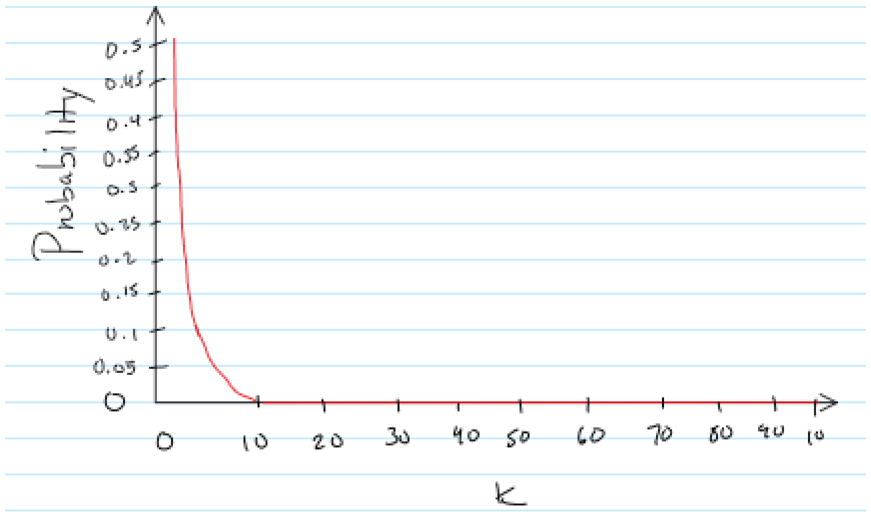
Homework 4

April 11, 2020

1 Recitation Exercises

1.1 Exercise 4

a) Plot the probability of obtaining one point from each cluster in a sample of size K for values of K between 2 to 100.



b) For K clusters, K = 10, 100, & 1000, find the probability that a sample of size 2K contains at least one point from each cluster.

p = $\frac{2K!}{K^K}$

For K = 10,

p = $\frac{2K!}{K^K} = \frac{2*10!}{10^{1}0} = 0.000728$

For K = 100,

p = $\frac{2K!}{K^K} = \frac{2*100!}{100^{1}00} = 1.867 \text{ x } 10^{-42}$

For K = 1000,

p = $\frac{2K!}{K^K} = \frac{2*1000!}{1000^{1}000} = 0$

1.2 Exercise 7

Given the data set: - there are m points & K clusters - half the points & clusters are in “more dense” regions - half the points & clusters are in “less dense” regions - the 2 regions are well-separated from each other

Which of the following should occur in order to minimize the squared error when finding K clusters:

The correct answer would be (c) which was to move centroids to the dense region. The less dense region would require more centroids if the squared error needs to be minimized. Recall that the less dense region tends to produce “noise” which would make it harder to identify clusters, hence needing more.

1.3 Exercise 11

Total SSE is the sume of the SSE for each separate attribute. - What does it mean if the SSE for one variable is low for all clusters?

```
If the SSE of one attribute is always low for all clusters, than the variable is just a consta
```

- Low for just one cluster?

  Then it be the opposite of the above, it would actually contribute to defining a cluster.

- High for all clusters?

  If it’s is high for every cluster, then I’d assume it is either noise or an outlier.

- High for one cluster?

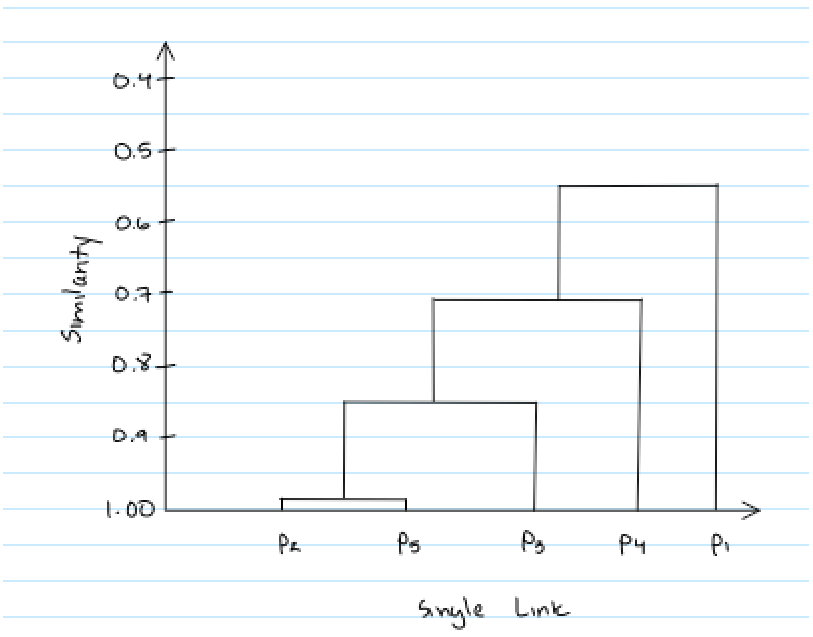
  Another outlier, which would not help defining a cluster.

