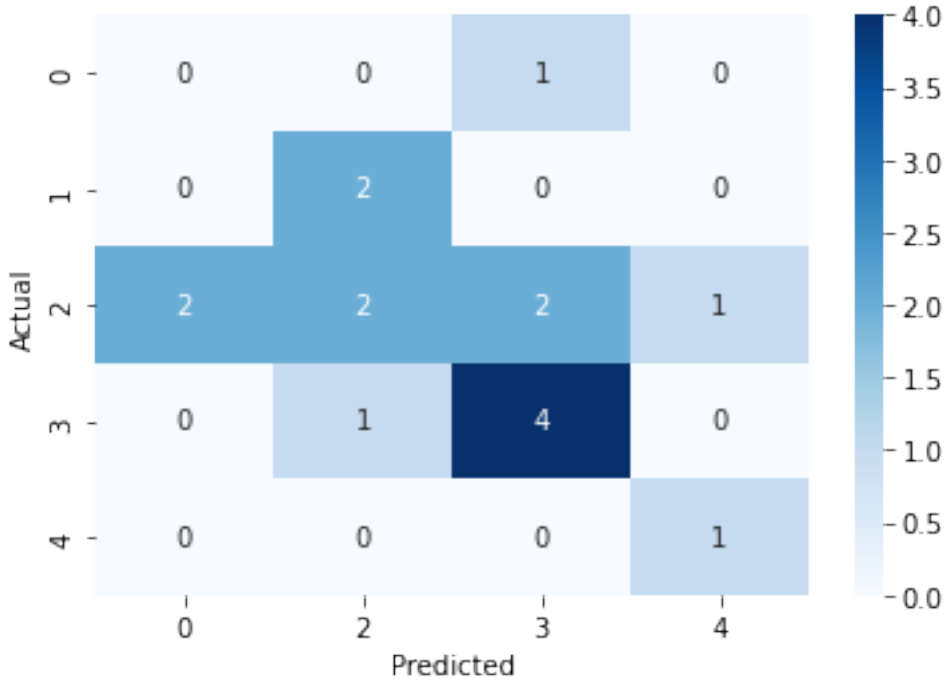


Test values

[15]:
```
y_pred=model.predict(X_test)
```

[16]:
```
confusion_matrix = pd.crosstab(Y_test, y_pred, rownames=['Actual'],␣
↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



Observations:

- STD works great to identify the N3 segments
- STD works fairly well to identify periods awake and N2
- STD doesn't seem to be working on REM identification
- The low count of N1 segments makes it difficult to assess its accuracy.
- It should be considered that it is possible that the third group of STD corresponds to epileptic seizures.

0.3 Medians: Lobes

Now we will obtain the average value for each band dividing each channel into lobes. The lobes taken into account were: Frontal, occipital, parietal, temporal, and central(even though it isn't a cerebral lobe, it doesn't share the same characteristics as the frontal or parietal lobes).

Importing the data

[19]:
```
data=pd.read_csv("Extracted_data/Data_medians_lobes.csv")
```

[20]:
```
del data['Unnamed: 0']
```

[21]:
```
# 0: Awake
# 1: N1
# 2: N2
# 3: N3
# 4: REM

for i in range(0,len(data)):
    if data.iloc[i,-1]=='Awake':
        data.iloc[i,-1]=0
    elif data.iloc[i,-1]=='N1':
        data.iloc[i,-1]=1
    elif data.iloc[i,-1]=='N2':
        data.iloc[i,-1]=2
    elif data.iloc[i,-1]=='N3':
        data.iloc[i,-1]=3
    elif data.iloc[i,-1]=='REM':
        data.iloc[i,-1]=4
    else:
        data.iloc[i,-1]=np.nan
```

Adapting the data

[22]:
```
data_2=data.dropna()
```

[23]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_2.iloc[:,:35])

Y=data_2.iloc[:,-1].to_numpy()

X_train=X[:int(len(X)*0.9)]
X_test=X[int(len(X)*0.9):]

Y_train=Y[:int(len(X)*0.9)]
Y_test=Y[int(len(X)*0.9):]
```

Defining a Kmeans model

[24]:
```
# Run local implementation of kmeans
model = KMeans(n_clusters=6, max_iter=100, init='random',n_init=10)
```

[25]:
```
model.fit(X_train)
```

[25]:
```
KMeans(init='random', max_iter=100, n_clusters=6)
```

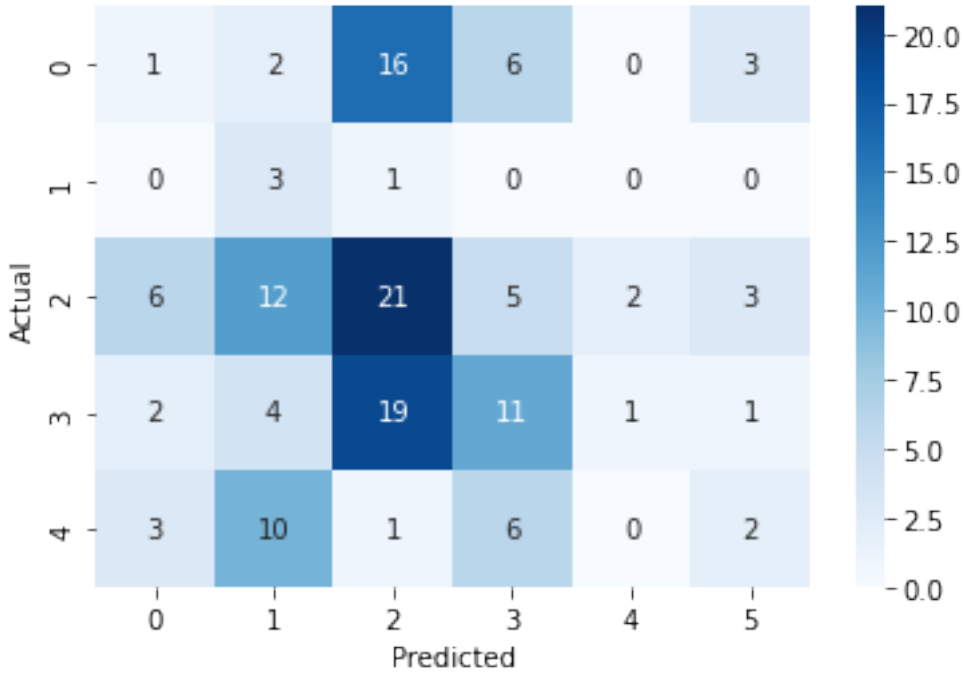
[26]:
```
#Obtaining clusters centroid
centroids = model.cluster_centers_

#To obtain the labels of each cluster
labels = model.labels_
```

Train values

[27]:
```
confusion_matrix = pd.crosstab(Y_train, labels, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```

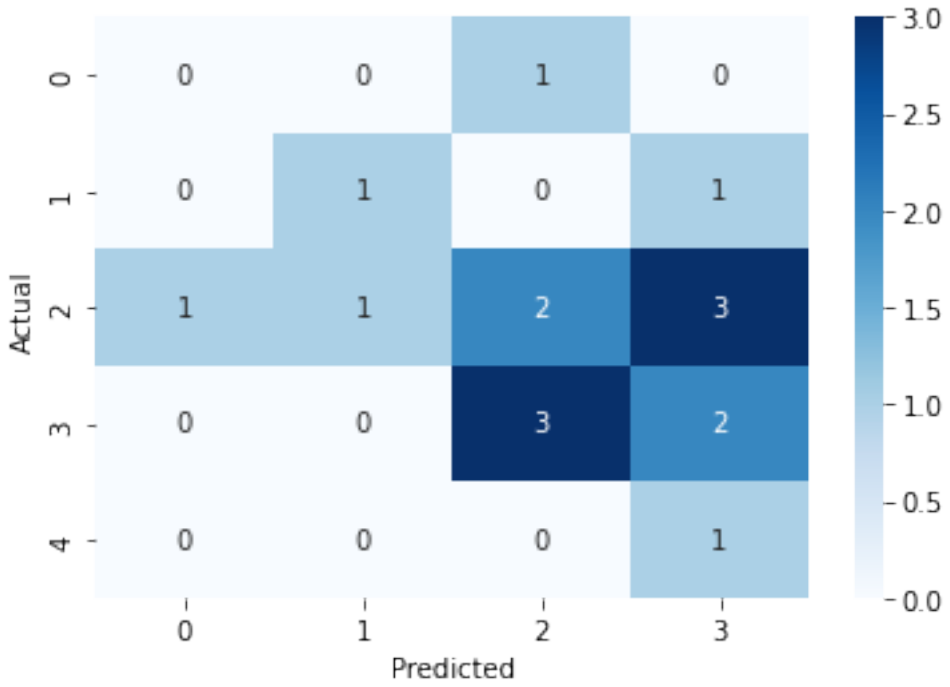


Test values

[28]:
```
y_pred=model.predict(X_test)
```

[29]:
```
confusion_matrix = pd.crosstab(Y_test, y_pred, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



Observations: * Lobe method confuses N2 with N3 * Moderately defined awake periods * First and last group very similar: One may correspond to epilepsy

0.4 Data_medians_horizontal

This data consists of horizontal bipolar derivations, similar to the double banana, but in a perpendicular direction.

Importing the data

[30]:
```
data=pd.read_csv("Extracted_data/Data_medians_horizontal.csv")
```

Adapting the data

[31]:
```
del data['Unnamed: 0']
```

[32]:
```
# 0: Awake
# 1: N1
# 2: N2
# 3: N3
# 4: REM

for i in range(0,len(data)):
    if data.iloc[i,-1]=='Awake':
        data.iloc[i,-1]=0
    elif data.iloc[i,-1]=='N1':
```

```
        data.iloc[i,-1]=1
    elif data.iloc[i,-1]=='N2':
        data.iloc[i,-1]=2
    elif data.iloc[i,-1]=='N3':
        data.iloc[i,-1]=3
    elif data.iloc[i,-1]=='REM':
        data.iloc[i,-1]=4
    else:
        data.iloc[i,-1]=np.nan
```

[33]:
```
data_2=data.dropna()
```

[34]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_2.iloc[:,:35])

Y=data_2.iloc[:,-1].to_numpy()

X_train=X[:int(len(X)*0.9)]
X_test=X[int(len(X)*0.9):]

Y_train=Y[:int(len(X)*0.9)]
Y_test=Y[int(len(X)*0.9):]
```

Defining a Kmeans model

[35]:
```
# Run local implementation of kmeans
model = KMeans(n_clusters=6, max_iter=100, init='random',n_init=10)
```

[36]:
```
model.fit(X_train)
```

[36]:
```
KMeans(init='random', max_iter=100, n_clusters=6)
```

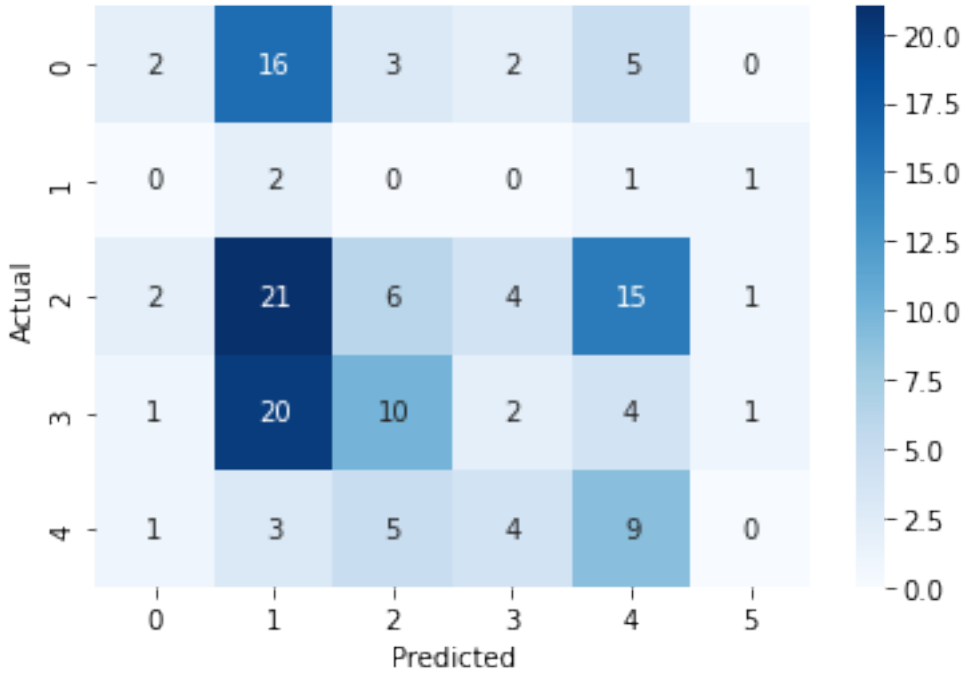
[37]:
```
#Obtaining clusters centroid
centroids = model.cluster_centers_

#To obtain the labels of each cluster
labels = model.labels_
```

Train values

[38]:
```
confusion_matrix = pd.crosstab(Y_train, labels, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```

