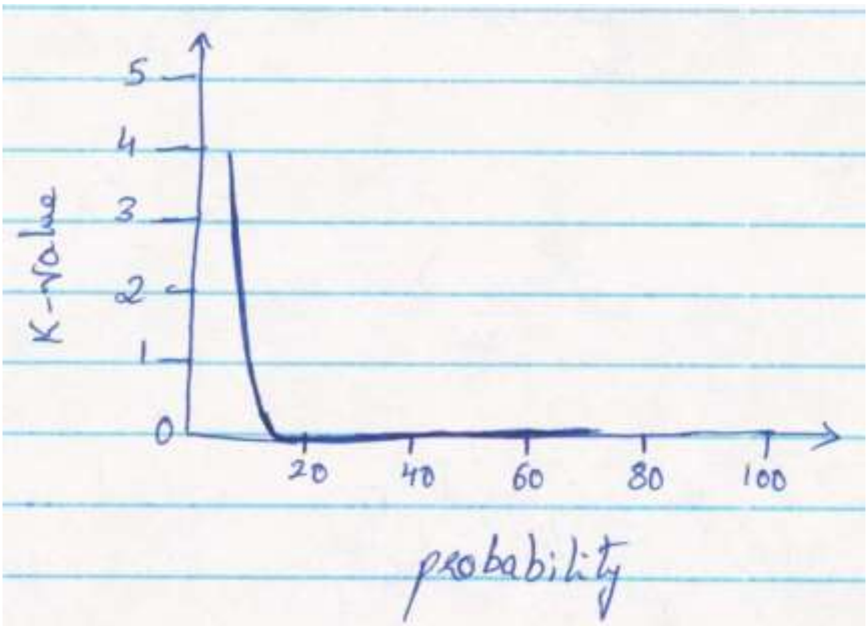
Homework#4

Recitation Exercises:

Problem 4:

a) Probability = number of ways to select one centroid from each cluster/ # of ways of selecting K clusters
P = factorial(k)/k^k




b) Probability that a sample of size 2K contains at least one points from each cluster.
P = 2 * factorial(k)/k^k
For K = 10
P = 2*factorial(10)/10^10 = 0.000728

For k = 100, p = 1.867*10^-42

For k = 1000, p = near to zero (0)

Problem 7:

Let first see the squared error function:

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist(c_i, x)^2$$

dist = standard Euclidean distance between two objects in Euclidean space. One easy way to reduce SSE is to increase K, the number of clusters.

**Option C** seems good reason that less dense region is normally consisting of noise and outliers. Also more closer the points higher the density. So, the result will be reduced SSE.

Problem 11:

if the SSE for one variable is low for all clusters?
- Then we have a variable which is constant and that has little use in dividing the data into group on clusters.

Low for just one cluster?
- It is the best case possible for any cluster as this attribute helps in defining this cluster.

High for all clusters?
- Then our data is scattered far away and contain only noise.

High for just one cluster?
- Then it is an odd with the information provided by the attributes with low SSE that define the cluster. This means that cluster defined by this attribute are different from those defined by other attributes but it does not help define the cluster.

How could you use the per variable SSE information to improve your clustering?
- The idea is to eliminate attributes that have poor distinguishing power between clusters. The attributes with high SSE are troublesome if they have a relatively high SSE with respect to other attributes since they introduce a lot of noise into the computation of the overall SSE.

Problem 16:

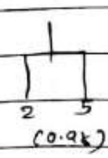
Table:

| | **p1** | **p2** | **p3** | **p4** | **p5** |
|---|---|---|---|---|---|
| p1 | 1.00 | 0.10 | 0.41 | 0.55 | 0.35 |
| p2 | 0.10 | 1.00 | 0.64 | 0.47 | 0.98 |
| p3 | 0.41 | 0.64 | 1.00 | 0.44 | 0.85 |
| p4 | 0.55 | 0.47 | 0.44 | 1.00 | 0.76 |
| p5 | 0.35 | 0.98 | 0.85 | 0.76 | 1.00 |

Single Link :

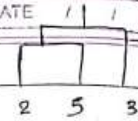
First, we will cluster P2 & P5 together since (similarity) distance is the highest.

| | P1 | P2, P5 | P3 | P4 |
|---|---|---|---|---|
| P1 | 1.00 | 0.35 | 0.41 | 0.55 |
| P2, P5 | 0.35 | 1.00 | 0.85 | 0.76 |
| P3 | 0.41 | 0.85 | 1.00 | 0.44 |
| P4 | 0.55 | 0.76 | 0.44 | 1.00 |



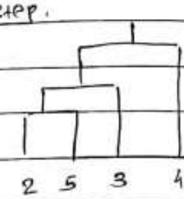

Next, cluster (P2, P5) with P3

|  | $P_1$ | $P_2, P_5, P_3$ | $P_4$ |
|---|---|---|---|
| $P_1$ | 1.00 | 0.41 | 0.55 |
| $P_2, P_5, P_3$ | 0.41 | 1.00 | 0.76 |
| $P_4$ | 0.55 | 0.76 | 1.00 |