- How could you use the per variable SSE info to improve your clustering?

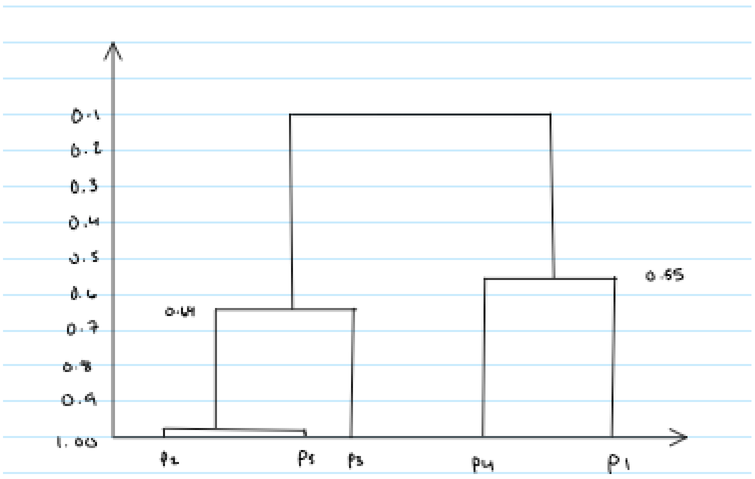
  It can help in deciding which attributes to eliminate. Ch. 7 mentioned how sampling the data before clustering could be useful to eliminate the noise or outliers within the data, which would be useful to conserve time of the computation.

1.4 Exercise 16

a) Single Link



b) Complete Link



1.5 Exercise 17

Given set of 1-dimensional points: {6, 12, 18, 24, 30, 42, 48}

a) For each of the following sets of inital centroids, create 2 clusters by assigning each point to the nearest centroid & then calculate the total squared error for each of the 2 clusters.

   i. {18, 45}

   Reasoning: Using the minimal difference between points to find out which cluster they belong in... i.e 30 - 18 = 12 vs 45 - 30 = 15

   Cluster 1 : {6, 12, 18, 24, 30}, Error = 360

Cluster 2: {42, 48}, Error = 18

Thus, total error = 378

ii. {15, 40}

Cluster 1 : {6, 12, 18, 24}, Error = 180

Cluster 2 : {30, 42, 48}, Error = 168

Thus, total error = 348

b) Do both sets of centroids represent stable solutions?

Yes, they do represent stable solutions since the above centroids represent centroids that are very far apart.

c) What are the 2 clusters produced by single link?

The minimal difference in (i) is 42-30 = 12. The minimal difference in (ii) is 42-30 = 6. So the two clusters formed by a single link is {6, 12, 18, 24, 30} & {42, 48}.

d) Which technique, K-means or single link, seems to produce the "most natural" clustering in this situation?

Since MIN usually produces the most natural clustering, I would go with MIN (single link).

e) What definition(s) of clustering does this natural clustering correspond to?

MIN produces continguous clsuters.

f) What well-known characteristic of the K-means alg. explains the previous behavior?

From what I recall, the K-means alg. is weak towards finding clusters that have a variety in sizes, or when not well-separated. The objective of minimizing squared error leads it to breaking the larger cluster, thus, producing the unnatural one in this case.

1.6 Exercise 21

Compute the entropy and purity for the confusion matrix...

Cluster #1:

Entropy = -[($\frac{1}{693}$)log($\frac{1}{693}$) + ($\frac{1}{693}$)log($\frac{1}{693}$) + ($\frac{0}{693}$)log($\frac{0}{693}$) + ($\frac{11}{693}$)log($\frac{11}{693}$) + ($\frac{4}{693}$)log($\frac{4}{693}$) + ($\frac{676}{693}$)log($\frac{676}{693}$)] = 0.199 = 0.2, Purity = $\frac{676}{693}$ = 0.975 = 0.98

Cluster #2 :

Entropy = 1.84, Purity = 0.53

Cluster #3:

Entropy = 1.7, Purity = 0.49

Total:

Entropy = 1.44, 0.61

1.7 Exercise 22

Given 2 sets of 100 points that fall within the unit square. One set of points is arranged so that the points are uniormly spaced. The other set of points is generated from a uniform distribution over the unit square.

a) Is there a difference between the 2 set of points?

   Definitely, the random points will have a region of less & more density, while the uniformly spaced will have uniform density.

b) If so, which set of points will typically have a smaller SSE for K=10 clusters?

   The random generated will have smaller SSE for K=10 clusters.

c) What will be the behavior of DBSCAN on the uniform data set? The random data set?

   Depending on the threshold, DBSCAN will either merge all the points in the uniform data set into a cluster or state they are all just noise. In terms of the random data set, DBSCAN can often find clusters in random data due to the variety of density between regions.

