In [1]:

```python
"""
Created on Fri Oct 28 10:42:10 2022

@author:Brandon Botzer btb5103


Data Set Information:

The examined group comprised kernels belonging to three different varieties
of wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for
the experiment. High quality visualization of the internal kernel structure
was detected using a soft X-ray technique. It is non-destructive and
considerably cheaper than other more sophisticated imaging techniques like
scanning microscopy or laser technology. The images were recorded on
13x18 cm X-ray KODAK plates. Studies were conducted using combine harvested
wheat grain originating from experimental fields, explored at the Institute
of Agrophysics of the Polish Academy of Sciences in Lublin.


Attribute Information:

To construct the data, seven geometric parameters of wheat kernels were
measured:
    1. area A,
    2. perimeter P,
    3. compactness C = 4*pi*A/P^2,
    4. length of kernel,
    5. width of kernel,
    6. asymmetry coefficient
    7. length of kernel groove.



Relevant Papers:

M. Charytanowicz, J. Niewczas, P. Kulczycki, P.A. Kowalski, S. Lukasik, S. Zak,
 'A Complete Gradient Clustering Algorithm for Features Analysis of
 X-ray Images', in: Information Technologies in Biomedicine, Ewa Pietka,
 Jacek Kawa (eds.), Springer-Verlag, Berlin-Heidelberg, 2010, pp. 15-24.





Please use this data to finish the following tasks.

    1. Explore the data set (10 points)
    2. Use K-means clustering to group the seed data. (30 points)
    3. Use different linkage type for Hierarchical clustering to the seed data,
    which linkage type give the best result? (30 points)
    4. Use DBscan clustering group the seed data and find the best epses
    and min_samples value. (30 points)

"""
```

Out[1]:

```
"\nCreated on Fri Oct 28 10:42:10 2022\n\n@author:Brandon Botzer btb5103\n\n\nData Set Information:\n\nThe examined group comprised kernels belonging to three different varieties \nof wheat: Kama, Rosa and Canadian, 70 elements each, randomly selected for \nthe experiment. High quality visualization of the internal kernel structure \nwas detected using a soft X-ray technique. It is non-destructive and \nconsiderably cheaper than other more sophisticated imaging techniques like \nscanning microscopy or laser technology. The images were recorded on \n13x18 cm X-ray KODAK plates. Studies were conducted using combine harvested \nwheat grain originating from experimental fields, explored at the Institute \nof Agrophysics of the Polish Academy of Sciences in Lublin. \n\n\nAttribute Information:\n\nTo construct the data, seven geometric parameters of wheat kernels were \nmeasured: \n    1. area A, \n    2. perimeter P, \n    3. compactness C = 4*pi*A/P^2, \n    4. length of kernel, \n    5. width of kernel, \n    6. asymmetry coefficient \n    7. length of kernel groove. \n\n\n    \nRelevant Papers:\n\nM. Charytanowicz, J. Niewczas, P. Kulczycki, P.A. Kowalski, S. Lukasik, S. Zak,\n 'A Complete Gradient Clustering Algorithm for Features Analysis of \n X-ray Images', in: Information Technologies in Biomedicine, Ewa Pietka, \n Jacek Kawa (eds.), Springer-Verlag, Berlin-Heidelberg, 2010, pp. 15-24.\n  \n\n\n\n\nPlease use this data to finish the following tasks.\n\n    1. Explore the data set (10 points)\n    2. Use K-means clustering to group the seed data. (30 points)\n    3. Use different linkage type for Hierarchical clustering to the seed data, \n    which linkage type give the best result? (30 points)\n    4. Use DBscan clustering group the seed data and find the best epses \n    and min_samples value. (30 points)\n    \n"
```