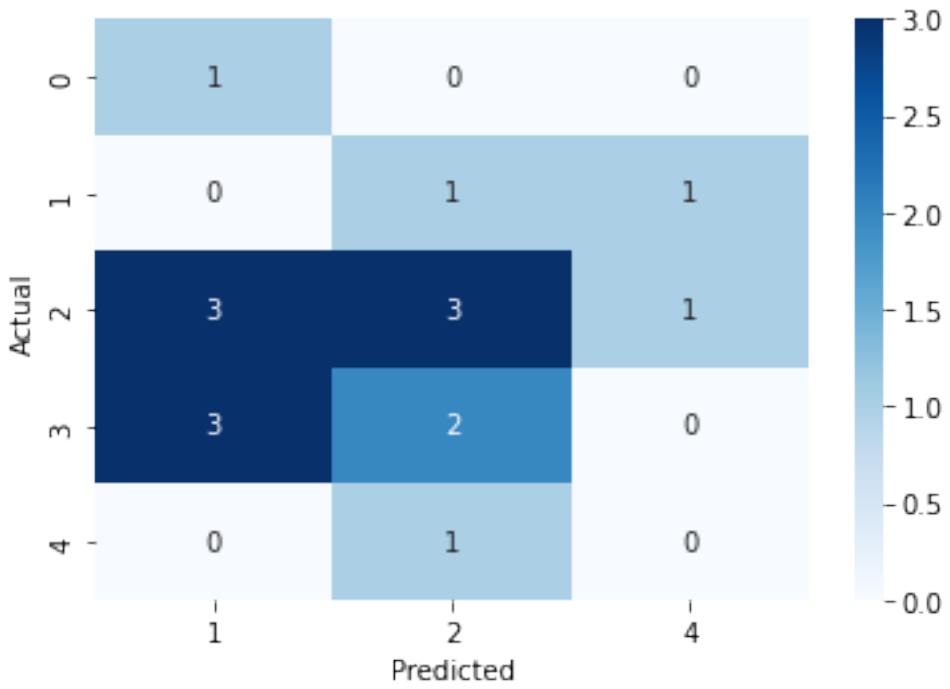


Test values

[39]:
```
y_pred=model.predict(X_test)
```

[40]:
```
confusion_matrix = pd.crosstab(Y_test, y_pred, rownames=['Actual'],␣
↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



Observations: * Two groups correspond to N2 and N3. Possibly one corresponds to segments with attacks * No group is particularly defined

0.5 Data_medians_vertical

Median values of the vertical bipolar derivations. Very similar to the first test but instead of using the standard derivation among the values, we will use the mean value of medians.

Importing the data

[41]:
```
data=pd.read_csv("Extracted_data/Data_medians_vertical.csv")
```

Adapting the data

[42]:
```
del data['Unnamed: 0']
```

[43]:
```
# 0: Awake
# 1: N1
# 2: N2
# 3: N3
# 4: REM

for i in range(0,len(data)):
    if data.iloc[i,-1]=='Awake':
        data.iloc[i,-1]=0
    elif data.iloc[i,-1]=='N1':
```

```
        data.iloc[i,-1]=1
    elif data.iloc[i,-1]=='N2':
        data.iloc[i,-1]=2
    elif data.iloc[i,-1]=='N3':
        data.iloc[i,-1]=3
    elif data.iloc[i,-1]=='REM':
        data.iloc[i,-1]=4
    else:
        data.iloc[i,-1]=np.nan
```

[44]:
```
data_2=data.dropna()
```

[45]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_2.iloc[:,:35])

Y=data_2.iloc[:,-1].to_numpy()

X_train=X[:int(len(X)*0.9)]
X_test=X[int(len(X)*0.9):]

Y_train=Y[:int(len(X)*0.9)]
Y_test=Y[int(len(X)*0.9):]
```

Defining a Kmeans model

[46]:
```
# Run local implementation of kmeans
model = KMeans(n_clusters=6, max_iter=100, init='random',n_init=10)
```

[47]:
```
model.fit(X_train)
```

[47]:
```
KMeans(init='random', max_iter=100, n_clusters=6)
```

[48]:
```
data_2=data.dropna()
```

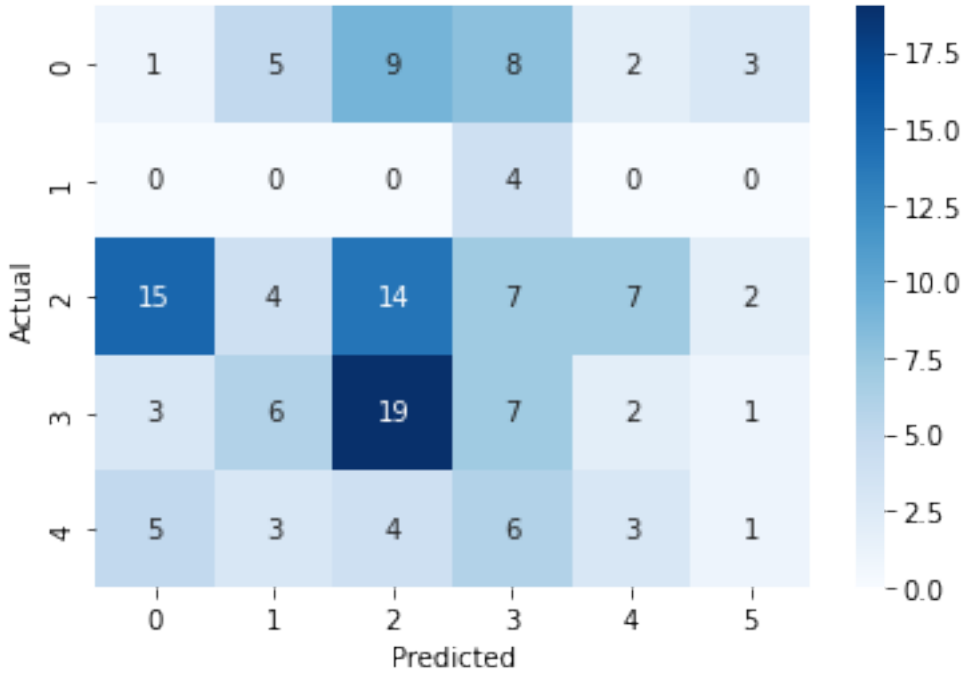
Train values

[49]:
```
#Obtaining clusters centroid
centroids = model.cluster_centers_

#To obtain the labels of each cluster
labels = model.labels_
```

[50]:
```
confusion_matrix = pd.crosstab(Y_train, labels, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```

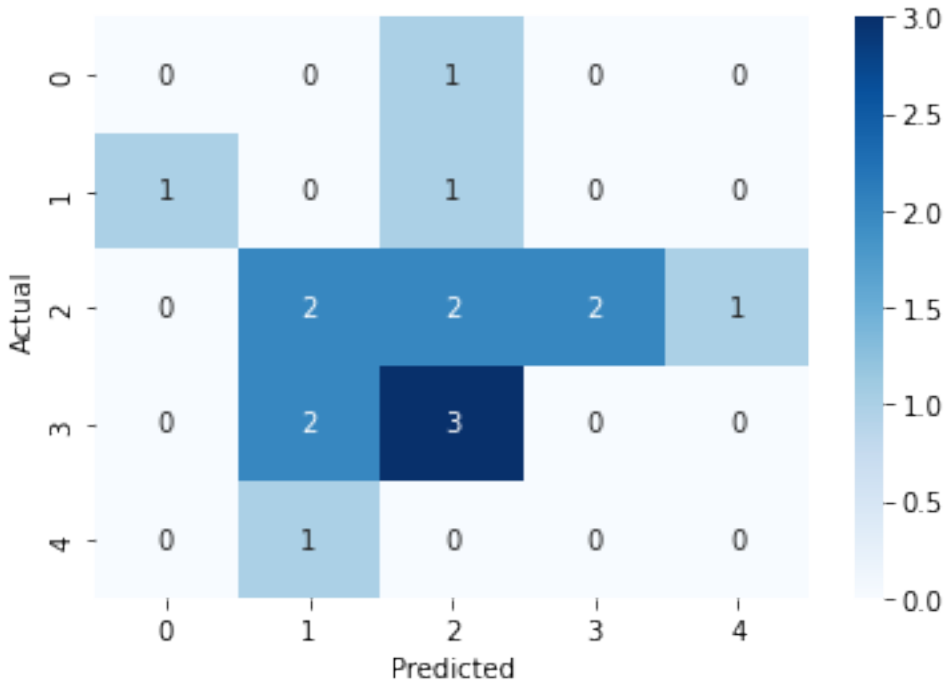


Test values

[51]:
```
y_pred=model.predict(X_test)
```

[52]:
```
confusion_matrix = pd.crosstab(Y_test, y_pred, rownames=['Actual'],␣
↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



0.6 Data_lobes_std

Single electrode recordings with the electrodes are classified into lobes. The value of interest here is the standard derivation in a single class of electrodes. Let's remember that the median value wasn't very accurate in the correct classification of N2 and N3 fragments. We hope that the electrophysiological properties of each stage are more distinguishable by using standard derivation.

Importing the data

[53]:
```
data=pd.read_csv("Extracted_data/Data_medians_3.csv")
```

Adapting the data

[54]:
```
del data['Unnamed: 0']
```

[55]:
```
# 0: Awake
# 1: N1
# 2: N2
# 3: N3
# 4: REM

for i in range(0,len(data)):
    if data.iloc[i,-1]=='Awake':
        data.iloc[i,-1]=0
    elif data.iloc[i,-1]=='N1':
```

```
        data.iloc[i,-1]=1
    elif data.iloc[i,-1]=='N2':
        data.iloc[i,-1]=2
    elif data.iloc[i,-1]=='N3':
        data.iloc[i,-1]=3
    elif data.iloc[i,-1]=='REM':
        data.iloc[i,-1]=4
    else:
        data.iloc[i,-1]=np.nan
```

[56]:
```
data_2=data.dropna()
```

[57]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_2.iloc[:,:25])

Y=data_2.iloc[:,-1].to_numpy()

X_train=X[:int(len(X)*0.9)]
X_test=X[int(len(X)*0.9):]

Y_train=Y[:int(len(X)*0.9)]
Y_test=Y[int(len(X)*0.9):]
```

Defining a Kmeans model

[58]:
```
# Run local implementation of kmeans
model = KMeans(n_clusters=6, max_iter=100, init='random',n_init=10)
```

[59]:
```
model.fit(X_train)
```

[59]:
```
KMeans(init='random', max_iter=100, n_clusters=6)
```

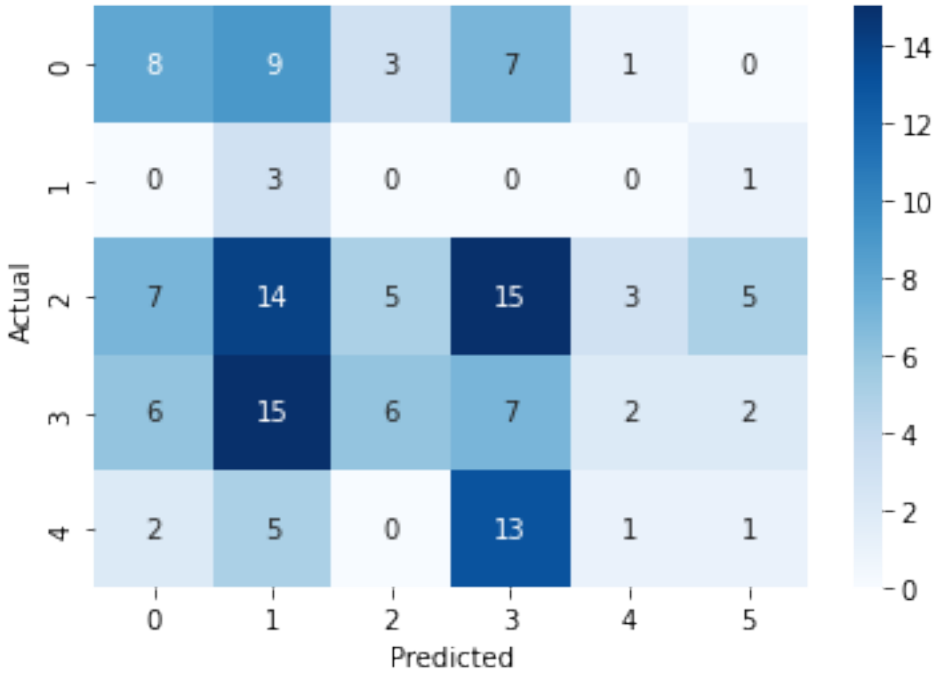
Train values

[60]:
```
#Obtaining clusters centroid
centroids = model.cluster_centers_

#To obtain the labels of each cluster
labels = model.labels_
```

[61]:
```
confusion_matrix = pd.crosstab(Y_train, labels, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```