2 Practicum Problems

2.1 Problem 1 - Auto-Mpg Dataset

[163]:
```
#All necessary imports
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from sklearn.impute import SimpleImputer
#from sklearn.datasets.samples_generator import make_blobs
#from sklearn.cluster import KMeans
from sklearn.cluster import AgglomerativeClustering
import scipy.cluster.hierarchy as sch
```

[164]:
```
#Load the auto-mpg sample dataset
autompg_ds = pd.read_csv('auto-mpg.csv', na_values=["?"])
autompg_ds = autompg_ds.drop(columns=['name', 'cylinders', 'model_year',␣
↪'origin'])
```

[165]:
```
#Check to make sure the dataset works
autompg_ds.describe()
```

[165]:
```
              mpg  displacement  horsepower       weight  acceleration
count  398.000000    398.000000  392.000000   398.000000    398.000000
mean    23.514573    193.425879  104.469388  2970.424623     15.568090
std      7.815984    104.269838   38.491160   846.841774      2.757689
min      9.000000     68.000000   46.000000  1613.000000      8.000000
25%     17.500000    104.250000   75.000000  2223.750000     13.825000
50%     23.000000    148.500000   93.500000  2803.500000     15.500000
```

```
75%     29.000000    262.000000  126.000000  3608.000000     17.175000
max     46.600000    455.000000  230.000000  5140.000000     24.800000
```

[166]:
```
#Impute any missing values with the mean of the dataset
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
imp_mean.fit(autompg_ds)
autompg_ds[autompg_ds.columns] = imp_mean.fit_transform(autompg_ds)
```

[167]:
```
#Check the dataset again
autompg_ds.describe()
```

[167]:
```
              mpg  displacement  horsepower       weight  acceleration
count  398.000000    398.000000  398.000000   398.000000    398.000000
mean    23.514573    193.425879  104.469388  2970.424623     15.568090
std      7.815984    104.269838   38.199187   846.841774      2.757689
min      9.000000     68.000000   46.000000  1613.000000      8.000000
25%     17.500000    104.250000   76.000000  2223.750000     13.825000
50%     23.000000    148.500000   95.000000  2803.500000     15.500000
75%     29.000000    262.000000  125.000000  3608.000000     17.175000
max     46.600000    455.000000  230.000000  5140.000000     24.800000
```

[168]:
```
#Perform Hierarchial Clustering with linkage set to average & default affinity␣
 ↪set to a euclidean.
#Remaining parameters must obtain a shallow tree with 3 clusters as targets

clustering = AgglomerativeClustering(n_clusters=3, affinity='euclidean',␣
 ↪linkage='average').fit(autompg_ds)
```

[169]:
```
labels = clustering.labels_
print(labels)
```

```
[2 2 2 2 2 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 2 2 0
 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 2 1 1 1 1 0 2 1
 1 1 0 0 0 0 0 0 0 0 0 1 2 2 1 2 1 1 1 1 1 1 2 0 0 0 0 0 0 1 1 1 1 0 0 0 0
 0 0 0 0 1 1 0 0 0 0 2 0 0 2 0 0 0 2 0 0 0 0 2 2 2 1 1 1 1 1 0 0 0 0 0 0 0
 0 0 0 0 0 2 2 0 1 1 1 1 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 1 2 1 0 2 0 0 0 0 0 0 2 2 2 0 0 0 0 0 0 2 0 0 2 1 1 2 2 0 0 0 0 0 2
 1 1 1 2 2 2 2 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 0 0 0 2 0 2
 0 2 2 2 2 0 1 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 2 2 2 2 2 1 1 2 2 0 0 0
 0 2 2 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 0 2 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

[170]:
```
#To find mean and variance of each cluster, let's place the labels into the␣
 ↪dataframe which range from 0 to 2
autompg_ds['cluster'] = clustering.labels_
autompg_ds.describe()
```

```
[170]:               mpg  displacement  horsepower       weight  acceleration  \
       count  398.000000    398.000000  398.000000   398.000000    398.000000
       mean    23.514573    193.425879  104.469388  2970.424623     15.568090
       std      7.815984    104.269838   38.199187   846.841774      2.757689
       min      9.000000     68.000000   46.000000  1613.000000      8.000000
       25%     17.500000    104.250000   76.000000  2223.750000     13.825000
       50%     23.000000    148.500000   95.000000  2803.500000     15.500000
       75%     29.000000    262.000000  125.000000  3608.000000     17.175000
       max     46.600000    455.000000  230.000000  5140.000000     24.800000

                 cluster
       count  398.000000
       mean     0.502513
       std      0.770190
       min      0.000000
       25%      0.000000
       50%      0.000000
       75%      1.000000
       max      2.000000
```

```
[171]: #This is the description for the first cluster
       c0 = autompg_ds.loc[autompg_ds['cluster'] == 0]
       c0.describe()
```