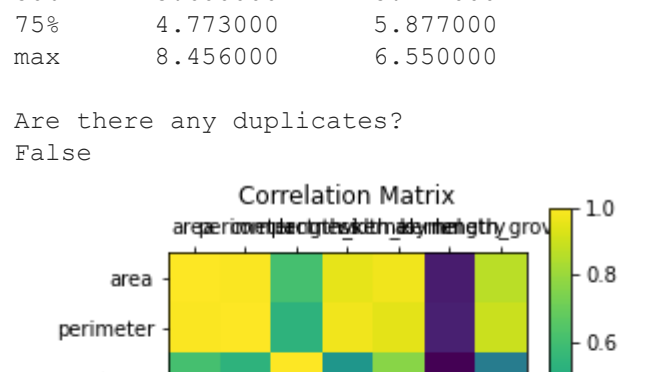
In [2]:

```python
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import metrics

#0. Read the data in

#set the read path
readpath = "J:\DSDegree\PennState\DAAN_862\Week 11\Homework"
#change the directory
os.chdir(readpath)
#This data set does not have a header for the column names
#Set the column names for the data frame
column_names = ["area", "perimeter", "compactness", "length_kernel",
                "width_kernel", "asymmetry", "length_grove", "type"]
#read in the csv data, set header=0 and names=column_names to
#have proper naming for the column information
df = pd.read_csv("seeds_dataset.csv", header=0, names=column_names)

#in case I want to split the class into dummy variables
dummy_class = pd.get_dummies(df["type"])
#drop class so I can use the dummy class instead
df_dummy = df.drop("type", axis = 1)
#Right Join the dummy classes to the seed dataframe
df_dummy = df_dummy.join(dummy_class, how = 'right')

#Bring the seed variables by themselves as a data frame
seeds = df.iloc[:, :7]
```

In [3]:

```python
#1. Explore the data set (10 points)

corr_val = seeds.corr()

#plot a correlation matrix
plt.matshow(corr_val)
plt.title("Correlation Matrix")
plt.colorbar()
plt.xticks(range(7), list(seeds.columns))
plt.yticks(range(7), list(seeds.columns))


#get some info about the seeds data frame
print("\nData frame information:\n")
print(seeds.info())
print()
print(seeds.describe())


#the data should be clean but I'll still check for duplicates
dups = seeds.duplicated()
print("\nAre there any duplicates?\n" + str(dups.max()))
```

```
Data frame information:

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 209 entries, 0 to 208
Data columns (total 7 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   area           209 non-null    float64
 1   perimeter      209 non-null    float64
 2   compactness    209 non-null    float64
 3   length_kernel  209 non-null    float64
 4   width_kernel   209 non-null    float64
 5   asymmetry      209 non-null    float64
 6   length_grove   209 non-null    float64
dtypes: float64(7)
memory usage: 11.6 KB
None

             area   perimeter  compactness  length_kernel  width_kernel  \
count  209.000000  209.000000   209.000000     209.000000    209.000000
mean    14.845550   14.557943     0.870999       5.627890      3.258349
std      2.916545    1.308949     0.023686       0.444029      0.378603
min     10.590000   12.410000     0.808100       4.899000      2.630000
25%     12.260000   13.450000     0.856700       5.262000      2.941000
50%     14.340000   14.290000     0.873500       5.520000      3.232000
75%     17.320000   15.730000     0.887900       5.980000      3.562000
max     21.180000   17.250000     0.918300       6.675000      4.033000

        asymmetry  length_grove
count  209.000000    209.000000
mean     3.707278      5.408971
std      1.503657      0.492487
min      0.765100      4.519000
25%      2.587000      5.045000
50%      3.600000      5.224000
75%      4.773000      5.877000
max      8.456000      6.550000

Are there any duplicates?
False
```



In [4]:

```python
#2. Use K-means clustering to group the seed data. (30 points)

#I will start this using k = 3
#I will later show using the elbow method that k = 3 is correct

from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler

#Normalize independent variables for the seed data

#make the scaler object
scaler = MinMaxScaler()
seed_X = scaler.fit_transform(seeds)
y = df.type


#build the k-Means object
kmeans = KMeans(n_clusters = 3, random_state=226)
#fit and predict the clusters
y_seed_pred = kmeans.fit_predict(seed_X)

#Get metrics
print("\nMetrics for K-Means at k = 3:")
print("  Homogentiy: " + str(metrics.homogeneity_score(y, y_seed_pred)))
print("  Completeness: " + str(metrics.completeness_score(y, y_seed_pred)))
print("  Adjusted Rand Score: " +
      str(metrics.adjusted_rand_score(y, y_seed_pred)))
print("  Silhouette Score: " +
      str(metrics.silhouette_score(seed_X, y_seed_pred, metric = 'euclidean')))
```

```
Metrics for K-Means at k = 3:
  Homogentiy: 0.6725949912605675
  Completeness: 0.6744319428588382
  Adjusted Rand Score: 0.7044261490237547
  Silhouette Score: 0.42119810498616345
```