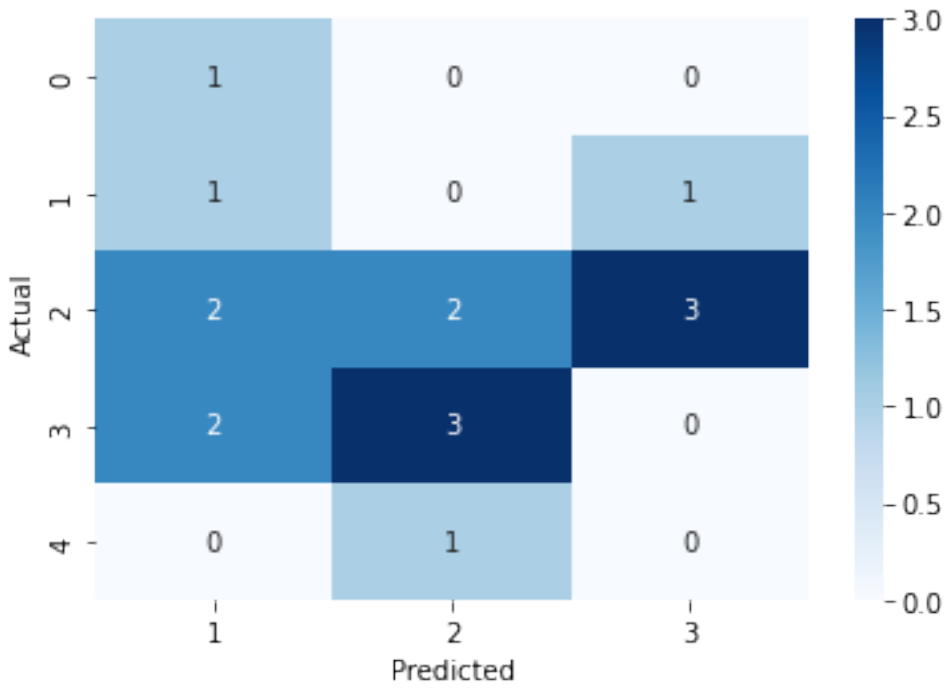


Test values

[62]:
```
y_pred=model.predict(X_test)
```

[63]:
```
confusion_matrix = pd.crosstab(Y_test, y_pred, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



[64]:
```
np.std(centroids,axis=0)
```

[64]:
```
array([1.10639682, 1.15550397, 1.19108799, 1.1278308 , 1.13799211,
       1.20562992, 1.22571886, 1.30946676, 1.29401192, 1.2754787 ,
       1.29055034, 1.35164439, 1.11808464, 1.15387115, 1.14641472,
       1.20852432, 1.26136467, 1.22864774, 0.18717068])
```

Observations: * Even though the correct classification of N2 and N3 fragments improved, the REM stage accuracy greatly decreased. * Awake periods also experienced a decreased accuracy.

0.7 Data_lobes_mean

We already tried the median value and the standard deviation for the lobe analysis. Each had its strengths and weaknesses. We hope the mean value can provide information both about the skewness of the PSD and the main frequency.

Importing the data

[65]:
```
data=pd.read_csv("Extracted_data/Data_medians_lobe_STD.csv")
```

Adapting the data

[66]:
```
del data['Unnamed: 0']
```

[67]:
```
# 0: Awake
# 1: N1
# 2: N2
# 3: N3
# 4: REM

for i in range(0,len(data)):
    if data.iloc[i,-1]=='Awake':
        data.iloc[i,-1]=0
    elif data.iloc[i,-1]=='N1':
        data.iloc[i,-1]=1
    elif data.iloc[i,-1]=='N2':
        data.iloc[i,-1]=2
    elif data.iloc[i,-1]=='N3':
        data.iloc[i,-1]=3
    elif data.iloc[i,-1]=='REM':
        data.iloc[i,-1]=4
    else:
        data.iloc[i,-1]=np.nan
```

[68]:
```
data_2=data.dropna()
```

[69]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_2.iloc[:,:25])

Y=data_2.iloc[:,-1].to_numpy()

X_train=X[:int(len(X)*0.9)]
X_test=X[int(len(X)*0.9):]

Y_train=Y[:int(len(X)*0.9)]
Y_validate=Y[int(len(X)*0.9):]
```

Defining a Kmeans model

[70]:
```
# Run local implementation of kmeans
model = KMeans(n_clusters=6, max_iter=100, init='random',n_init=10)
```

[71]:
```
model.fit(X_train)
```

[71]:
```
KMeans(init='random', max_iter=100, n_clusters=6)
```

[72]:
```
#Obtaining clusters centroid
centroids = model.cluster_centers_

#To obtain the labels of each cluster
labels = model.labels_
```

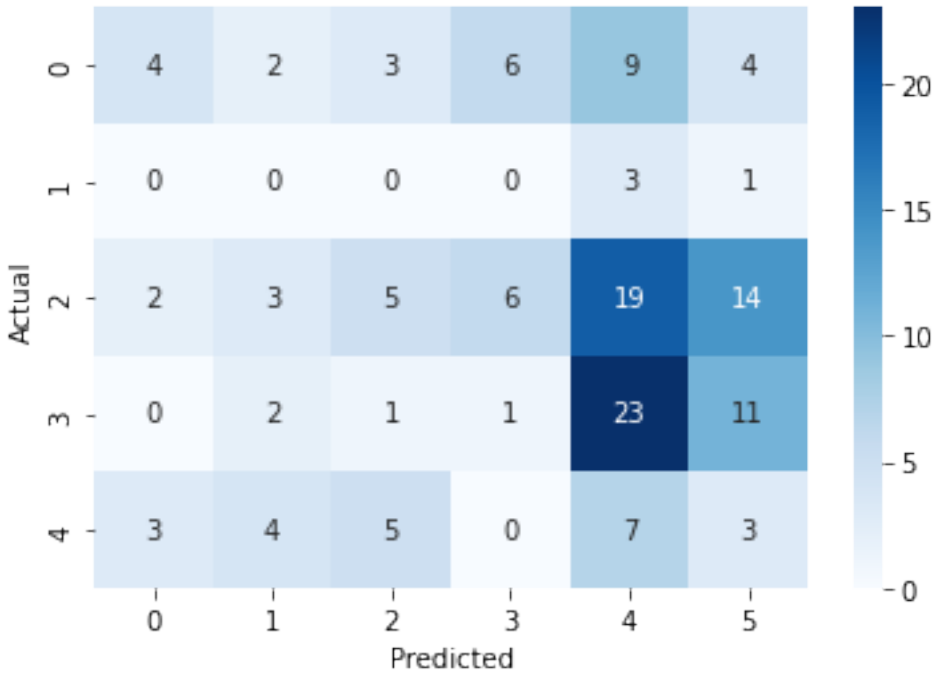
Train values

[73]:
```
#Obtaining clusters centroid
centroids = model.cluster_centers_

#To obtain the labels of each cluster
labels = model.labels_
```

[74]:
```
confusion_matrix = pd.crosstab(Y_train, labels, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```

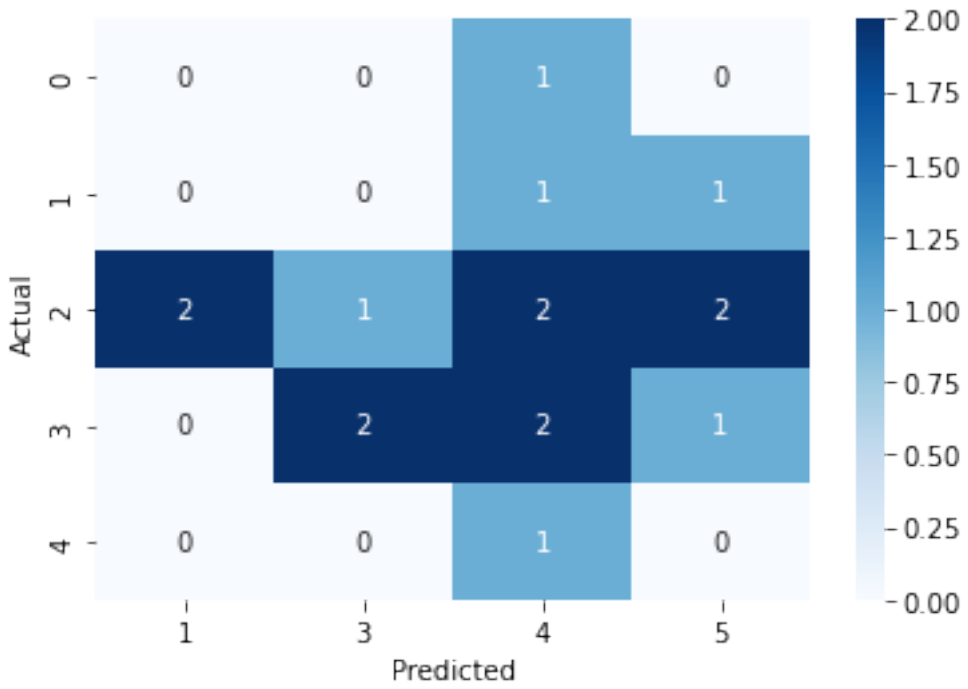


Test values

[75]:
```
y_pred=model.predict(X_test)
```

[76]:
```
confusion_matrix = pd.crosstab(Y_test, y_pred, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



Observations: * Somehow we have worse results than with the median and with the STD. * Maybe the best parameter group includes both the median and the STD, but with a generalized version of parameters.

0.8 Data_lobes_std+medians

Now we will try to use the two best parameter groups and hope the algorithm won't be over-adjusted to the data.

Importing the data

[78]:
```
data=pd.read_csv("Extracted_data/Data_medians_4.csv")
```

Adapting the data

[79]:
```
del data['Unnamed: 0']
```

[80]:
```
del data['38']
```

[81]:
```
# 0: Awake
# 1: N1
# 2: N2
# 3: N3
# 4: REM

for i in range(0,len(data)):
```

```
    if data.iloc[i,-1]=='Awake':
        data.iloc[i,-1]=0
    elif data.iloc[i,-1]=='N1':
        data.iloc[i,-1]=1
    elif data.iloc[i,-1]=='N2':
        data.iloc[i,-1]=2
    elif data.iloc[i,-1]=='N3':
        data.iloc[i,-1]=3
    elif data.iloc[i,-1]=='REM':
        data.iloc[i,-1]=4
    else:
        data.iloc[i,-1]=np.nan
```

[82]:
```
data_2=data.dropna()
```

[83]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_2.iloc[:,:17])

Y=data_2.iloc[:,-1].to_numpy()

X_train=X[:int(len(X)*0.85)]
X_test=X[int(len(X)*0.85):]

Y_train=Y[:int(len(X)*0.85)]
Y_test=Y[int(len(X)*0.85):]
```

Defining a Kmeans model

[84]:
```
# Run local implementation of kmeans
model = KMeans(n_clusters=6, max_iter=100, init='random',n_init=10)
```

[85]:
```
model.fit(X_train)
```

[85]:
```
KMeans(init='random', max_iter=100, n_clusters=6)
```

[86]:
```
#Obtaining clusters centroid
centroids = model.cluster_centers_

#To obtain the labels of each cluster
labels = model.labels_
```

Train values

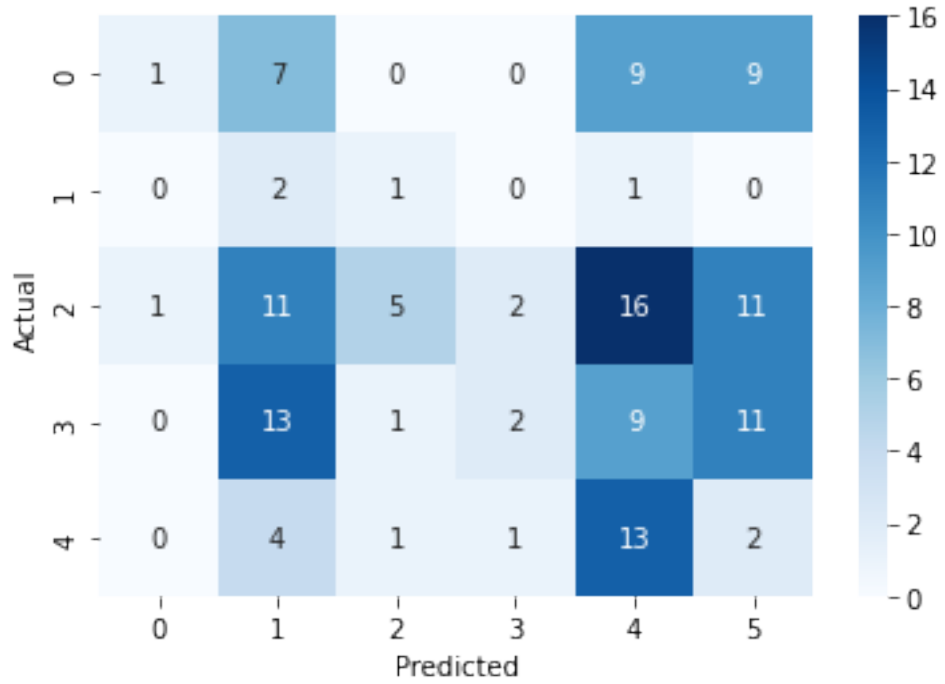
[87]:
```
y_pred=model.predict(X_train)
```

[88]:
```
np.shape(Y)
```

[88]: (157,)

[89]:
```
confusion_matrix = pd.crosstab(Y_train, y_pred, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



[90]:
```
np.shape(centroids)
```

[90]: (6, 17)

[91]:
```
np.std(centroids,axis=0)
```

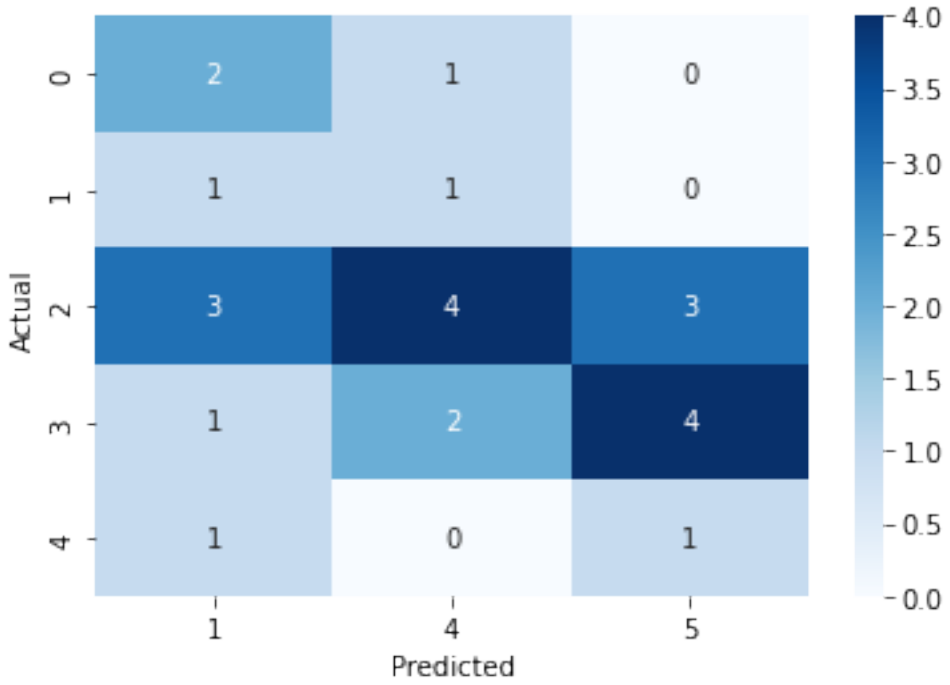
[91]:
```
array([1.31508141, 1.44789848, 1.4518577 , 1.50542897, 1.74182931,
       1.68103272, 1.43457101, 1.64228555, 1.57416316, 1.62400309,
       1.85653065, 1.81386237, 1.35501158, 1.47113136, 1.41265323,
       1.41818002, 1.65714423])
```