```
[171]:               mpg  displacement  horsepower       weight  acceleration  \
       count  266.000000    266.000000  266.000000   266.000000    266.000000
       mean    27.365414    131.934211   84.300061  2459.511278     16.298120
       std      6.478913     53.179727   19.213107   427.354771      2.391296
       min     13.000000     68.000000   46.000000  1613.000000     10.000000
       25%     22.075000     97.000000   70.000000  2124.250000     14.550000
       50%     26.900000    119.000000   85.000000  2395.000000     16.000000
       75%     32.000000    149.750000   95.000000  2805.250000     17.600000
       max     46.600000    455.000000  225.000000  3302.000000     24.800000

              cluster
       count    266.0
       mean       0.0
       std        0.0
       min        0.0
       25%        0.0
       50%        0.0
       75%        0.0
       max        0.0
```

```
[172]: #This is the description for the second cluster
       c1 = autompg_ds.loc[autompg_ds['cluster'] == 1]
       c1.describe()
```

[172]:
```
             mpg  displacement  horsepower       weight  acceleration  cluster
count  64.000000     64.000000   64.000000    64.000000     64.000000     64.0
mean   13.889062    358.093750  167.046875  4398.593750     13.025000      1.0
std     1.832781     46.240818   27.504937   272.602899      1.895106      0.0
min     9.000000    260.000000  110.000000  4042.000000      8.500000      1.0
25%    13.000000    318.000000  149.750000  4183.750000     12.000000      1.0
50%    14.000000    350.000000  156.500000  4357.000000     13.000000      1.0
75%    15.125000    400.000000  180.000000  4530.250000     14.000000      1.0
max    17.500000    455.000000  230.000000  5140.000000     19.000000      1.0
```

[173]:
```
#This is the description for the third cluster
c2 = autompg_ds.loc[autompg_ds['cluster'] == 2]
c2.describe()
```

[173]:
```
             mpg  displacement  horsepower       weight  acceleration  cluster
count  68.000000     68.000000   68.000000    68.000000     68.000000     68.0
mean   17.510294    278.985294  124.470588  3624.838235     15.105882      2.0
std     2.971513     53.688847   26.703720   194.359999      3.249151      0.0
min    11.000000    163.000000   72.000000  3329.000000      8.000000      2.0
25%    15.000000    231.000000  105.000000  3435.250000     12.725000      2.0
50%    17.550000    260.000000  120.000000  3616.500000     15.450000      2.0
75%    19.125000    318.000000  150.000000  3782.000000     17.250000      2.0
max    26.600000    400.000000  190.000000  3988.000000     22.200000      2.0
```

[174]:
```
#To find a difference if we had use the feature 'origin' as a class label,␣
 ↪let's create another dataset
autompg_ds2 = pd.read_csv('auto-mpg.csv', na_values=["?"])
autompg_ds2 = autompg_ds2.drop(columns=['name','cylinders', 'model_year'])
autompg_ds2.describe()
```

[174]:
```
              mpg  displacement  horsepower       weight  acceleration  \
count  398.000000    398.000000  392.000000   398.000000    398.000000
mean    23.514573    193.425879  104.469388  2970.424623     15.568090
std      7.815984    104.269838   38.491160   846.841774      2.757689
min      9.000000     68.000000   46.000000  1613.000000      8.000000
25%     17.500000    104.250000   75.000000  2223.750000     13.825000
50%     23.000000    148.500000   93.500000  2803.500000     15.500000
75%     29.000000    262.000000  126.000000  3608.000000     17.175000
max     46.600000    455.000000  230.000000  5140.000000     24.800000

           origin
count  398.000000
mean     1.572864
std      0.802055
min      1.000000
25%      1.000000
50%      1.000000
```