In [5]:

```python
#try to create a plt that shows the clustering worked
#Try this for all combinations of the variables to see good
#separating distinguishers

#I'll comment the loop out later and ony show one plot of this
"""
for i in range (0,7):
    for j in range(0,7):
        if i != j:

            centers = kmeans.cluster_centers_
            centers_a = centers[:, i]
            centers_b = centers[:, j]

            plt.figure()
            # predicted  categories (c = y_pred).
            plt.scatter(seed_X[:, i], seed_X[:, j], c = y_seed_pred,
                        label = 'Predicted')
            # PLot the cluster centroids with 'X' and color is red
            plt.scatter(centers_a, centers_b, s= 100, c = 'r',
                        marker = 'x', label = 'Cluster Centers')
            plt.title('Clustering results')
            plt.xlabel(seeds.columns[i])
            plt.ylabel(seeds.columns[j])
            plt.legend()
"""
```

Out[5]:

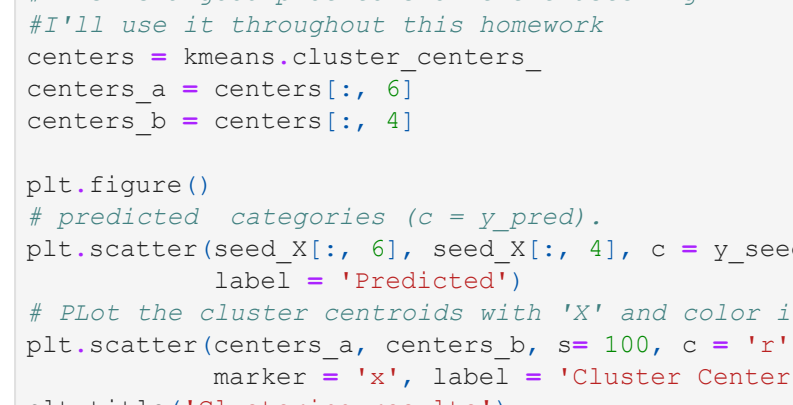
```
"\nfor i in range (0,7):\n    for j in range(0,7):\n        if i != j: \n\n            centers = kmeans.cluster_centers_\n            centers_a = centers[:, i]\n            centers_b = centers[:, j]\n            \n            plt.figure()\n            # predicted  categories (c = y_pred).\n            plt.scatter(seed_X[:, i], seed_X[:, j], c = y_seed_pred, \n                        label = 'Predicted')\n            # PLot the cluster centroids with 'X' and color is red\n            plt.scatter(centers_a, centers_b, s= 100, c = 'r', \n                        marker = 'x', label = 'Cluster Centers')\n            plt.title('Clustering results')\n            plt.xlabel(seeds.columns[i])\n            plt.ylabel(seeds.columns[j])\n            plt.legend()\n"
```

In [6]:

```python
#This is a good plot to show the clustering
#I'll use it throughout this homework
centers = kmeans.cluster_centers_
centers_a = centers[:, 6]
centers_b = centers[:, 4]

plt.figure()
# predicted  categories (c = y_pred).
plt.scatter(seed_X[:, 6], seed_X[:, 4], c = y_seed_pred,
            label = 'Predicted')
# PLot the cluster centroids with 'X' and color is red
plt.scatter(centers_a, centers_b, s= 100, c = 'r',
            marker = 'x', label = 'Cluster Centers')
plt.title('Clustering results')
plt.ylabel("Width_kernal")
plt.xlabel("Length_Grove")
plt.legend()
```