Test values

[92]:
```
y_pred=model.predict(X_test)
```

[93]:
```
confusion_matrix = pd.crosstab(Y_test, y_pred, rownames=['Actual'],␣
↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



0.9 Keras

[94]:
```
from sklearn.neighbors import KNeighborsClassifier
```

The following series of experiments take means and standard deviations of the 6 main areas: Frontal, left temporal, right temporal, central, parietal, and occipital. 10-second segments were taken that were not on the border of a sleep stage. A PSD was obtained for each segment and each electrode. The first three quintiles were calculated for each PSD (ie when the area under the curve corresponded to 20%, 40%, and 60% of the total area). Finally, the average of each quintile was obtained for each area and all the segments of the same sleep phase, as well as the standard deviation.

Finally, the activity and mobility of each stage were calculated. These two values are two of the Hjorth parameters and are calculated as:

Activity $\rightarrow A = \sigma_0^2$

Mobility $\rightarrow M = \frac{\sigma_1}{\sigma_0}$

Where σ_i represents the variance of the ith derivative of the EEG recording. That is, σ_0 is the

variance of the raw EEG values. For the Hjorth parameters, all available electrodes were collapsed.

[128]:
```
data=pd.read_csv("Extracted_data/Data_medians_4.csv")
del data['Unnamed: 0']
del data['38']
```

[129]:
```
for i in range(0,len(data)):
    if data.iloc[i,-1]!='Awake':
        if data.iloc[i,-1]!='N1':
            if data.iloc[i,-1]!='N2':
                if data.iloc[i,-1]!='N3':
                    if data.iloc[i,-1]!='REM':
                        data.iloc[i,-1]=np.nan
```

[130]:
```
data
```

[130]:

	0	1	2	3	4	5	\
0	19.911429	21.826667	19.901111	19.076667	21.918889	14.850000	
1	40.660000	35.816667	38.405556	42.591111	40.958889	37.191667	
2	28.203333	24.951111	25.603333	33.302222	28.286667	24.416667	
3	18.286190	16.575556	17.173333	22.212222	18.586667	15.928333	
4	11.339524	10.327778	11.073333	9.653333	12.070000	10.046667	
..	…	…	…	…	…	…	
160	23.572857	21.060000	22.835556	24.652222	19.970000	18.556667	
161	18.736190	16.664444	17.992222	19.476667	16.011111	14.800000	
162	14.926190	13.685556	14.278889	15.434444	12.957778	11.726667	
163	10.816190	9.702222	10.213333	10.985556	9.072222	8.145000	
164	4.886190	4.464444	4.621111	4.972222	4.191111	3.780000	

	6	7	8	9	…	29	\
0	58.638571	60.644444	61.956667	57.580000	…	210.934721	
1	148.707143	114.740000	116.255556	180.642222	…	601.949326	
2	104.157619	78.117778	77.450000	141.416667	…	395.701329	
3	72.619048	55.015556	55.386667	98.113333	…	258.530174	
4	42.745238	33.640000	34.700000	37.300000	…	156.242749	
..	…	…	…	…	…	…	
160	62.023810	55.311111	62.484444	65.027778	…	233.876713	
161	47.885238	42.373333	47.960000	49.993333	…	179.285611	
162	37.418571	33.524444	37.350000	38.766667	…	137.755238	
163	27.838095	24.458889	27.936667	28.534444	…	99.599288	
164	12.803333	12.212222	13.241111	14.298889	…	41.261791	

	30	31	32	33	34	\
0	627.235043	648.175431	626.022091	643.377231	540.470154	
1	2122.510839	1710.063114	1684.448166	3062.885569	2100.222584	
2	1433.984345	1147.643210	1131.199674	2268.983257	1375.903119	
3	951.861362	774.702891	776.389821	1493.072896	943.001124	

```
4     587.817203   465.337699   467.104798   546.078070   612.918988
..           …            …            …            …            …
160   956.606279   901.840306  1079.626406  1000.445620   627.724242
161   731.916047   694.028468   835.901439   759.237127   474.306587
162   558.178258   505.701924   648.375509   566.989604   363.330987
163   449.385031   411.347601   560.246779   459.417619   270.892424
164   165.142991   188.748088   225.965814   206.167366   122.058242

               35           36        37     39
0      495.039453  4.422272e+07  0.001152  Awake
1     1750.616297  2.508360e+06  0.002201     N2
2     1170.567469  1.469640e+06  0.001673    REM
3      787.208156  5.040967e+06  0.002785     N2
4      465.555425  9.224363e+06  0.001658     N3
..            …           …       …    …
160    663.898963  1.033058e+05  0.000996    REM
161    509.926345  3.701008e+04  0.000528     N1
162    390.608436  9.521701e+04  0.000363     N2
163    296.452757  6.754039e+06  0.000063     N3
164    109.744217  2.577399e+06  0.000092     N2

[165 rows x 39 columns]
```

[131]:
```
data_2=data.dropna()
```

[132]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_2.iloc[:,:35])

Y=data_2.iloc[:,-1].to_numpy()

X_train=X[:int(len(X)*0.75)]
X_test=X[int(len(X)*0.75):]

Y_train=Y[:int(len(X)*0.75)]
Y_test=Y[int(len(X)*0.75):]
```

[102]:
```
k_neighbor=KNeighborsClassifier(5)
```

[103]:
```
print(np.shape(X_train))
print(np.shape(Y_train))
```

```
(117, 35)
(117,)
```

[104]:
```
Y_train
```

[104]:
```
array(['Awake', 'N2', 'REM', 'N2', 'N3', 'REM', 'N3', 'REM', 'N2', 'N3',
       'REM', 'Awake', 'N2', 'N2', 'N3', 'REM', 'Awake', 'N2', 'N2', 'N3',
       'N3', 'Awake', 'REM', 'N2', 'Awake', 'N2', 'N3', 'REM', 'N3',
       'REM', 'Awake', 'REM', 'Awake', 'Awake', 'N2', 'Awake', 'N3',
       'Awake', 'N2', 'N3', 'REM', 'N2', 'N3', 'N2', 'N1', 'N2', 'N3',
       'N2', 'N1', 'N3', 'N3', 'N2', 'N2', 'REM', 'N2', 'Awake', 'N3',
       'N2', 'N3', 'N2', 'Awake', 'N2', 'N3', 'Awake', 'N2', 'Awake',
       'REM', 'N2', 'Awake', 'N2', 'N3', 'N2', 'N3', 'N2', 'N2', 'N3',
       'Awake', 'N1', 'Awake', 'Awake', 'N2', 'N3', 'REM', 'N2', 'N3',
       'N2', 'Awake', 'N3', 'N2', 'N3', 'N2', 'N3', 'REM', 'N2', 'N2',
       'N3', 'N2', 'N2', 'REM', 'N3', 'REM', 'N2', 'Awake', 'N2', 'N3',
       'N2', 'N3', 'N2', 'N3', 'REM', 'Awake', 'N2', 'N3', 'REM', 'N3',
       'REM', 'N2'], dtype=object)
```

[105]:
```
k_neighbor.fit(X_train,Y_train)
```

[105]:
```
KNeighborsClassifier()
```

[106]:
```
y_pred=k_neighbor.predict(X_train)
```

[107]:
```
confusion_matrix = pd.crosstab(Y_train, y_pred, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```

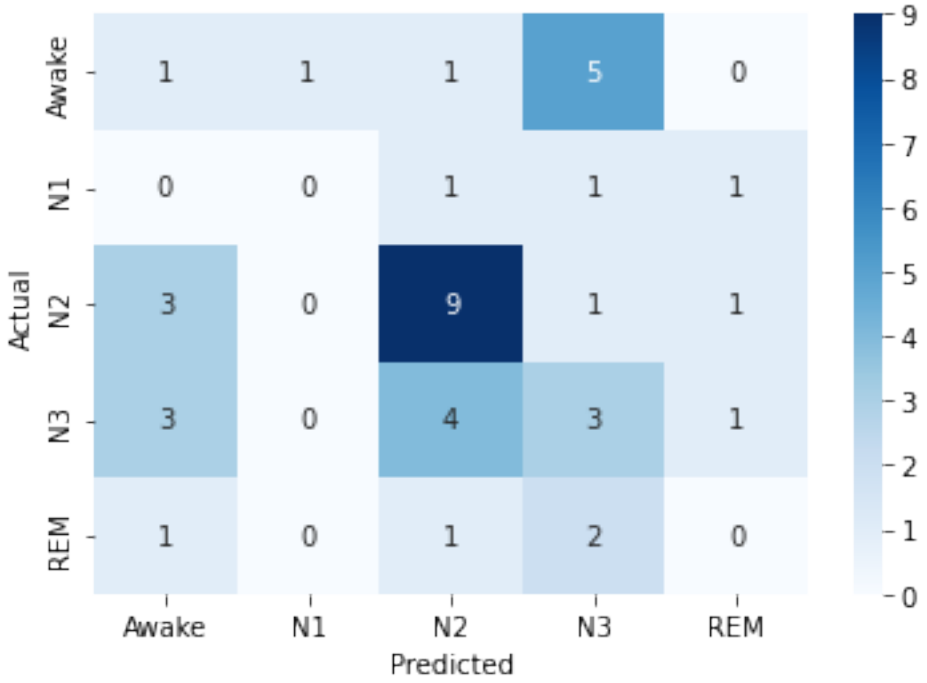
[108]:
```
sum(confusion_matrix.iloc[i,i] for i in range(0,len(confusion_matrix)))/
↪len(Y_train)
```

[108]: 0.4188034188034188

[109]:
```
y_pred=k_neighbor.predict(X_test)
```

[110]:
```
confusion_matrix = pd.crosstab(Y_test, y_pred, rownames=['Actual'],␣
↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



[111]:
```
sum(confusion_matrix.iloc[i,i] for i in range(0,len(confusion_matrix)))/
↪len(Y_test)
```

[111]: 0.325

0.10 Comparison between sections

From the last lesson, we learned that epileptic attacks occur mostly in the N2 and N3 sleep stages. So now we will try to split the sub-segments of each stage into two clusters. We expect that, with

the correct set of parameters, one of the clusters will correspond to epileptic seizures and the other will correspond to normal recordings.

Please note that this classification can't be evaluated and validated because there are no markings that allow us to compare the results to a gold standard.

0.10.1 N2

[112]:
```
data_c=data
```

[113]:
```
for i in range(0,len(data)):
    if data.iloc[i,-1]!="N2":
        data_c.iloc[i,-1]=np.nan
```

[114]:
```
data_c=data_c.dropna()
```

[115]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_c.iloc[:,:37])

X_train=X[:int(len(X)*0.75)]
```

[116]:
```
# Run local implementation of kmeans
model = KMeans(n_clusters=2, max_iter=100, init='random',n_init=10)
```

[117]:
```
model.fit(X)
```

[117]:
```
KMeans(init='random', max_iter=100, n_clusters=2)
```

[118]:
```
#Obtaining clusters centroid
centroids = model.cluster_centers_
```

[119]:
```
centroids
```

[119]:
```
array([[-0.61783467, -0.62070957, -0.61891999, -0.62231813, -0.59319319,
        -0.58682692, -0.58719187, -0.59447495, -0.57491152, -0.58687597,
        -0.55478317, -0.55559992, -0.54025856, -0.6013481 , -0.56503859,
        -0.55747612, -0.56713299, -0.56355206, -0.61688024, -0.62213048,
        -0.62049631, -0.6234374 , -0.59625971, -0.59148912, -0.58258563,
        -0.59645084, -0.57565797, -0.58404583, -0.5579193 , -0.56066686,
        -0.52861544, -0.60169546, -0.55909407, -0.55051283, -0.56889795,
        -0.56726235,  0.03406   ],
       [ 1.11210241,  1.11727723,  1.11405598,  1.12017263,  1.06774773,
         1.05628846,  1.05694536,  1.07005492,  1.03484073,  1.05637674,
         0.99860971,  1.00007986,  0.9724654 ,  1.08242657,  1.01706945,
         1.00345702,  1.02083938,  1.0143937 ,  1.11038443,  1.11983486,
         1.11689335,  1.12218731,  1.07326748,  1.06468042,  1.04865413,
         1.07361151,  1.03618435,  1.0512825 ,  1.00425474,  1.00920034,
         0.95150778,  1.08305183,  1.00636933,  0.99092309,  1.02401631,
```