```
75%        2.000000
max        3.000000
```

[175]:
```
#Impute any missing values with the mean of the dataset
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
imp_mean.fit(autompg_ds2)
autompg_ds2[autompg_ds2.columns] = imp_mean.fit_transform(autompg_ds2)
```

[176]:
```
#This is the description for the second cluster
o1 = autompg_ds2.loc[autompg_ds2['origin'] == 1]
o1.describe()
```

[176]:
```
              mpg  displacement  horsepower       weight  acceleration  origin
count  249.000000    249.000000  249.000000   249.000000    249.000000   249.0
mean    20.083534    245.901606  118.814769  3361.931727     15.033735     1.0
std      6.402892     98.501839   39.617323   794.792506      2.751112     0.0
min      9.000000     85.000000   52.000000  1800.000000      8.000000     1.0
25%     15.000000    151.000000   88.000000  2720.000000     13.000000     1.0
50%     18.500000    250.000000  105.000000  3365.000000     15.000000     1.0
75%     24.000000    318.000000  150.000000  4054.000000     16.900000     1.0
max     39.000000    455.000000  230.000000  5140.000000     22.200000     1.0
```

[177]:
```
#This is the description for the third cluster
o2 = autompg_ds2.loc[autompg_ds2['origin'] == 2]
o2.describe()
```

[177]:
```
             mpg  displacement  horsepower       weight  acceleration  origin
count  70.000000     70.000000   70.000000    70.000000     70.000000    70.0
mean   27.891429    109.142857   81.241983  2423.300000     16.787143     2.0
std     6.723930     22.582079   20.264743   490.043191      3.045687     0.0
min    16.200000     68.000000   46.000000  1825.000000     12.200000     2.0
25%    24.000000     92.250000   70.000000  2067.250000     14.500000     2.0
50%    26.500000    104.500000   77.500000  2240.000000     15.700000     2.0
75%    30.650000    121.000000   90.750000  2769.750000     18.900000     2.0
max    44.300000    183.000000  133.000000  3820.000000     24.800000     2.0
```

[178]:
```
#This is the description for the third cluster
o3 = autompg_ds2.loc[autompg_ds2['origin'] == 3]
o3.describe()
```

[178]:
```
             mpg  displacement  horsepower       weight  acceleration  origin
count  79.000000     79.000000   79.000000    79.000000     79.000000    79.0
mean   30.450633    102.708861   79.835443  2221.227848     16.172152     3.0
std     6.090048     23.140126   17.819199   320.497248      1.954937     0.0
min    18.000000     70.000000   52.000000  1613.000000     11.400000     3.0
25%    25.700000     86.000000   67.000000  1985.000000     14.600000     3.0
50%    31.600000     97.000000   75.000000  2155.000000     16.400000     3.0
```

```
75%    34.050000   119.000000   95.000000  2412.500000     17.550000    3.0
max    46.600000   168.000000  132.000000  2930.000000     21.000000    3.0
```

To answer the final question, I do not see a clear relationship between the cluster assigment and the class labels...The data seems to be pretty far apart or barely different in different attributes when comparing.

2.2 Problem 2 - Boston Dataset

[101]:
```
#Imports
from sklearn.cluster import KMeans
from sklearn.datasets import load_boston
from sklearn import preprocessing
from sklearn import metrics
```

[104]:
```
#Load the dataset into a dataframe
boston_ds = load_boston()
boston_df = pd.DataFrame(boston_ds.data, columns=boston_ds.feature_names)
boston_df.describe()
```

[104]:
```
             CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std      8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min      0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%      0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%      0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%      3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max     88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

              AGE         DIS         RAD         TAX     PTRATIO           B  \
count  506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean    68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std     28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%     77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%     94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            LSTAT
count  506.000000
mean    12.653063
std      7.141062
min      1.730000
25%      6.950000
50%     11.360000
```

```
75%      16.955000
max      37.970000
```

[126]:
```
#Scale the dataset
df_scaled = pd.DataFrame(data=preprocessing.scale(boston_ds.data),␣
 ↪columns=boston_ds.feature_names)
df_scaled.describe()
```

[126]:
```
               CRIM            ZN         INDUS          CHAS           NOX  \
count  5.060000e+02  5.060000e+02  5.060000e+02  5.060000e+02  5.060000e+02
mean  -8.787437e-17 -6.343191e-16 -2.682911e-15  4.701992e-16  2.490322e-15
std    1.000990e+00  1.000990e+00  1.000990e+00  1.000990e+00  1.000990e+00
min   -4.197819e-01 -4.877224e-01 -1.557842e+00 -2.725986e-01 -1.465882e+00
25%   -4.109696e-01 -4.877224e-01 -8.676906e-01 -2.725986e-01 -9.130288e-01
50%   -3.906665e-01 -4.877224e-01 -2.110985e-01 -2.725986e-01 -1.442174e-01
75%    7.396560e-03  4.877224e-02  1.015999e+00 -2.725986e-01  5.986790e-01
max    9.933931e+00  3.804234e+00  2.422565e+00  3.668398e+00  2.732346e+00

                 RM           AGE           DIS           RAD           TAX  \
count  5.060000e+02  5.060000e+02  5.060000e+02  5.060000e+02  5.060000e+02
mean  -1.145230e-14 -1.407855e-15  9.210902e-16  5.441409e-16 -8.868619e-16
std    1.000990e+00  1.000990e+00  1.000990e+00  1.000990e+00  1.000990e+00
min   -3.880249e+00 -2.335437e+00 -1.267069e+00 -9.828429e-01 -1.313990e+00
25%   -5.686303e-01 -8.374480e-01 -8.056878e-01 -6.379618e-01 -7.675760e-01
50%   -1.084655e-01  3.173816e-01 -2.793234e-01 -5.230014e-01 -4.646726e-01
75%    4.827678e-01  9.067981e-01  6.623709e-01  1.661245e+00  1.530926e+00
max    3.555044e+00  1.117494e+00  3.960518e+00  1.661245e+00  1.798194e+00