Out[6]:

```
<matplotlib.legend.Legend at 0x117871f64c0>
```



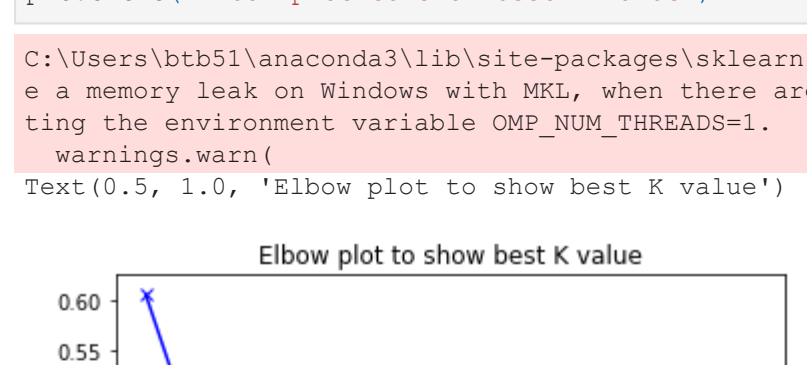
In [7]:

```python
#Use the elbow method to show that k = 3 is the proper choice

#computing distances from two inputs
from scipy.spatial.distance import cdist

#set up a range of k values
k_vals = range(1,10)

#set an empty list for the mean dispersions
meanDisp = []

for k in k_vals:

    #build the K-Means object with the variable n_clusters
    kmeans = KMeans(n_clusters=k)
    #fit the data via K-means
    kmeans.fit(seed_X)

    #from the sklearn book (Hackeling 2nd ed., pg. 208)
    #Find the average minimum distance to the cluster centers
    #shows if you've done a good job with clustering
    #if K = number of points then this would be zero
    meanDisp.append(sum(np.min(cdist(seed_X, kmeans.cluster_centers_,
                                     'euclidean'),
                               axis = 1)) / seed_X.shape[0])

#plot the elbow plot
plt.figure()
plt.plot(k_vals, meanDisp, 'bx-')
plt.xlabel("K")
plt.ylabel("Average Dispersion")
plt.title("Elbow plot to show best K value")
```

```
C:\Users\btb51\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

Out[7]:

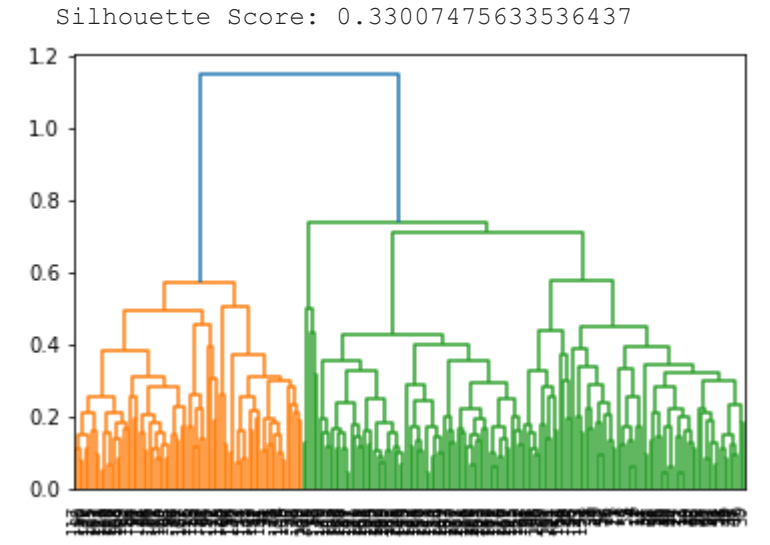
```
Text(0.5, 1.0, 'Elbow plot to show best K value')
```



In [8]:

```python
#3. Use different linkage type for Hierarchical clustering to the seed data,
#   which linkage type give the best result? (30 points)


from sklearn.cluster import AgglomerativeClustering


#Using linkage = 'average'
hier1_seed_pred = AgglomerativeClustering(n_clusters=3,
                                          affinity='euclidean',
                                          linkage='average').fit_predict(seed_X)