```
        1.02107224, -0.06130801]])
```

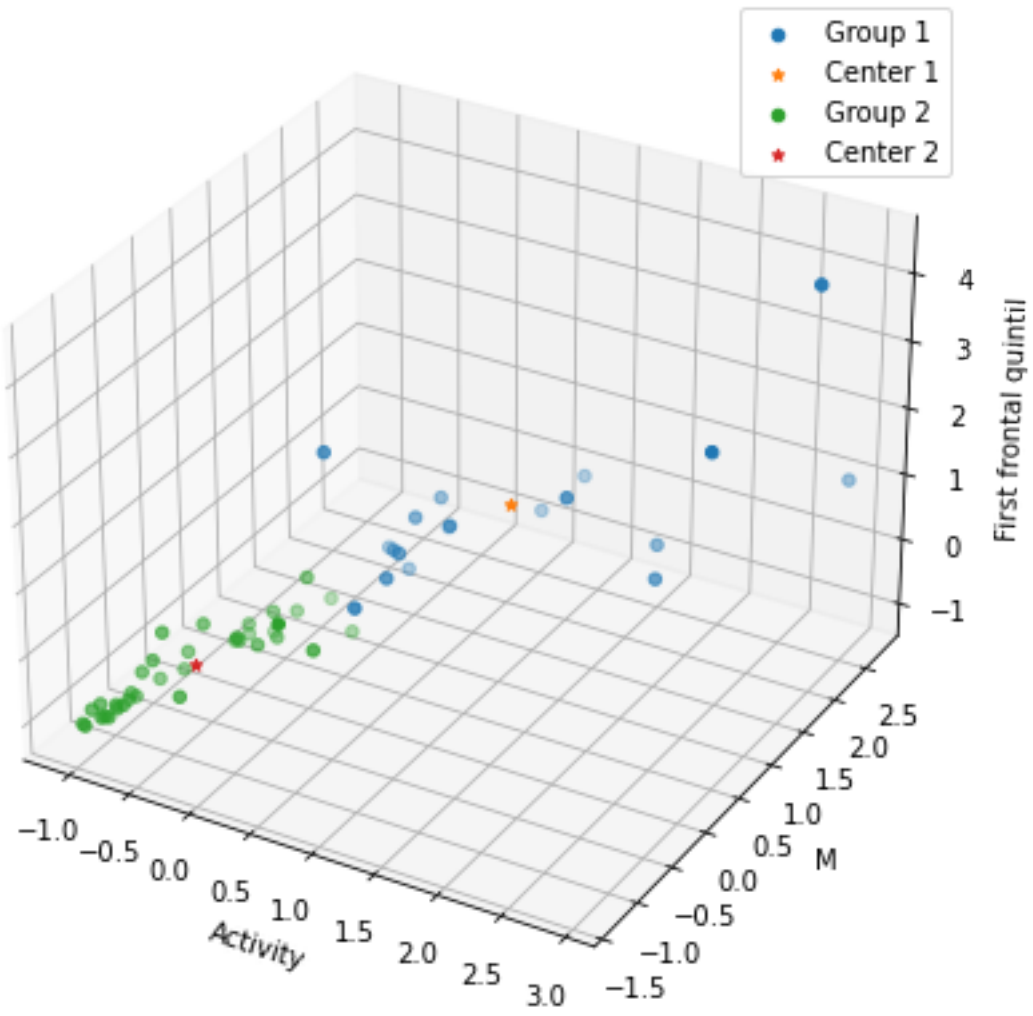
[120]:
```
centroids[0]-centroids[1]
```

[120]:
```
array([-1.72993707, -1.7379868 , -1.73297596, -1.74249076, -1.66094092,
       -1.64311539, -1.64413722, -1.66452987, -1.60975225, -1.64325271,
       -1.55339288, -1.55567978, -1.51272396, -1.68377467, -1.58210804,
       -1.56093315, -1.58797237, -1.57794576, -1.72726467, -1.74196534,
       -1.73738966, -1.74562471, -1.66952719, -1.65616955, -1.63123976,
       -1.67006235, -1.61184232, -1.63532833, -1.56217404, -1.5698672 ,
       -1.48012322, -1.68474729, -1.56546341, -1.54143592, -1.59291426,
       -1.58833459,  0.09536801])
```

[121]:
```
#To obtain the labels of each cluster
labels = model.labels_
```

[122]:
```
# Creating figure
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")
x=-2
y=1
z=30
ax.scatter3D(X[labels>0][:,x],X[labels>0][:,y],X[labels>0][:,z],label="Group 1")
ax.scatter3D(centroids[1,x],centroids[1,y],centroids[1,0],label="Center␣
 ↪1",marker='*')
ax.scatter3D(X[labels<1][:,x],X[labels<1][:,y],X[labels<1][:,z],label="Group 2")
ax.scatter3D(centroids[0,x],centroids[0,y],centroids[0,z],label="Center␣
 ↪2",marker='*')
ax.set_ylabel("M")
ax.set_xlabel("Activity")
ax.set_zlabel("First frontal quintil")
ax.legend()
```

[122]:
```
<matplotlib.legend.Legend at 0x7fa2bda879d0>
```



Observations:

- The first group has the mayority of points and is more compact than the second group.
- The sub-segments that are classified in the second group have more extreme parameters. This might be due to the precence of an artifact or because it corresponds to a epileptic seizure.
- Based on the positions of the centroids we can infer that, if the clasification is correct, the first group corresponds to epileptic seizures and the second group to normal recodings.
- Every single parameter is higher in the first group except the movility.

0.10.2 N3

[133]:
```
data_c=data
```

[134]:
```
for i in range(0,len(data)):
    if data.iloc[i,-1]!="N3":
        data_c.iloc[i,-1]=np.nan
```

[135]:
```
data_c=data_c.dropna()
```

[136]:
```
data_c
```

[136]:

	0	1	2	3	4	5	\
4	11.339524	10.327778	11.073333	9.653333	12.070000	10.046667	
6	26.998571	26.215556	29.127778	29.162222	28.964444	28.295000	
9	10.481905	10.418889	11.724444	11.373333	10.986667	10.811667	
14	34.483810	39.271111	39.704444	42.848889	52.626667	48.441667	
20	12.452381	13.240000	13.191111	12.627778	13.446667	12.915000	
22	19.874286	17.511111	15.896667	18.567778	17.408889	16.461667	
29	31.032857	29.833333	29.096667	26.802222	27.352222	28.403333	
32	20.797619	19.597778	19.213333	17.788889	17.948889	18.641667	
41	17.056190	15.736667	17.441111	18.685556	18.053333	16.200000	
44	22.938571	20.206667	24.743333	23.447778	24.880000	20.580000	
47	13.509524	12.738889	15.117778	14.091111	14.761111	12.363333	
52	27.002857	30.531111	27.175556	30.455556	23.521111	30.885000	
55	6.249524	6.042222	6.161111	6.868889	5.803333	5.753333	
56	30.269524	27.227778	27.304444	31.934444	29.443333	27.925000	
63	22.354286	22.023333	19.170000	21.132222	20.140000	20.373333	
65	12.850476	12.911111	11.314444	12.953333	12.010000	12.263333	
69	21.699048	16.857778	16.901111	14.498889	19.883333	12.946667	
77	3.702857	3.464444	3.512222	3.314444	3.535556	3.431667	
79	26.012857	25.021111	25.410000	25.485556	24.295556	20.575000	
82	26.410952	23.981111	25.848889	26.752222	24.001111	23.941667	
88	14.148571	12.938889	11.632222	13.528889	12.736667	12.380000	
91	31.956667	27.707778	18.122222	29.747778	27.568889	23.048333	
94	22.989524	20.058889	14.544444	20.261111	18.684444	15.398333	
96	8.130476	7.284444	6.515556	7.003333	6.388889	5.288333	
98	11.078571	13.480000	9.510000	12.590000	10.761111	8.158333	
102	48.024286	45.288889	56.105556	38.996667	42.208889	46.661667	
106	16.691905	17.795556	17.750000	13.072222	15.040000	13.595000	
111	39.234762	38.838889	38.456667	38.068889	36.417778	33.683333	
113	26.122381	24.840000	25.090000	25.616667	24.204444	21.866667	
115	30.772857	37.638889	40.477778	30.845556	33.401111	35.395000	
119	20.350000	18.946667	27.378889	18.882222	21.194444	15.316667	
121	7.587619	7.875556	10.221111	6.840000	7.737778	5.386667	
126	17.987619	15.767778	16.215556	18.902222	16.343333	15.956667	
130	4.200000	3.641111	3.693333	4.238889	3.674444	3.561667	
132	16.167619	28.574444	30.382222	23.164444	25.064444	23.246667	
137	2.534286	2.466667	2.526667	2.556667	2.502222	2.265000	
140	15.134286	15.461111	15.403333	16.684444	15.866667	14.901667	
145	27.841429	29.337778	29.293333	27.997778	25.055556	25.330000	
149	12.059524	12.661111	12.512222	12.027778	10.525556	10.926667	
153	34.179524	33.141111	31.646667	33.260000	31.634444	29.551667	
154	26.902381	27.300000	25.703333	26.954444	25.543333	24.010000	
158	28.656190	26.148889	28.142222	29.854444	25.000000	23.476667	

```
163  10.816190   9.702222  10.213333  10.985556   9.072222   8.145000

              6          7          8          9  …          29  \
4     42.745238  33.640000  34.700000  37.300000  …  156.242749
6     61.641429  60.512222  82.962222  74.073333  …  399.858713
9     25.147143  24.993333  33.655556  28.965556  …  144.186463
14    87.269048 112.805556 105.626667 113.292222  …  660.603977
20    25.688571  26.050000  26.112222  24.443333  …  129.994494
22    44.143333  35.455556  31.470000  38.522222  …  168.509782
29    58.532857  54.425556  53.760000  52.906667  …  279.149177
32    39.318571  36.075556  35.627778  35.066667  …  183.913391
41    38.504286  36.480000  38.413333  39.751111  …  173.948698
44    58.670000  48.557778  63.928889  59.466667  …  262.032101
47    34.662381  30.922222  40.020000  36.181111  …  158.756784
52    53.664286  67.217778  57.476667  63.621111  …  383.438299
55    11.253333  10.975556  11.208889  12.394444  …   48.283386
56    61.611429  56.700000  51.467778  63.308889  …  295.746814
63    55.316667  51.916667  40.483333  48.080000  …  221.670005
65    32.771905  31.150000  24.155556  31.478889  …  133.566442
69    60.640952  49.612222  47.582222  43.042222  …  184.263274
77     7.904286   7.245556   7.262222   6.896667  …   21.421253
79    65.421429  70.426667  62.573333  68.556667  …  266.678385
82    63.301905  59.346667  62.635556  59.471111  …  273.349123
88    28.691905  26.242222  24.961111  27.225556  …  126.354677
91    91.949048  63.400000  50.353333  66.278889  …  244.080074
94    73.354762  48.211111  37.993333  46.555556  …  162.397470
96    25.550476  17.801111  19.468889  15.501111  …   49.980521
98    33.675714  41.008889  29.511111  37.497778  …  106.413721
102  172.341905 156.258889 187.771111 129.700000  …  756.633924
106   44.405714  50.278889  47.530000  29.537778  …  140.196937
111   75.847619  76.986667  76.892222  72.163333  …  313.933238
113   48.101905  45.718889  47.163333  46.223333  …  206.259057
115  102.132381 124.227778 131.256667 109.773333  …  605.024322
119   63.618095  64.472222  85.874444  60.328889  …  244.212924
121   23.472381  26.840000  35.091111  21.463333  …   71.407751
126   32.170000  29.741111  30.374444  36.917778  …  151.865404
130    7.272857   6.743333   6.746667   8.026667  …   28.214914
132   38.160476  72.165556  75.347778  48.813333  …  263.592880
137    4.841429   4.704444   4.878889   4.747778  …   13.632215
140   34.830952  37.225556  37.133333  41.551111  …  179.454228
145   53.556667  58.505556  58.205556  54.065556  …  253.110709
149   25.241905  28.402222  27.533333  25.598889  …  113.256666
153   76.524762  73.398889  66.352222  67.616667  …  285.148926
154   61.178571  67.254444  59.908889  59.874444  …  250.997183
158   71.534286  65.567778  72.303333  75.062222  …  290.153260
163   27.838095  24.458889  27.936667  28.534444  …   99.599288
```

	30	31	32	33	34	\
4	587.817203	465.337699	467.104798	546.078070	612.918988	
6	875.714794	795.695026	1081.203383	976.650182	1007.277740	
9	329.504906	299.966163	400.458743	347.526865	350.097442	
14	810.729764	1178.118628	1073.187894	1112.275571	1286.495199	
20	235.258307	273.743060	243.592546	257.479009	283.038934	
22	466.936157	369.663944	313.986636	401.507249	355.099730	
29	601.866864	564.956831	556.644310	581.754025	529.543763	
32	396.192725	369.603986	365.086165	380.098674	346.435712	
41	390.992855	354.784589	378.417194	408.356892	385.593292	
44	660.175106	555.743070	728.627763	655.188247	701.497471	
47	389.865403	345.386991	449.811277	390.568133	418.003661	
52	662.537763	1230.418855	798.101891	925.404313	537.842633	
55	98.718814	93.877851	96.610335	116.956018	94.046291	
56	636.094697	524.642010	493.004100	610.609312	542.743026	
63	657.426995	563.646951	401.770693	631.536068	495.514555	
65	393.295247	350.744256	242.966376	477.452683	294.922349	
69	727.622711	573.013377	546.437332	516.997233	558.992780	
77	47.885933	44.538181	42.768123	44.231258	48.017281	
79	981.816777	1021.609019	811.503872	1061.325166	649.918134	
82	718.692429	688.950026	721.330118	642.779865	529.542734	
88	327.491416	300.667696	288.439385	309.124707	293.245258	
91	1986.062811	1082.466498	879.061319	1003.909821	766.299146	
94	1703.199243	884.173153	865.625408	752.394480	543.215938	
96	579.800415	370.898427	527.854273	242.218301	161.193851	
98	438.797076	484.642007	377.384311	415.397009	354.127376	
102	3039.386077	2380.556244	2878.895408	1864.050784	1554.563101	
106	744.050453	741.812404	684.449737	414.872184	479.822748	
111	783.234271	785.081986	834.365976	736.142512	695.171512	
113	508.891489	507.931461	540.571387	478.699402	447.418727	
115	1928.838600	1911.900744	1976.685758	1885.518235	1451.778778	
119	807.615376	834.401797	1621.700797	786.025970	754.618554	
121	295.966877	361.806818	743.585816	275.536118	265.440138	
126	308.896578	287.130775	293.200326	444.668751	292.554209	
130	54.663652	49.433609	49.256356	71.999100	50.639364	
132	448.815948	1578.282792	1730.909541	506.564951	566.225446	
137	29.314164	27.933372	28.957759	28.136708	29.234411	
140	456.293709	496.922674	482.631157	540.505234	437.217073	
145	671.441757	742.741410	737.986796	650.834585	573.037639	
149	310.814455	348.890665	340.459252	299.829648	258.393600	
153	1049.807550	1083.384077	932.847137	859.836816	730.518049	
154	794.803224	1074.479630	890.432675	792.040132	675.661065	
158	1118.145036	1082.963101	1274.547841	1167.185522	768.765636	
163	449.385031	411.347601	560.246779	459.417619	270.892424	

	35	36	37	39
4	465.555425	9.224363e+06	0.001658	N3