            PTRATIO             B         LSTAT
count  5.060000e+02  5.060000e+02  5.060000e+02
mean  -9.205636e-15  8.163101e-15 -3.370163e-16
std    1.000990e+00  1.000990e+00  1.000990e+00
min   -2.707379e+00 -3.907193e+00 -1.531127e+00
25%   -4.880391e-01  2.050715e-01 -7.994200e-01
50%    2.748590e-01  3.811865e-01 -1.812536e-01
75%    8.065758e-01  4.336510e-01  6.030188e-01
max    1.638828e+00  4.410519e-01  3.548771e+00
```

[112]:
```
#Perform K-Means with the scaled data & number of clusters = 2
clust_model = KMeans(n_clusters=2, init='k-means++')
clust_labels = clust_model.fit_predict(df_scaled)
print(clust_labels)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 1 1 0 1 1 1 1 1 0 1 0 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

[115]:
```python
#Provide the Silhouette
silhouette_avg = metrics.silhouette_score(df_scaled, clust_labels)
print(silhouette_avg)
```

```
0.36011768587358606
```

[116]:
```python
#Perform K-Means with the scaled data & number of clusters = 3
clust_model = KMeans(n_clusters=3, init='k-means++')
clust_labels = clust_model.fit_predict(df_scaled)
print(clust_labels)
```

```
[2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 1 1 1 1 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 1 2 2 2 2
 2 2 2 2 2 2 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 2 2 2 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 2 2 1 2 1 1 1 1 1 1 1 1 1 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2
 2 2 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1
 2 2 2 2 2 2 2 2 1 2 1 1 2 2 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

[117]:
```python
#Provide the Silhouette
silhouette_avg = metrics.silhouette_score(df_scaled, clust_labels)
print(silhouette_avg)
```

```
0.2574894522739469
```

[118]:
```python
#Perform K-Means with the scaled data & number of clusters = 4
clust_model = KMeans(n_clusters=4, init='k-means++')
clust_labels = clust_model.fit_predict(df_scaled)
print(clust_labels)
```

```
[3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 1 1 1 1 1 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 1 3 1 3 3 3 3
 3 3 3 3 3 3 1 3 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 3 0 3 3 3 3 0 3 0 3 0 0 0 0 2 0 0 0 0 0
 0 0 0 0 2 0 2 2 0 3 3 3 2 3 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 2 2 2 2 2 3 3 3 2 3 2 2 2 2
 2 3 3 3 3 3 3 3 3 3 3 3 2 3 2 3 1 1 1 1 1 1 3 3 1 3 1 1 1 1 1 1 1 1 1 3 3
 3 3 3 3 3 3 3 3 3 3 2 3 1 3 2 2 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 3 3 3
 3 3 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1
 3 3 3 3 3 3 3 3 1 3 1 1 3 3 1 1 1 1 1 1 1 1 1 2 2 2 0 0 0 0 2 2 0 0 0 0 2
 2 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3]
```

[119]:
```
#Provide the Silhouette
silhouette_avg = metrics.silhouette_score(df_scaled, clust_labels)
print(silhouette_avg)
```

```
0.2809804562187518
```

[122]:
```
#Perform K-Means with the scaled data & number of clusters = 5
clust_model = KMeans(n_clusters=5, init='k-means++')
clust_labels = clust_model.fit_predict(df_scaled)
print(clust_labels)
```

```
[1 1 1 1 1 1 1 1 4 1 1 1 1 1 4 1 1 4 1 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 1
 1 1 3 3 1 1 1 1 1 1 1 4 1 1 1 1 1 3 3 3 3 1 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 1 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4
 4 4 4 4 2 4 2 2 4 4 4 4 2 4 2 2 4 4 4 4 4 4 4 4 4 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 4 2 2 2 2 2 1 4 1 2 4 2 2 2 2
 2 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 3 1 1 1 1 3 1 1 1 1 1 1 1 1 3 3 3 3 3 1 1
 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 2 1 2 2 1 1 1 1 2 3 3 3 3 3 3 3 3 3 3 1 1 1
 1 1 3 3 3 1 3 3 1 1 1 1 1 4 4 1 4 1 1 4 4 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3
 1 1 1 1 1 1 1 1 3 1 3 3 1 1 3 3 3 3 3 3 3 3 3 2 2 2 0 0 0 0 2 2 0 0 0 0 2
 2 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 4 4 0 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
```

[123]:
```
#Provide the Silhouette
silhouette_avg = metrics.silhouette_score(df_scaled, clust_labels)
print(silhouette_avg)
```

```
0.24783915229552517
```

[124]:
```python
#Perform K-Means with the scaled data & number of clusters = 6
clust_model = KMeans(n_clusters=6, init='k-means++')
clust_labels = clust_model.fit_predict(df_scaled)
print(clust_labels)
```