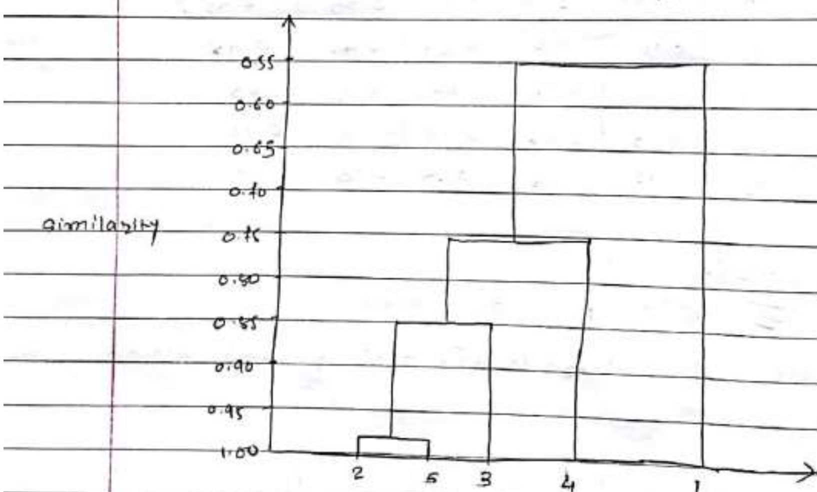



cluster, $(P_2, P_5), P_3$ with $P_4$ in the next step.

|  | $P_1$ | $P_2, P_5, P_3, P_4$ |
|---|---|---|
| $P_1$ | 1.00 | 0.55 |
| $P_2, P_5, P_3, P_4$ | 0.55 | 1.00 |




1 will be the last to follow.

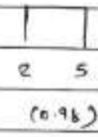



Single link dendrogram

⇒ Complete Link:~

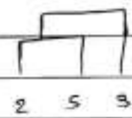
Initially, join $P_2$ & $P_5$ to form a Cluster.

| | $P_1$ | $P_2,P_5$ | $P_3$ | $P_4$ |
|---|---|---|---|---|
| $P_1$ | 1.00 | 0.10 | 0.41 | 0.55 |
| $P_2,P_5$ | 0.10 | 1.00 | 0.64 | 0.47 |
| $P_3$ | 0.41 | 0.64 | 1.00 | 0.44 |
| $P_4$ | 0.55 | 0.47 | 0.44 | 1.00 |



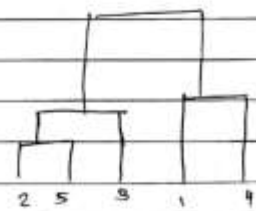

Next, cluster $P_2,P_5$ with $P_3$.

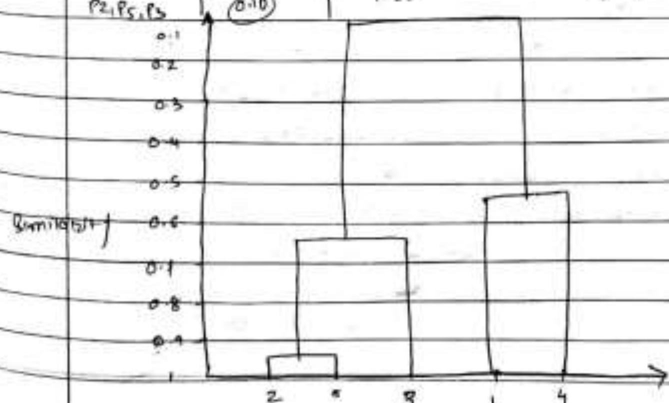
| | $P_1$ | $P_2,P_5,P_3$ | $P_4$ |
|---|---|---|---|
| $P_1$ | 1.00 | 0.10 | 0.55 |
| $P_2,P_5,P_3$ | 0.10 | 1.00 | 0.44 |
| $P_4$ | 0.55 | 0.44 | 1.00 |




Clusters $P_1$ & $P_4$ together.

| | $P_1,P_4$ | $P_2,P_5,P_3$ |
|---|---|---|
| $P_1,P_4$ | 1.00 | 0.10 |
| $P_2,P_5,P_3$ | 0.10 | 1.00 |




Problem 17:

a) Data set: {6, 12, 18, 24, 30, 42, 48}

1) Centroids: {18, 45}

So, for 1st centroid 18 our cluster will be = {6, 12, 18, 24, 30}

SSE = (18-6)^2 + (18-12)^2 + (18-18)^2 + (18-24)^2 + (18-30)^2 = **360**

For $2^{nd}$ centroid 45 our cluster will be = {42, 48}

SSE = 18

Total SSE = 360 + 18 = **378**

2) Centroids: {15, 40}
$1^{st}$ SSE = 180
$2^{nd}$ SSE = 168
Total SSE = 180 +168 = **348**

b) I believe both centroid {18, 45} and {15, 40} represent stable solution. I do not think so we need any changes in the cluster generation.

c) Two cluster produced by single link are {6, 12, 18, 24, 30} and {42, 48}

d) MIN produces the most natural clustering in this solution that define cluster proximity as the proximity between the closest 2 points that are different clusters.

e) This natural clustering corresponds to MIN that produce contiguous cluster. But center based is also taken as one of center given desired clusters.

f) K mean is not good at finding clusters of different sizes at least when they are not well separated. The reason for this is that the objective of minimizing squared error causes it to break the large cluster.