```
6    1048.574140  2.324112e+07  0.000369  N3
9     367.198218  3.268244e+07  0.000392  N3
14   1287.632856  4.220465e+07  0.001066  N3
20    254.441075  1.402703e+08  0.000571  N3
22    326.303332  3.574233e+08  0.000113  N3
29    548.070935  2.867187e+08  0.000065  N3
32    360.993817  2.020305e+08  0.000056  N3
41    350.101424  3.644143e+08  0.000212  N3
44    598.048333  9.533055e+07  0.000159  N3
47    363.520168  6.683818e+07  0.000195  N3
52   1396.628739  1.886213e+07  0.000228  N3
55     92.639052  1.967117e+07  0.000181  N3
56    533.690439  9.170754e+07  0.000226  N3
63    447.726223  1.532783e+09  0.000261  N3
65    291.012161  3.030377e+08  0.000256  N3
69    427.396602  9.360937e+06  0.000407  N3
77     43.032654  5.721059e+07  0.000639  N3
79    565.242040  3.889402e+07  0.000400  N3
82    546.895263  4.879537e+07  0.000291  N3
88    288.084486  1.020876e+07  0.000219  N3
91    617.692599  1.250906e+07  0.000206  N3
94    427.940223  2.291153e+07  0.000121  N3
96    129.802689  1.167292e+07  0.000173  N3
98    286.121791  2.376138e+07  0.000372  N3
102  1766.158443  4.743891e+05  0.000940  N3
106   307.205007  7.949271e+06  0.000346  N3
111   657.977234  5.047649e+06  0.003162  N3
113   424.540621  2.048876e+08  0.000277  N3
115  1591.044722  5.051843e+06  0.000230  N3
119   663.226873  2.203814e+06  0.000565  N3
121   216.778679  2.882108e+06  0.000623  N3
126   302.807136  1.635488e+08  0.000204  N3
130    51.094968  4.744499e+08  0.000126  N3
132   555.668905  4.273410e+07  0.000158  N3
137    25.364101  3.148235e+07  0.000190  N3
140   459.117927  2.925963e+07  0.000163  N3
145   577.772239  3.294283e+07  0.000260  N3
149   266.383254  4.062186e+07  0.000196  N3
153   677.864444  1.908625e+07  0.000203  N3
154   623.961387  2.925272e+07  0.000156  N3
158   805.499974  1.381545e+07  0.000045  N3
163   296.452757  6.754039e+06  0.000063  N3

[43 rows x 39 columns]
```

[137]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_c.iloc[:,:37])
```

```
X_train=X[:int(len(X)*0.75)]
```

[138]:
```
# Run local implementation of kmeans
model = KMeans(n_clusters=2, max_iter=100, init='random',n_init=10)
```

[139]:
```
model.fit(X)
```

[139]:
```
KMeans(init='random', max_iter=100, n_clusters=2)
```

[140]:
```
#Obtaining clusters centroid
centroids = model.cluster_centers_
```

[141]:
```
centroids
```

[141]:
```
array([[ 0.9022813 ,  0.94666278,  0.94364226,  0.96600637,  0.92142726,
         0.91359058,  0.78781903,  0.86812783,  0.84073075,  0.91252208,
         0.83504189,  0.84035134,  0.64084523,  0.83582256,  0.78937134,
         0.83271777,  0.82464295,  0.82983492,  0.89916541,  0.94996185,
         0.94533212,  0.96746171,  0.92697076,  0.91912787,  0.77404641,
         0.86492543,  0.83912937,  0.91035214,  0.84266109,  0.84449831,
         0.61796436,  0.83180087,  0.77590607,  0.83014204,  0.83089472,
         0.83230979, -0.28308837],
       [-0.64964254, -0.6815972 , -0.67942243, -0.69552458, -0.66342762,
        -0.65778521, -0.5672297 , -0.62505204, -0.60532614, -0.65701589,
        -0.60123016, -0.60505296, -0.46140856, -0.60179224, -0.56834737,
        -0.5995568 , -0.59374292, -0.59748114, -0.64739909, -0.68397253,
        -0.68063912, -0.69657243, -0.66741894, -0.66177207, -0.55731341,
        -0.62274631, -0.60417314, -0.65545354, -0.60671598, -0.60803878,
        -0.44493434, -0.59889663, -0.55865237, -0.59770227, -0.5982442 ,
        -0.59926305,  0.20382363]])
```

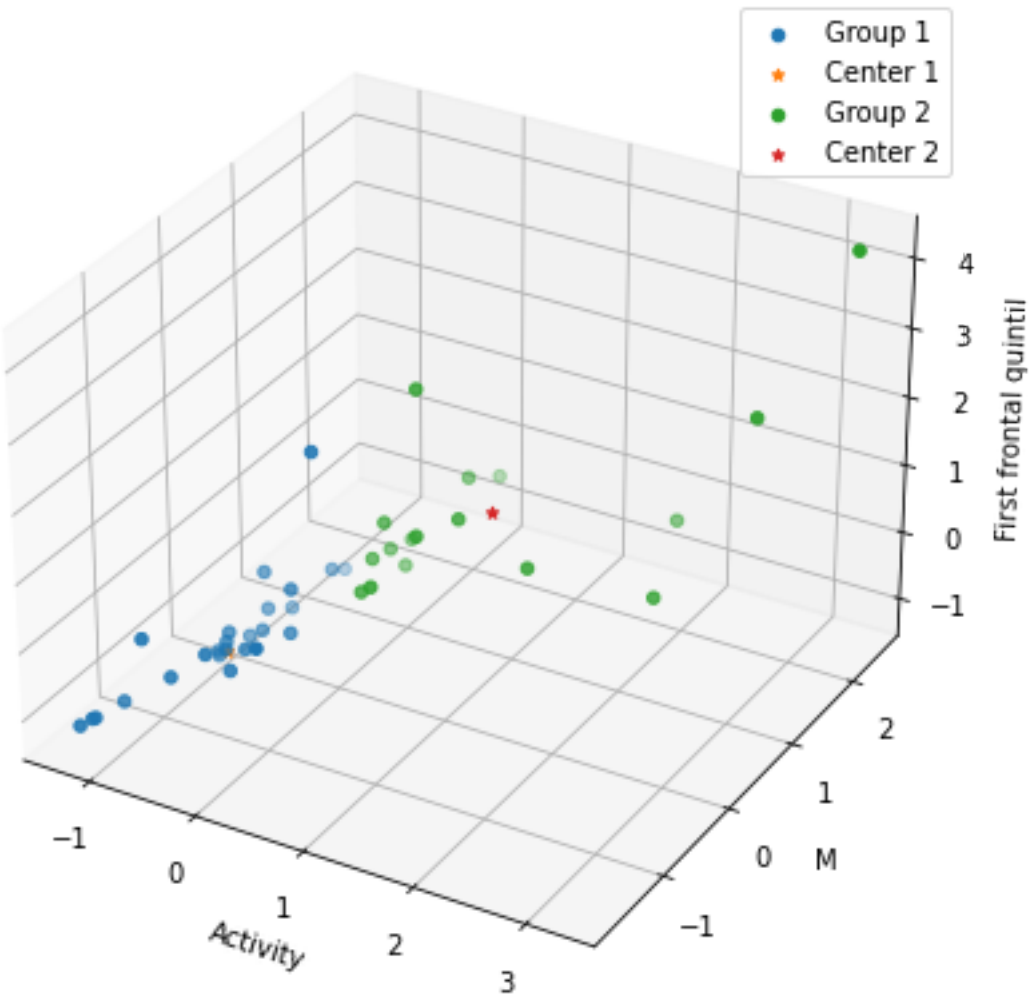
[142]:
```
centroids[0]-centroids[1]
```

[142]:
```
array([ 1.55192384,  1.62825998,  1.62306469,  1.66153095,  1.58485488,
        1.57137579,  1.35504874,  1.49317986,  1.44605689,  1.56953797,
        1.43627205,  1.44540431,  1.10225379,  1.43761481,  1.35771871,
        1.43227457,  1.41838587,  1.42731607,  1.5465645 ,  1.63393438,
        1.62597124,  1.66403414,  1.5943897 ,  1.58089994,  1.33135982,
        1.48767174,  1.44330251,  1.56580567,  1.44937707,  1.45253709,
        1.0628987 ,  1.4306975 ,  1.33455845,  1.42784432,  1.42913892,
        1.43157284, -0.486912  ])
```

[143]:
```
#To obtain the labels of each cluster
labels = model.labels_
```

[144]:
```
# Creating figure
fig = plt.figure(figsize = (10, 7))
ax = plt.axes(projection ="3d")
x=-2
y=1
z=30
ax.scatter3D(X[labels>0][:,x],X[labels>0][:,y],X[labels>0][:,z],label="Group 1")
ax.scatter3D(centroids[1,x],centroids[1,y],centroids[1,0],label="Center␣
↪1",marker='*')
ax.scatter3D(X[labels<1][:,x],X[labels<1][:,y],X[labels<1][:,z],label="Group 2")
ax.scatter3D(centroids[0,x],centroids[0,y],centroids[0,z],label="Center␣
↪2",marker='*')
ax.set_ylabel("M")
ax.set_xlabel("Activity")
ax.set_zlabel("First frontal quintil")
ax.legend()
```

[144]:
```
<matplotlib.legend.Legend at 0x7fa2be9241f0>
```



Observations:

- The results are very similar to the ones obtained in the N2 stage
- The first group corresponds to the more extreme data while the second one has the mayority of the sub-segments
- All the parameters expect the mobility are higher in the first goup

0.11 Temporal lobe collapse

To reduce the number of parameters we will collapse the parameters of the two temporal lobes into a single group.

[168]:
```
data=pd.read_csv("Extracted_data/Data_medians_4.csv")
del data['Unnamed: 0']
del data['38']
```

[169]:
```
data_t=data
```

[170]:
```
data_t
```

[170]:

	0	1	2	3	4	5	\
0	19.911429	21.826667	19.901111	19.076667	21.918889	14.850000	
1	40.660000	35.816667	38.405556	42.591111	40.958889	37.191667	
2	28.203333	24.951111	25.603333	33.302222	28.286667	24.416667	
3	18.286190	16.575556	17.173333	22.212222	18.586667	15.928333	
4	11.339524	10.327778	11.073333	9.653333	12.070000	10.046667	
..	…	…	…	…	…	…	
160	23.572857	21.060000	22.835556	24.652222	19.970000	18.556667	
161	18.736190	16.664444	17.992222	19.476667	16.011111	14.800000	
162	14.926190	13.685556	14.278889	15.434444	12.957778	11.726667	
163	10.816190	9.702222	10.213333	10.985556	9.072222	8.145000	
164	4.886190	4.464444	4.621111	4.972222	4.191111	3.780000	

	6	7	8	9	…	29	\
0	58.638571	60.644444	61.956667	57.580000	…	210.934721	
1	148.707143	114.740000	116.255556	180.642222	…	601.949326	
2	104.157619	78.117778	77.450000	141.416667	…	395.701329	
3	72.619048	55.015556	55.386667	98.113333	…	258.530174	
4	42.745238	33.640000	34.700000	37.300000	…	156.242749	
..	…	…	…	…	…	…	
160	62.023810	55.311111	62.484444	65.027778	…	233.876713	
161	47.885238	42.373333	47.960000	49.993333	…	179.285611	
162	37.418571	33.524444	37.350000	38.766667	…	137.755238	
163	27.838095	24.458889	27.936667	28.534444	…	99.599288	
164	12.803333	12.212222	13.241111	14.298889	…	41.261791	

	30	31	32	33	34	\
0	627.235043	648.175431	626.022091	643.377231	540.470154	
1	2122.510839	1710.063114	1684.448166	3062.885569	2100.222584	
2	1433.984345	1147.643210	1131.199674	2268.983257	1375.903119	
3	951.861362	774.702891	776.389821	1493.072896	943.001124	
4	587.817203	465.337699	467.104798	546.078070	612.918988	
..	…	…	…	…	…	
160	956.606279	901.840306	1079.626406	1000.445620	627.724242	
161	731.916047	694.028468	835.901439	759.237127	474.306587	
162	558.178258	505.701924	648.375509	566.989604	363.330987	
163	449.385031	411.347601	560.246779	459.417619	270.892424	
164	165.142991	188.748088	225.965814	206.167366	122.058242	

	35	36	37	39
0	495.039453	4.422272e+07	0.001152	Awake
1	1750.616297	2.508360e+06	0.002201	N2
2	1170.567469	1.469640e+06	0.001673	REM

```
3      787.208156  5.040967e+06  0.002785     N2
4      465.555425  9.224363e+06  0.001658     N3
..            …           …       …    …
160    663.898963  1.033058e+05  0.000996    REM
161    509.926345  3.701008e+04  0.000528     N1
162    390.608436  9.521701e+04  0.000363     N2
163    296.452757  6.754039e+06  0.000063     N3
164    109.744217  2.577399e+06  0.000092     N2

[165 rows x 39 columns]
```

To collapse the values of both temporal lobes, we must find a way to calculate the averages and standard deviations of two groups of the same size. The average is pretty straightforward. Just remember the equations:

$M_1 = \frac{1}{n_1}\sum_{n=1}^{n_1} X_n$

$M_2 = \frac{1}{n_2}\sum_{n=1}^{n_2} Y_n$

$M_t = \frac{1}{n_1+n_2}(\sum_{n=1}^{n_1} X_n + \sum_{n=1}^{n_2} Y_n)$

Note that $n_1 = n_2 = N$, so:

$M_t = \frac{1}{2N}(\sum_{n=1}^{N} X_n + \sum_{n=1}^{N} Y_n)$

$\therefore M_t = \frac{M_1+M_2}{2}$

[148]:
```
for i in range(0,len(data_t)):
     #We collapse the averages of the first quintile
    data_t.iloc[i,1]=(data.iloc[i,1]+data.iloc[i,2])/2
     # We collapse the averages of the second quintile
    data_t.iloc[i,7]=(data.iloc[i,7]+data.iloc[i,8])/2
     # We collapse the averages of the third quintile
    data_t.iloc[i,13]=(data.iloc[i,13]+data.iloc[i,14])/2

#Remove collapsed data columns
del data_t['2']
del data_t["8"]
del data_t["14"]
```

Now we are going to collapse the standard deviations. Let us remember that:

$S_1 = \sqrt{\frac{1}{n_1}\sum_{i=1}^{n_1}(x_i - \bar{x}_1)^2}$

$S_2 = \sqrt{\frac{1}{n_2}\sum_{i=1}^{n_2}(y_i - \bar{y}_2)^2}$

$S_t = \sqrt{\frac{1}{2N}\sum_{i=1}^{N}(x_i - \bar{y})^2 + (y_i - \bar{y})^2}$

En donde $\bar{y} = \frac{\bar{x}_1+\bar{y}_2}{2}$

$$\rightarrow S_t = \sqrt{\frac{1}{2N}\sum_{i=1}^{N}(x_i - \frac{\bar{x}_1+\bar{y}_2}{2})^2 + (y_i - \frac{\bar{x}_1+\bar{y}_2}{2})^2}$$

$$\rightarrow S_t = \sqrt{\frac{1}{2N}\sum_{i=1}^{N} x_i^2 - x_i(\bar{x}_1+\bar{y}_2) + \frac{(\bar{x}_1+\bar{y}_2)^2}{4} + y_i^2 - y_i(\bar{x}_1+\bar{y}_2) + \frac{(\bar{x}_1+\bar{y}_2)^2}{4}}$$

$$\rightarrow S_t = \sqrt{\frac{1}{2N}\sum_{i=1}^{N} x_i^2 - (x_i+y_i)(\bar{x}_1+\bar{y}_2) + \frac{(\bar{x}_1+\bar{y}_2)^2}{2} + y_i^2}$$

$$\rightarrow S_t = \sqrt{\frac{1}{2N}\sum_{i=1}^{N} x_i^2 - (x_i\bar{x}_1 + x_i\bar{y}_2 + y_i\bar{x}_1 + y_i\bar{y}_2) + \frac{(\bar{x}_1+\bar{y}_2)^2}{2} + y_i^2}$$

From the standard deviations of the individual groups we can derive that:

$$n_1 S_1^2 = \sum_{i=1}^{n_1}(x_i - \bar{x})^2$$

$$n_2 S_2^2 = \sum_{i=1}^{n_2}(y_i - \bar{y})^2$$

$$S_t = \sqrt{\frac{1}{2N}(NS_1^2 + NS_2^2 + N(\bar{y}_1 - \bar{y})^2 + N(\bar{y}_2 - \bar{y})^2)}$$

$$\therefore S_t = \sqrt{\frac{1}{2}(S_1^2 + S_2^2 + (\bar{y}_1 - \bar{y})^2 + (\bar{y}_2 - \bar{y})^2)}$$

[149]:
```
data_t.loc[0,'1']
```

[149]:
```
20.863888888888887
```

[150]:
```
for i in range(0,len(data_t)):
     #We collapse the standard deviations of the first quintile
    data_t.iloc[i,19]=((data.iloc[i,19]**2+data.iloc[i,20]**2+(data.
 ↪iloc[i,1]-data_t.loc[i ,'1'])**2+(data.iloc[i,2]-data_t.loc[i,'1'])**2)/
 ↪2)**0.5
     # We collapse the standard deviations of the second quintile
    data_t.iloc[i,25]=((data.iloc[i,25]**2+data.iloc[i,26]**2+(data.
 ↪iloc[i,7]-data_t.loc[i ,'7'])**2+(data.iloc[i,8]-data_t.loc[i,'7'])**2)/
 ↪2)**0.5
     # We collapse the standard deviations of the third quintile
    data_t.iloc[i,31]=((data.iloc[i,31]**2+data.iloc[i,32]**2+(data.
 ↪iloc[i,13]-data_t.loc[i ,'13'])**2+(data.iloc[i,14]-data_t.loc[i,'13'])**2)/
 ↪2)**0.5

#Remove collapsed data columns
del data_t['20']
del data_t['26']
del data_t['32']
```

[151]:
```
np.shape(data_t)
```

[151]:
```
(165, 33)
```

Notice that we managed to go down 6 dimensions.

0.11.1 K means

[152]:
```
# 0: Awake
# 1: N1
# 2: N2
# 3: N3
# 4: REM

for i in range(0,len(data_t)):
    if data_t.iloc[i,-1]=='Awake':
        data_t.iloc[i,-1]=0
    elif data_t.iloc[i,-1]=='N1':
        data_t.iloc[i,-1]=1
    elif data_t.iloc[i,-1]=='N2':
        data_t.iloc[i,-1]=2
    elif data_t.iloc[i,-1]=='N3':
        data_t.iloc[i,-1]=3
    elif data_t.iloc[i,-1]=='REM':
        data_t.iloc[i,-1]=4
    else:
        data_t.iloc[i,-1]=np.nan
```

[153]:
```
data_t2=data_t.dropna()
```

[154]:
```
data_t2
```

[154]:

	0	1	3	4	5	6	\
0	19.911429	20.863889	19.076667	21.918889	14.850000	58.638571	
1	40.660000	37.111111	42.591111	40.958889	37.191667	148.707143	
2	28.203333	25.277222	33.302222	28.286667	24.416667	104.157619	
3	18.286190	16.874444	22.212222	18.586667	15.928333	72.619048	
4	11.339524	10.700556	9.653333	12.070000	10.046667	42.745238	
..	…	…	…	…	…	…	
160	23.572857	21.947778	24.652222	19.970000	18.556667	62.023810	
161	18.736190	17.328333	19.476667	16.011111	14.800000	47.885238	
162	14.926190	13.982222	15.434444	12.957778	11.726667	37.418571	
163	10.816190	9.957778	10.985556	9.072222	8.145000	27.838095	
164	4.886190	4.542778	4.972222	4.191111	3.780000	12.803333	

	7	9	10	11	…	28	\
0	61.300556	57.580000	54.570000	43.411667	…	240.741312	
1	115.497778	180.642222	136.624444	117.346667	…	663.806544	
2	77.783889	141.416667	91.605556	77.440000	…	439.829679	
3	55.201111	98.113333	65.053333	53.331667	…	293.769864	
4	34.170000	37.300000	41.403333	33.041667	…	179.658886	
..	…	…	…	…	…	…	
160	58.897778	65.027778	46.384444	46.100000	…	235.548155	

```
161   45.166667   49.993333   35.975556   35.510000  …  180.235205
162   35.437222   38.766667   28.706667   27.921667  …  139.774550
163   26.197778   28.534444   20.034444   19.595000  …  101.098596
164   12.726667   14.298889   10.335556    9.438333  …   43.916360

             29          30          31          33          34  \
0    210.934721  627.235043  648.175431  643.377231  518.721683
1    601.949326 2122.510839 1710.063114 3062.885569 1934.006195
2    395.701329 1433.984345 1147.643210 2268.983257 1277.791388
3    258.530174  951.861362  774.702891 1493.072896  868.889384
4    156.242749  587.817203  465.337699  546.078070  544.583428
..          …           …           …           …           …
160  233.876713  956.606279  901.840306 1000.445620  648.364502
161  179.285611  731.916047  694.028468  759.237127  494.270849
162  137.755238  558.178258  505.701924  566.989604  378.484287
163   99.599288  449.385031  411.347601  459.417619  285.576327
164   41.261791  165.142991  188.748088  206.167366  116.909527

              35           36        37  39
0     495.039453 4.422272e+07  0.001152   0
1    1750.616297 2.508360e+06  0.002201   2
2    1170.567469 1.469640e+06  0.001673   4
3     787.208156 5.040967e+06  0.002785   2
4     465.555425 9.224363e+06  0.001658   3
..           …          …       …  ..
160   663.898963 1.033058e+05  0.000996   4
161   509.926345 3.701008e+04  0.000528   1
162   390.608436 9.521701e+04  0.000363   2
163   296.452757 6.754039e+06  0.000063   3
164   109.744217 2.577399e+06  0.000092   2

[157 rows x 33 columns]
```

[155]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_t2.iloc[:,:32])

Y=data_t2.iloc[:,-1].to_numpy()

X_train=X[:int(len(X)*0.85)]
X_test=X[int(len(X)*0.85):]

Y_train=Y[:int(len(X)*0.85)]
Y_test=Y[int(len(X)*0.85):]
```

[156]:
```
# Run local implementation of kmeans
model = KMeans(n_clusters=5, max_iter=100, init='random',n_init=10)
```

[157]:
```
model.fit(X_train)
```

[157]:
```
KMeans(init='random', max_iter=100, n_clusters=5)
```

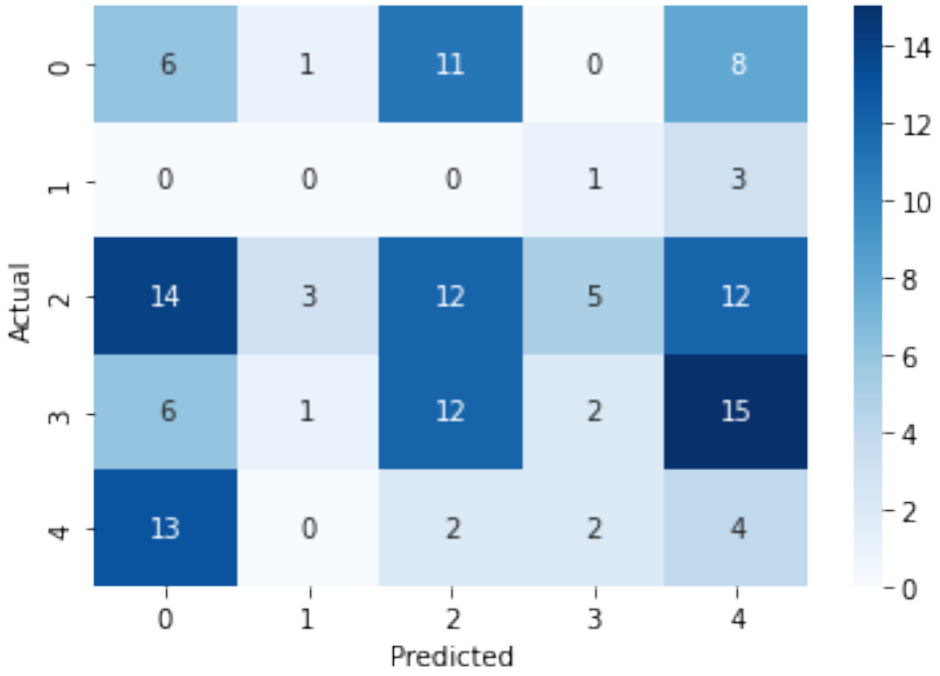
[158]:
```
#Obtaining clusters centroid
centroids = model.cluster_centers_

#To obtain the labels of each cluster
labels = model.labels_
```

[159]:
```
y_pred=model.predict(X_train)
```

[160]:
```
confusion_matrix = pd.crosstab(Y_train, y_pred, rownames=['Actual'],␣
↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



Observations:

- By adding the Hjorth parameters the accuracy to identify REM stages increased, however the N2 segment indentifiaction is worse.
- Some N2 and N3 segments have similar properties to the awake class.
- Other N2 segments were misidentified as REM.

0.11.2 K neighbors

Now we will try a clustering by neighbors method. By using this type of algorithm we hope to be able to find a big difference between the awake, NREM, and REM segments. Classification between NREM stages (N1, N2, N3) might not be accurate because they correspond to gradual progress. This means that some points of an N2 stage might be closer to an N3 segment than another N2 segment.

[171]:
```
for i in range(0,len(data_t)):
    if data_t.iloc[i,-1]!='Awake':
        if data_t.iloc[i,-1]!='N2':
            if data_t.iloc[i,-1]!='N3':
                if data_t.iloc[i,-1]!='REM':
                    data_t.iloc[i,-1]=np.nan
```

[172]:
```
data_t
```

[172]:

	0	1	2	3	4	5	\
0	19.911429	21.826667	19.901111	19.076667	21.918889	14.850000	
1	40.660000	35.816667	38.405556	42.591111	40.958889	37.191667	
2	28.203333	24.951111	25.603333	33.302222	28.286667	24.416667	
3	18.286190	16.575556	17.173333	22.212222	18.586667	15.928333	
4	11.339524	10.327778	11.073333	9.653333	12.070000	10.046667	
..	…	…	…	…	…	…	
160	23.572857	21.060000	22.835556	24.652222	19.970000	18.556667	
161	18.736190	16.664444	17.992222	19.476667	16.011111	14.800000	
162	14.926190	13.685556	14.278889	15.434444	12.957778	11.726667	
163	10.816190	9.702222	10.213333	10.985556	9.072222	8.145000	
164	4.886190	4.464444	4.621111	4.972222	4.191111	3.780000	

	6	7	8	9	…	29	\
0	58.638571	60.644444	61.956667	57.580000	…	210.934721	
1	148.707143	114.740000	116.255556	180.642222	…	601.949326	
2	104.157619	78.117778	77.450000	141.416667	…	395.701329	
3	72.619048	55.015556	55.386667	98.113333	…	258.530174	
4	42.745238	33.640000	34.700000	37.300000	…	156.242749	
..	…	…	…	…	…	…	
160	62.023810	55.311111	62.484444	65.027778	…	233.876713	
161	47.885238	42.373333	47.960000	49.993333	…	179.285611	
162	37.418571	33.524444	37.350000	38.766667	…	137.755238	
163	27.838095	24.458889	27.936667	28.534444	…	99.599288	
164	12.803333	12.212222	13.241111	14.298889	…	41.261791	