```
[4 4 4 4 4 4 4 4 5 4 4 4 4 4 5 4 4 5 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 4
 4 4 1 1 4 4 4 4 4 4 4 5 4 4 4 4 4 1 1 1 1 4 4 4 4 4 4 4 1 1 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 4 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 2 5 5 5 5 5
 5 5 5 5 2 5 2 2 5 5 5 5 2 5 2 2 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 4 4 5 2 2 2 2 2 4 5 4 2 5 2 2 2 2
 2 4 4 4 4 4 4 4 4 4 4 4 2 4 2 4 1 4 4 4 4 1 4 4 4 4 4 4 4 4 1 1 1 1 1 4 4
 4 4 4 4 4 4 4 4 4 4 2 4 4 4 2 2 4 2 2 4 4 4 4 2 1 1 1 1 1 1 1 1 1 1 4 4 4
 4 4 1 1 1 4 1 1 4 4 4 4 4 5 5 4 5 5 5 5 5 5 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1
 4 4 4 4 4 4 4 4 1 4 1 1 4 4 1 1 1 1 1 1 1 1 1 2 2 2 0 0 0 0 2 2 0 0 3 0 2
 2 0 2 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 0
 0 0 3 3 3 3 3 3 3 3 3 3 3 0 0 0 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 0 0 0 0 0
 0 3 0 0 0 0 3 0 0 0 3 3 3 3 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 5 5 5 5 5 5 5 5 5 5 5 5 5]
```

[125]:
```python
#Provide the Silhouette
silhouette_avg = metrics.silhouette_score(df_scaled, clust_labels)
print(silhouette_avg)
```

```
0.2617743548302116
```

Since the highest silouette score was when the total # of clusters of 2, it is the optimal k because when a high value is shown, it indicates that the object is well matched to its own cluster & poorly matched to neighboring clusters.

[131]:
```python
#Redo the cluster with n_clusters = 2 to calucate the mean of all values
clust_model = KMeans(n_clusters=2, init='k-means++')
clust_labels = clust_model.fit_predict(df_scaled)
print(clust_labels)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 0 0 1 0 0 0 0 0 1 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

[132]:
```
#Add the labels to the df_scaled
df_scaled['cluster'] = clust_labels
df_scaled.describe()
```

[132]:
```
               CRIM            ZN         INDUS          CHAS           NOX  \
count  5.060000e+02  5.060000e+02  5.060000e+02  5.060000e+02  5.060000e+02
mean  -8.787437e-17 -6.343191e-16 -2.682911e-15  4.701992e-16  2.490322e-15
std    1.000990e+00  1.000990e+00  1.000990e+00  1.000990e+00  1.000990e+00
min   -4.197819e-01 -4.877224e-01 -1.557842e+00 -2.725986e-01 -1.465882e+00
25%   -4.109696e-01 -4.877224e-01 -8.676906e-01 -2.725986e-01 -9.130288e-01
50%   -3.906665e-01 -4.877224e-01 -2.110985e-01 -2.725986e-01 -1.442174e-01
75%    7.396560e-03  4.877224e-02  1.015999e+00 -2.725986e-01  5.986790e-01
max    9.933931e+00  3.804234e+00  2.422565e+00  3.668398e+00  2.732346e+00

                 RM           AGE           DIS           RAD           TAX  \
count  5.060000e+02  5.060000e+02  5.060000e+02  5.060000e+02  5.060000e+02
mean  -1.145230e-14 -1.407855e-15  9.210902e-16  5.441409e-16 -8.868619e-16
std    1.000990e+00  1.000990e+00  1.000990e+00  1.000990e+00  1.000990e+00
min   -3.880249e+00 -2.335437e+00 -1.267069e+00 -9.828429e-01 -1.313990e+00
25%   -5.686303e-01 -8.374480e-01 -8.056878e-01 -6.379618e-01 -7.675760e-01
50%   -1.084655e-01  3.173816e-01 -2.793234e-01 -5.230014e-01 -4.646726e-01
75%    4.827678e-01  9.067981e-01  6.623709e-01  1.661245e+00  1.530926e+00
max    3.555044e+00  1.117494e+00  3.960518e+00  1.661245e+00  1.798194e+00

            PTRATIO             B         LSTAT     cluster
count  5.060000e+02  5.060000e+02  5.060000e+02  506.000000
mean  -9.205636e-15  8.163101e-15 -3.370163e-16    0.349802
std    1.000990e+00  1.000990e+00  1.000990e+00    0.477379
min   -2.707379e+00 -3.907193e+00 -1.531127e+00    0.000000
25%   -4.880391e-01  2.050715e-01 -7.994200e-01    0.000000
50%    2.748590e-01  3.811865e-01 -1.812536e-01    0.000000
75%    8.065758e-01  4.336510e-01  6.030188e-01    1.000000
max    1.638828e+00  4.410519e-01  3.548771e+00    1.000000
```

[134]:
```
#Show the mean values for all features in the first cluster
c0 = df_scaled.loc[df_scaled['cluster'] == 0]
c0.mean()
```