#Get metrics
print("\nMetrics for Hierarchical using Average Linkage:")
print("  Homogentiy: " + str(metrics.homogeneity_score(y, hier1_seed_pred)))
print("  Completeness: " + str(metrics.completeness_score(y, hier1_seed_pred)))
print("  Adjusted Rand Score: " +
      str(metrics.adjusted_rand_score(y, hier1_seed_pred)))
print("  Silhouette Score: " +
      str(metrics.silhouette_score(seed_X, hier1_seed_pred,
                                   metric = 'euclidean')))

#Create the dendrogram of the clusters
from scipy.cluster.hierarchy import dendrogram, linkage

Z_avg = linkage(seed_X, method = 'average')

plt.figure()
den = dendrogram(Z_avg, leaf_font_size = 8)
```

```
Metrics for Hierarchical using Average Linkage:
  Homogentiy: 0.49095617606258773
  Completeness: 0.7252345398264144
  Adjusted Rand Score: 0.5079770294939454
  Silhouette Score: 0.33007475633536437
```



In [9]:

```python
#Using linkage = 'complete'
hier2_seed_pred = AgglomerativeClustering(n_clusters=3,
                                          affinity='euclidean',
                                          linkage='complete').fit_predict(seed_X)
#Get metrics
print("\nMetrics for Hierarchical using Complete Linkage:")
print("  Homogentiy: " + str(metrics.homogeneity_score(y, hier2_seed_pred)))
print("  Completeness: " + str(metrics.completeness_score(y, hier2_seed_pred)))
print("  Adjusted Rand Score: " +
      str(metrics.adjusted_rand_score(y, hier2_seed_pred)))
print("  Silhouette Score: " +
      str(metrics.silhouette_score(seed_X, hier2_seed_pred,
                                   metric = 'euclidean')))
```

```
Metrics for Hierarchical using Complete Linkage:
  Homogentiy: 0.6086070715344328
  Completeness: 0.6345373983245087
  Adjusted Rand Score: 0.5651989291978214
  Silhouette Score: 0.362782119361832
```

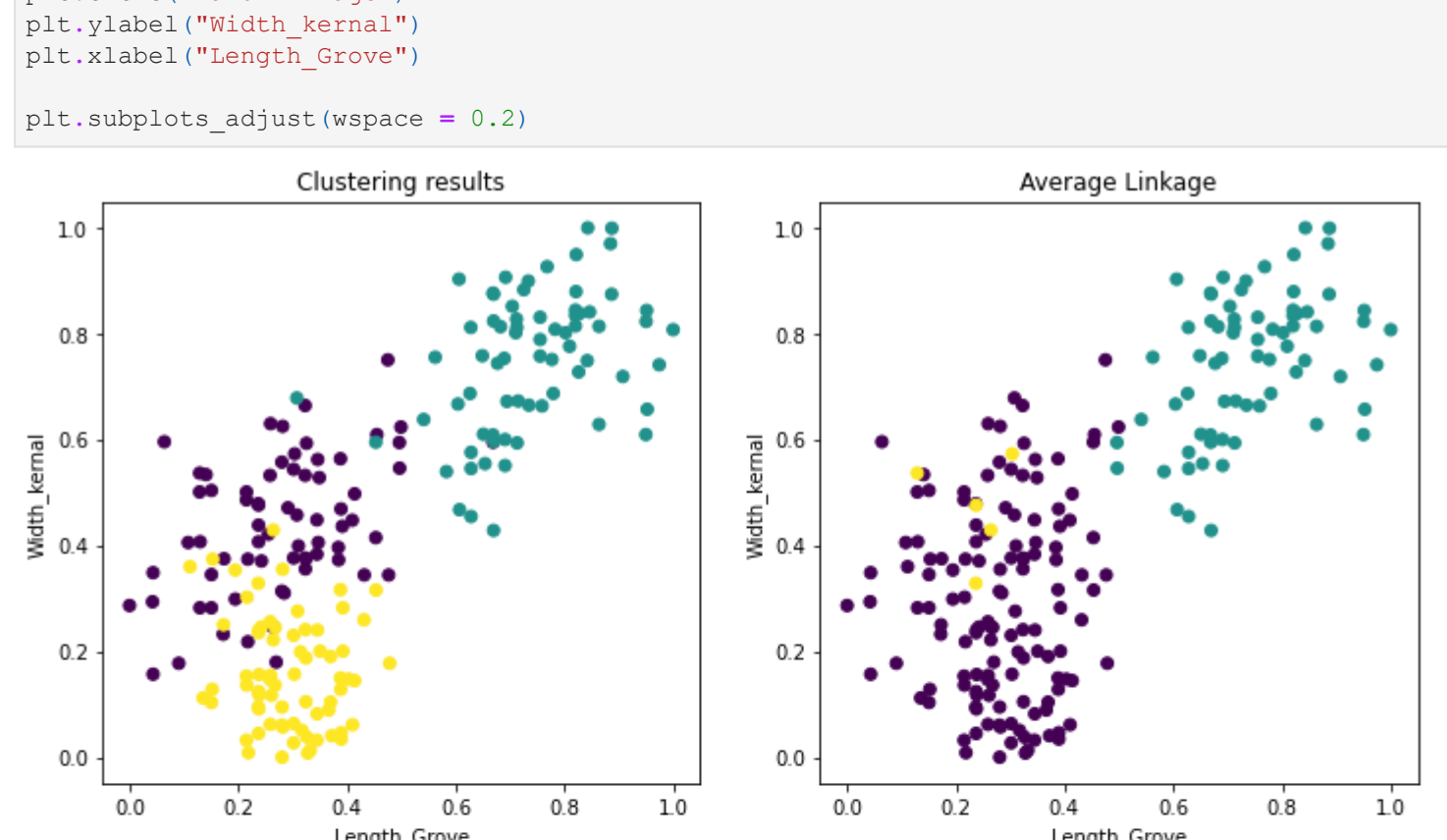
In [10]:

```python
#Using linkage = 'ward'
hier3_seed_pred = AgglomerativeClustering(n_clusters=3,
                                          affinity='euclidean',
                                          linkage='ward').fit_predict(seed_X)
#Get metrics
print("\nMetrics for Hierarchical using Ward Linkage:")
print("  Homogentiy: " + str(metrics.homogeneity_score(y, hier3_seed_pred)))
print("  Completeness: " + str(metrics.completeness_score(y, hier3_seed_pred)))
print("  Adjusted Rand Score: " +
      str(metrics.adjusted_rand_score(y, hier3_seed_pred)))
print("  Silhouette Score: " +
      str(metrics.silhouette_score(seed_X, hier3_seed_pred,
                                   metric = 'euclidean')))
```

```
Metrics for Hierarchical using Ward Linkage:
  Homogentiy: 0.6457668883106167
  Completeness: 0.6511585613272635
  Adjusted Rand Score: 0.6521911239329006
  Silhouette Score: 0.39063836643092875
```

In [11]:

```python
#Based on the metrics run, the Ward Linkage performs the best for clustering
#this data set having the highest silohette score (amoung other values)


#Plot the different Hierarchical Clusterings linkage types
plt.figure(figsize = (12,12))
plt.subplot(221)
plt.scatter(seed_X[:, 6], seed_X[:, 4], c = y)

plt.title('Clustering results')
plt.ylabel("Width_kernal")
plt.xlabel("Length_Grove")


plt.subplot(222)
plt.scatter(seed_X[:, 6], seed_X[:, 4], c = hier1_seed_pred)
plt.title('Average Linkage')
plt.ylabel("Width_kernal")
plt.xlabel("Length_Grove")

plt.subplot(223)
plt.scatter(seed_X[:, 6], seed_X[:, 4], c = hier2_seed_pred)
plt.title('Complete Linkage')
plt.ylabel("Width_kernal")
plt.xlabel("Length_Grove")

plt.subplot(224)
plt.scatter(seed_X[:, 6], seed_X[:, 4], c = hier3_seed_pred)
plt.title('Ward Linkage')
plt.ylabel("Width_kernal")
plt.xlabel("Length_Grove")

plt.subplots_adjust(wspace = 0.2)
```



In [12]:

```python
#4. Use DBscan clustering group the seed data and find the best epses
#   and min_samples value. (30 points)

from sklearn.cluster import DBSCAN


#build the model and fit predict the clusters
db_y_seed_pred = DBSCAN(eps = 0.2, min_samples=5,).fit_predict(seed_X)

#Get metrics
print("\nMetrics for DBSCAN with eps = 0.2 and min_samples = 5:")
print("  Homogentiy: " + str(metrics.homogeneity_score(y, db_y_seed_pred)))
print("  Completeness: " + str(metrics.completeness_score(y, db_y_seed_pred)))
print("  Adjusted Rand Score: " +
      str(metrics.adjusted_rand_score(y, db_y_seed_pred)))
print("  Silhouette Score: " +
      str(metrics.silhouette_score(seed_X, db_y_seed_pred,
                                   metric = 'euclidean')))
```

```
Metrics for DBSCAN with eps = 0.2 and min_samples = 5:
  Homogentiy: 0.46515564949711663
  Completeness: 0.45915663725116695
  Adjusted Rand Score: 0.38544486634577096
  Silhouette Score: 0.08210506434215879
```

In [13]:

```python
#Run a search for the best 'epses' and 'min_samples' for the DBSCAN
print("\n\nSearching for the best parameters for DBSCAN... \n\n")
#make and empty list to store results
results = []

#make the 'epses' and 'min_samples' grid
epses = [0.1, 0.2, 0.3, 0.4, 0.5, 1.0]
min_sampels = [2, 5, 10, 15]

#run over the search grid manually
for e in epses:
    for n in min_sampels:

        #set up the DBSCAN model
        model = DBSCAN(eps = e, min_samples = n)

        #Get the predicted clusters
        db_y_seed_pred_temp = model.fit_predict(seed_X)

        #find the number of clusters
        n_clusters = np.unique(model.labels_).size

        #get the sihouellette metrics
        #Check if there is just one cluster, this will error the Silhouette
        #so if there is only one cluster this is a bad score and will
        #equal 0 (assuming there is more than one cluster...)
        if n_clusters == 1:
            results.append((e,n,np.nan, n_clusters))
        else:
            score = metrics.silhouette_score(seed_X, db_y_seed_pred_temp,
                                             metric = 'euclidean')
            #store the score results
            results.append((e, n, score, n_clusters))

#put results in to a data frame
res_hold = pd.DataFrame(results, columns = ["epses",
                                            "min_samples",
                                            "sil_score",
                                            "n_clusters"])

print("\nThe Silhouette scores from the first DBSCAN search:\n")
print(res_hold)

#Get the epses, min_samps, and sil_score as reference values for the loop
r_epses = res_hold.epses[res_hold.sil_score.idxmax()]
r_min_samp = res_hold.min_samples[res_hold.sil_score.idxmax()]
r_sil_score = res_hold.sil_score.max()
```

```
Searching for the best parameters for DBSCAN...



The Silhouette scores from the first DBSCAN search:

    epses  min_samples  sil_score  n_clusters
0     0.1            2  -0.311723          27
1     0.1            5        NaN           1
2     0.1           10        NaN           1
3     0.1           15        NaN           1
4     0.2            2  -0.079938           9
5     0.2            5   0.082105           5
6     0.2           10   0.125795           4
7     0.2           15  -0.193016           4
8     0.3            2  -0.052258           3
9     0.3            5   0.055722           2
10    0.3           10   0.055722           2
11    0.3           15   0.199218           2
12    0.4            2        NaN           1
13    0.4            5        NaN           1
14    0.4           10        NaN           1
15    0.4           15   0.080234           2
16    0.5            2        NaN           1
17    0.5            5        NaN           1
18    0.5           10        NaN           1
19    0.5           15        NaN           1
20    1.0            2        NaN           1
21    1.0            5        NaN           1
22    1.0           10        NaN           1
23    1.0           15        NaN           1
```

In [14]:

```python
#We've found the epses and min_sampels around 0.3 and 15 to be best

#will refine search to obtain better sil_score
#DO this in the while loop
#Find espes and min_num from trial 1 above, then do + / - around the params
#while loop goes until sil_score results don't increase (could be local max)
#   I have not accounted for local maximums (could provide inertia for this)

check = False

epses_inc = 0.05
min_samp_inc = 1

while check == False:

    #make and empty list to store results
    results = []

    #make the 'epses' and 'min_samples' grid
    epses_hold = [r_epses-(2*epses_inc),
                  r_epses-epses_inc,
                  r_epses,
                  r_epses+epses_inc,
                  r_epses+(2*epses_inc)]
    min_sampels_hold = [r_min_samp-(2*min_samp_inc),
                        r_min_samp-min_samp_inc,
                        r_min_samp,
                        r_min_samp+min_samp_inc,
                        r_min_samp+(2*min_samp_inc)]


    #run over the search grid manually
    for e in epses_hold:
        for n in min_sampels_hold:

            #set up the DBSCAN model
            model = DBSCAN(eps = e, min_samples = n)

            #Get the predicted clusters
            db_y_seed_pred_temp = model.fit_predict(seed_X)

            #find the number of clusters
            n_clusters = np.unique(model.labels_).size

            #get the sihouellette metrics
            #Check if there is just one cluster, this will error the Silhouette
            #so if there is only one cluster this is a bad score and will
            #equal 0 (assuming there is more than one cluster...)
            if n_clusters == 1:
                results.append((e,n,np.nan, n_clusters))
            else:
                score = metrics.silhouette_score(seed_X, db_y_seed_pred_temp,
                                                 metric = 'euclidean')
                #store the score results
                results.append((e, n, score, n_clusters))

    #put results in to a temp data frame
    res_temp = pd.DataFrame(results, columns = ["epses",
                                                "min_samples",
                                                "sil_score",
                                                "n_clusters"])

    r_epses_temp = res_temp.epses[res_temp.sil_score.idxmax()]
    r_min_samp_temp = res_temp.min_samples[res_temp.sil_score.idxmax()]
    r_sil_score_temp = res_temp.sil_score.max()

    #check the reference sil_score for changes
    if r_sil_score_temp > r_sil_score:
        #a better score has been found.  Reassign the reference vals and go again
        r_sil_score = r_sil_score_temp
        r_epses = r_epses_temp
        r_min_samp = r_min_samp_temp
    elif r_sil_score_temp <= r_sil_score:
        #No improvements have been made.  Hit the flag and break the loop
        check = True




#reset the reference scores
res = res_temp
best_clusters = res.n_clusters[res.sil_score.idxmax()]

print("\nThe Silhouette scores from the last DBSCAN search:\n")
print(res)
print("\nThe best DBSCAN values are:")
print("  epses:             " + str(r_epses))
print("  min_samples:       " + str(r_min_samp))
print("  Silhouette Score: " + str(r_sil_score))
print("  N Clusters:        " + str(best_clusters))
```

```
The Silhouette scores from the last DBSCAN search:

    epses  min_samples  sil_score  n_clusters
0    0.15            9  -0.067371           3
1    0.15           10  -0.097755           2
2    0.15           11        NaN           1
3    0.15           12        NaN           1
4    0.15           13        NaN           1
5    0.20            9   0.118907           5
6    0.20           10   0.125795           4
7    0.20           11   0.104666           4
8    0.20           12   0.082696           4
9    0.20           13   0.055205           4
10   0.25            9   0.143059           2
11   0.25           10   0.346873           3
12   0.25           11   0.346873           3
13   0.25           12   0.338391           3
14   0.25           13   0.329665           3
15   0.30            9   0.055722           2
16   0.30           10   0.055722           2
17   0.30           11   0.059091           2
18   0.30           12   0.059091           2
19   0.30           13   0.116260           2
20   0.35            9   0.104581           2
21   0.35           10   0.104581           2
22   0.35           11   0.104581           2
23   0.35           12   0.104581           2
24   0.35           13   0.104581           2

The best DBSCAN values are:
  epses:             0.25
  min_samples:       11
  Silhouette Score: 0.3468734616603017
  N Clusters:        3
```