	30	31	32	33	34	\
0	627.235043	648.175431	626.022091	643.377231	540.470154	
1	2122.510839	1710.063114	1684.448166	3062.885569	2100.222584	
2	1433.984345	1147.643210	1131.199674	2268.983257	1375.903119	
3	951.861362	774.702891	776.389821	1493.072896	943.001124	

```
4     587.817203   465.337699   467.104798   546.078070   612.918988
..           …            …            …            …            …
160   956.606279   901.840306  1079.626406  1000.445620   627.724242
161   731.916047   694.028468   835.901439   759.237127   474.306587
162   558.178258   505.701924   648.375509   566.989604   363.330987
163   449.385031   411.347601   560.246779   459.417619   270.892424
164   165.142991   188.748088   225.965814   206.167366   122.058242

               35           36        37     39
0      495.039453  4.422272e+07  0.001152  Awake
1     1750.616297  2.508360e+06  0.002201     N2
2     1170.567469  1.469640e+06  0.001673    REM
3      787.208156  5.040967e+06  0.002785     N2
4      465.555425  9.224363e+06  0.001658     N3
..            …           …       …    …
160    663.898963  1.033058e+05  0.000996    REM
161    509.926345  3.701008e+04  0.000528    NaN
162    390.608436  9.521701e+04  0.000363     N2
163    296.452757  6.754039e+06  0.000063     N3
164    109.744217  2.577399e+06  0.000092     N2

[165 rows x 39 columns]
```

[173]:
```
data_t3=data_t.dropna()
```

[174]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_t3.iloc[:,:32])

Y=data_t3.iloc[:,-1].to_numpy()

X_train=X[:int(len(X)*0.75)]
X_test=X[int(len(X)*0.75):]

Y_train=Y[:int(len(X)*0.75)]
Y_test=Y[int(len(X)*0.75):]
```

[175]:
```
k_neighbor=KNeighborsClassifier(5)
```

[176]:
```
print(np.shape(X_train))
print(np.shape(Y_train))
```

```
(113, 32)
(113,)
```

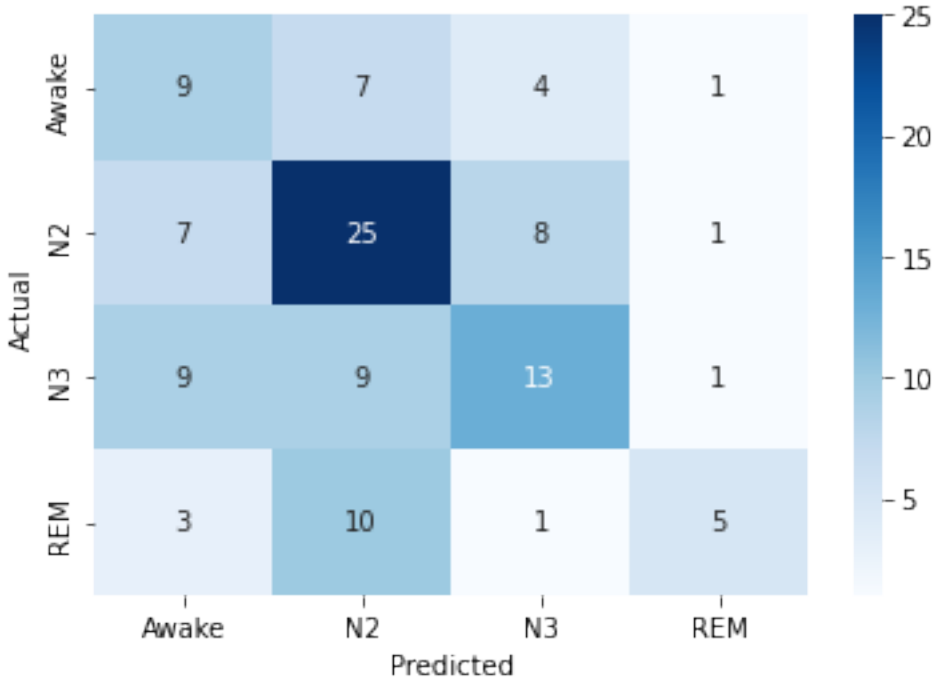
[177]:
```
k_neighbor.fit(X_train,Y_train)
```

[177]:
```
KNeighborsClassifier()
```

[178]:
```
y_pred=k_neighbor.predict(X_train)
```

[179]:
```
confusion_matrix = pd.crosstab(Y_train, y_pred, rownames=['Actual'],␣
↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



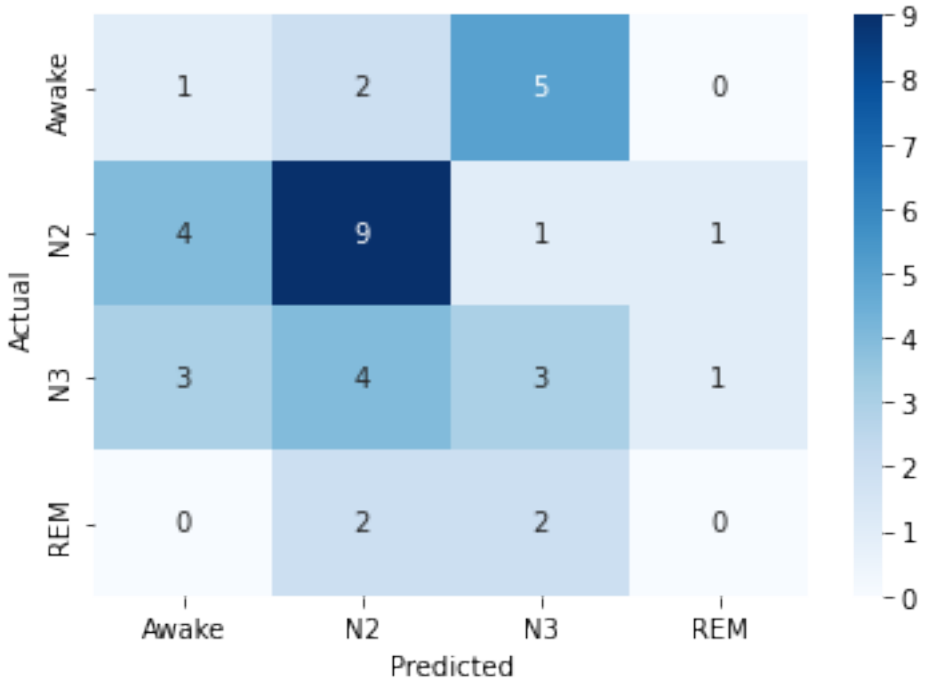
[180]:
```
sum(confusion_matrix.iloc[i,i] for i in range(0,len(confusion_matrix)))/
↪len(Y_train)
```

[180]: 0.46017699115044247

[181]:
```
y_pred=k_neighbor.predict(X_test)
```

[182]:
```
confusion_matrix = pd.crosstab(Y_test, y_pred, rownames=['Actual'],␣
↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



[183]:
```
sum(confusion_matrix.iloc[i,i] for i in range(0,len(confusion_matrix)))/
↪len(Y_test)
```

[183]: 0.34210526315789475

Observations: * Our initial hypothesis proved to be completely wrong. The results concluded the opposite to what we expected. * Differences beween N2 and N3 segments was outstanding. The identification of REM and Awake segments was an absolute caos. Most of the REM segments were classified as N2 and most of the awake sagments were classified as N3.

0.11.3 N1 vs N2 vs REM

Now we will try the same neighbors clustering analysis but excluding the awake segments. To distinguish between awake and asleep an ECG is a far better method than an EEG.

[184]:
```
for i in range(0,len(data_t)):
    if data_t.iloc[i,-1]!='N2':
        if data_t.iloc[i,-1]!='N3':
            if data_t.iloc[i,-1]!='REM':
                data_t.iloc[i,-1]=np.nan
```

[185]:
```
data_t4=data_t.dropna()
```

[186]:
```
# Standardize the data
X = StandardScaler().fit_transform(data_t4.iloc[:,:32])
```

```
Y=data_t4.iloc[:,-1].to_numpy()

X_train=X[:int(len(X)*0.75)]
X_test=X[int(len(X)*0.75):]

Y_train=Y[:int(len(X)*0.75)]
Y_test=Y[int(len(X)*0.75):]
```

[187]:
```
k_neighbor=KNeighborsClassifier(3)
```

[188]:
```
print(np.shape(X_train))
print(np.shape(Y_train))
```

```
(91, 32)
(91,)
```

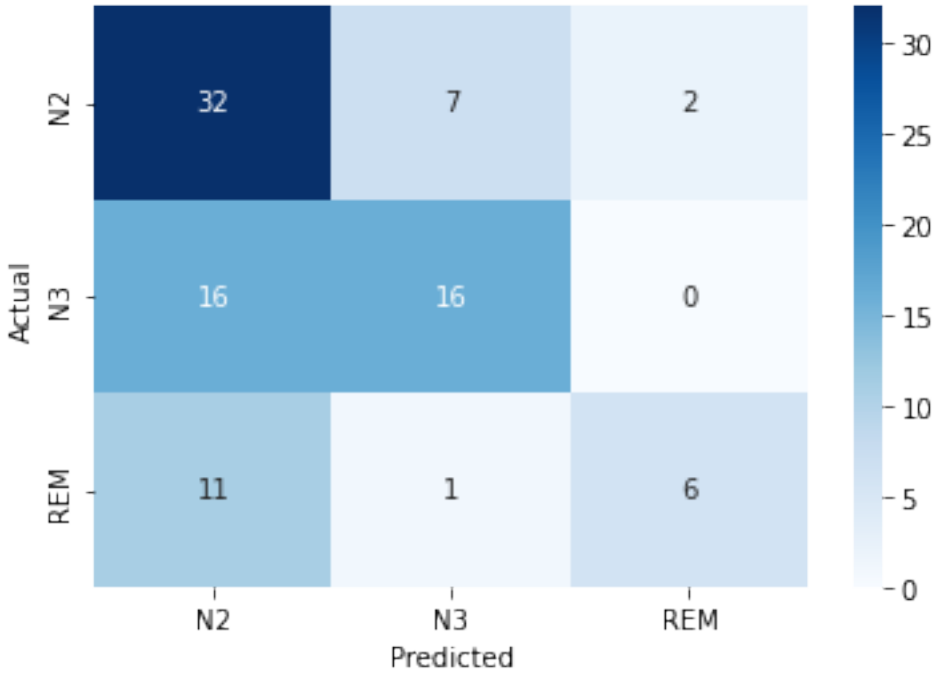
[189]:
```
k_neighbor.fit(X_train,Y_train)
```

[189]:
```
KNeighborsClassifier(n_neighbors=3)
```

[190]:
```
y_pred=k_neighbor.predict(X_train)
```

[191]:
```
confusion_matrix = pd.crosstab(Y_train, y_pred, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



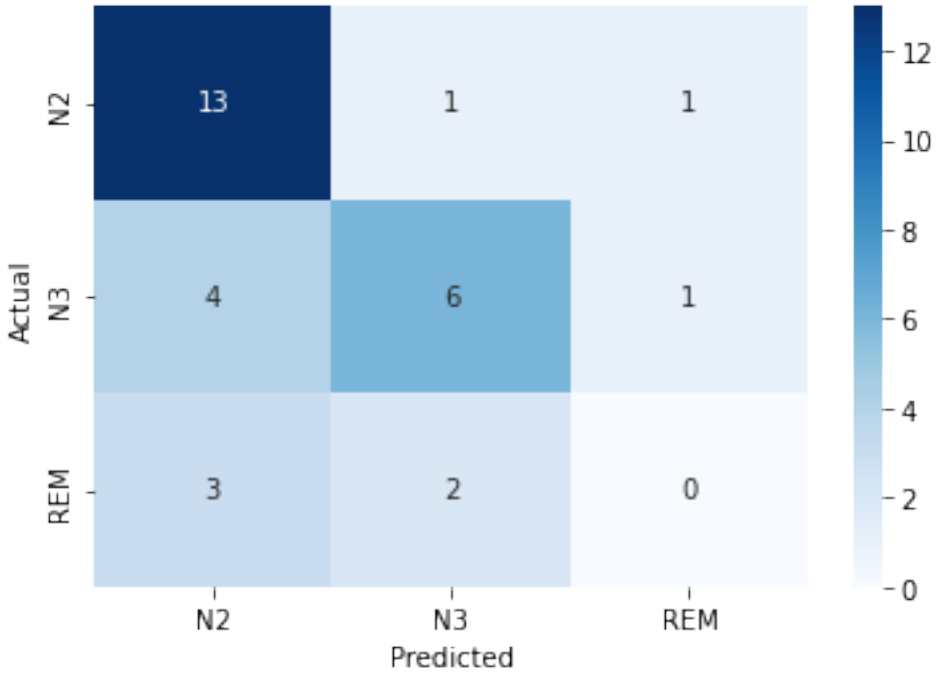
[192]:
```
sum(confusion_matrix.iloc[i,i] for i in range(0,len(confusion_matrix)))/
 ↪len(Y_train)
```

[192]: 0.5934065934065934

[193]:
```
y_pred=k_neighbor.predict(X_test)
```

[194]:
```
confusion_matrix = pd.crosstab(Y_test, y_pred, rownames=['Actual'],␣
 ↪colnames=['Predicted'])

sns.heatmap(confusion_matrix, annot=True, cmap="Blues")
plt.show()
```



[195]:
```
sum(confusion_matrix.iloc[i,i] for i in range(0,len(confusion_matrix)))/
↪len(Y_test)
```

[195]:
```
0.6129032258064516
```

Observations: * This has been the best algorithm so far. * It has a general accuracy of 61.29% * There is a slight bias in the training because most of the segments correspond to the N2 stage. * REM identification isn't accurate

Sensibility and accuracy: * N2: * Sensitivity: 13/15=86.66% * PPV: 13/20=65% * N3: * Sensitivity: 6/11= 54.54% * PPV: 6/9= 66.66% * REM: * Sensitivity: 0% * PPV: 0%

[ ]:
```

```