[134]:
```
CRIM      -0.390124
ZN         0.262392
INDUS     -0.620368
CHAS       0.002912
NOX       -0.584675
RM         0.243315
AGE       -0.435108
```

```
DIS         0.457222
RAD        -0.583801
TAX        -0.631460
PTRATIO    -0.285808
B           0.326451
LSTAT      -0.446421
cluster     0.000000
dtype: float64
```

[135]:
```python
#Show the mean values for all features in the second cluster
c1 = df_scaled.loc[df_scaled['cluster'] == 1]
c1.mean()
```

[135]:
```
CRIM        0.725146
ZN         -0.487722
INDUS       1.153113
CHAS       -0.005412
NOX         1.086769
RM         -0.452263
AGE         0.808760
DIS        -0.849865
RAD         1.085145
TAX         1.173731
PTRATIO     0.531248
B          -0.606793
LSTAT       0.829787
cluster     1.000000
dtype: float64
```

To answer the final question for this problem, it is important to remember that the mean of a cluster is the same as the centroid coordinate.

2.3 Problem 3 - Wine Dataset

[138]:
```python
#Imports
from sklearn.datasets import load_wine
```

[139]:
```python
#Load the dataset
wine_ds = load_wine()
wine_scaled = pd.DataFrame(data=preprocessing.scale(wine_ds.data),␣
 ↪columns=wine_ds.feature_names)
wine_scaled.describe()
```

[139]:
```
            alcohol    malic_acid           ash  alcalinity_of_ash  \
count  1.780000e+02  1.780000e+02  1.780000e+02       1.780000e+02
mean   7.841418e-15  2.444986e-16 -4.059175e-15      -7.110417e-17
std    1.002821e+00  1.002821e+00  1.002821e+00       1.002821e+00
```

```
min   -2.434235e+00 -1.432983e+00 -3.679162e+00      -2.671018e+00
25%   -7.882448e-01 -6.587486e-01 -5.721225e-01      -6.891372e-01
50%    6.099988e-02 -4.231120e-01 -2.382132e-02       1.518295e-03
75%    8.361286e-01  6.697929e-01  6.981085e-01       6.020883e-01
max    2.259772e+00  3.109192e+00  3.156325e+00       3.154511e+00

          magnesium  total_phenols    flavanoids  nonflavanoid_phenols  \
count  1.780000e+02   1.780000e+02  1.780000e+02          1.780000e+02
mean  -2.494883e-17  -1.955365e-16  9.443133e-16         -4.178929e-16
std    1.002821e+00   1.002821e+00  1.002821e+00          1.002821e+00
min   -2.088255e+00  -2.107246e+00 -1.695971e+00         -1.868234e+00
25%   -8.244151e-01  -8.854682e-01 -8.275393e-01         -7.401412e-01
50%   -1.222817e-01   9.595986e-02  1.061497e-01         -1.760948e-01
75%    5.096384e-01   8.089974e-01  8.490851e-01          6.095413e-01
max    4.371372e+00   2.539515e+00  3.062832e+00          2.402403e+00

       proanthocyanins  color_intensity           hue  \
count     1.780000e+02     1.780000e+02  1.780000e+02
mean     -1.540590e-15    -4.129032e-16  1.398382e-15
std       1.002821e+00     1.002821e+00  1.002821e+00
min      -2.069034e+00    -1.634288e+00 -2.094732e+00
25%      -5.972835e-01    -7.951025e-01 -7.675624e-01
50%      -6.289785e-02    -1.592246e-01  3.312687e-02
75%       6.291754e-01     4.939560e-01  7.131644e-01
max       3.485073e+00     3.435432e+00  3.301694e+00

       od280/od315_of_diluted_wines       proline
count                  1.780000e+02  1.780000e+02
mean                   2.126888e-15 -6.985673e-17
std                    1.002821e+00  1.002821e+00
min                   -1.895054e+00 -1.493188e+00
25%                   -9.522483e-01 -7.846378e-01
50%                    2.377348e-01 -2.337204e-01
75%                    7.885875e-01  7.582494e-01
max                    1.960915e+00  2.971473e+00
```

[140]:
```
clust_model = KMeans(n_clusters=3, init='k-means++')
clust_labels = clust_model.fit_predict(wine_scaled)
print(clust_labels)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0 0 0 0 0 0 0 0 1
 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 2 0 0 1 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

```
[160]: #Obtain class labels
       target_values = wine_ds.target
       target_labels = pd.qcut(x=target_values, q=2, labels=[1,2])
       class_labels = target_labels.astype('int32') - 1
       print(class_labels)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

```
[161]: #Get the homogenity & completeness
       h_score = metrics.homogeneity_score(class_labels, clust_labels)
       c_score = metrics.completeness_score(class_labels, clust_labels)

       print(h_score, c_score)
```

```
0.8900387051166616 0.4745258086227089
```

To answer the final question, homogeneity means all of the observations witht he same class label are within the same cluster. Completeness means all members of the same class are within the same cluster. Both scores range from 0 to 1, with the higher # being the best outcome.

```
[ ]:
```