Problem 21:

Entropy: entropy of a cluster ei = - sum [(Pij)*log2(Pij)]

Overall entropy e = sum[mi/m*ei]

Entropy_1 = -[1/693*log2(1/693) + 1/693*log2(1/693) + 0 + 11/693*log2(11/693) + 4/693*log2(4/693) + 676/693*log2(676/693)

Entropy_1 = 0.20

Entropy_2 = 1.84

Entropy_3 = 1.70

Total Entropy = **1.44**

Purity: purity of a cluster Pi = max(Pij)

Overall purity = sum(mi/m*Pi)

Purity_1 = 676/693 = 0.98

Purity _2 = 827/1562 = 0.53

Purity_3 = 465/949 = 0.49

Total Purity = **0.61**

Problem 22:

a) Yes, the set of points that are uniformly spaced but have random arrangements will have regions of lesser or greater density. While the other set of points which are uniformly distributed over the unit square will have uniform density.

b) For K = 10 cluster the uniformly distributed points will have a smaller SSE, because the points will be equally separated from the mean.

c) DBSCAN will merge all points in one uniform data set into one cluster or classify them all as noise that will depend on its threshold boundary issues may arrive for the points that are at the edge. DBSCAN can find clusters in random data as well as it doesn't have validation in density.

# Problem 1

In [2]:
```
import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import Imputer
from sklearn.cluster import AgglomerativeClustering
```

In [5]:
```
# data import for UIC
dataSet = pd.read_csv("http://archive.ics.uci.edu/ml/machine-learning-database
s/auto-mpg/auto-mpg.data",
                     delim_whitespace = True, header = None,
                     names = ['mpg','cylinders','displacement','horsepower','w
eight','acceleration','model','origin', 'car_name'])
```

In [9]:
```
dataSet.describe()
```

Out[9]:

| | mpg | cylinders | displacement | weight | acceleration | model | |
|---|---|---|---|---|---|---|---|
| **count** | 398.000000 | 398.000000 | 398.000000 | 398.000000 | 398.000000 | 398.000000 | 398 |
| **mean** | 23.514573 | 5.454774 | 193.425879 | 2970.424623 | 15.568090 | 76.010050 | 1.5 |
| **std** | 7.815984 | 1.701004 | 104.269838 | 846.841774 | 2.757689 | 3.697627 | 0.8 |
| **min** | 9.000000 | 3.000000 | 68.000000 | 1613.000000 | 8.000000 | 70.000000 | 1.0 |
| **25%** | 17.500000 | 4.000000 | 104.250000 | 2223.750000 | 13.825000 | 73.000000 | 1.0 |
| **50%** | 23.000000 | 4.000000 | 148.500000 | 2803.500000 | 15.500000 | 76.000000 | 1.0 |
| **75%** | 29.000000 | 8.000000 | 262.000000 | 3608.000000 | 17.175000 | 79.000000 | 2.0 |
| **max** | 46.600000 | 8.000000 | 455.000000 | 5140.000000 | 24.800000 | 82.000000 | 3.0 |

In [12]:
```
dataSet.dtypes
```

Out[12]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight          float64
acceleration    float64
model             int64
origin            int64
car_name         object
dtype: object
```

In [13]:
```
# see all unique values in horsepower because it is a object type col
dataSet.horsepower.unique()
```

Out[13]:
```
array(['130.0', '165.0', '150.0', '140.0', '198.0', '220.0', '215.0',
       '225.0', '190.0', '170.0', '160.0', '95.00', '97.00', '85.00',
       '88.00', '46.00', '87.00', '90.00', '113.0', '200.0', '210.0',
       '193.0', '?', '100.0', '105.0', '175.0', '153.0', '180.0', '110.0',
       '72.00', '86.00', '70.00', '76.00', '65.00', '69.00', '60.00',
       '80.00', '54.00', '208.0', '155.0', '112.0', '92.00', '145.0',
       '137.0', '158.0', '167.0', '94.00', '107.0', '230.0', '49.00',
       '75.00', '91.00', '122.0', '67.00', '83.00', '78.00', '52.00',
       '61.00', '93.00', '148.0', '129.0', '96.00', '71.00', '98.00',
       '115.0', '53.00', '81.00', '79.00', '120.0', '152.0', '102.0',
       '108.0', '68.00', '58.00', '149.0', '89.00', '63.00', '48.00',
       '66.00', '139.0', '103.0', '125.0', '133.0', '138.0', '135.0',
       '142.0', '77.00', '62.00', '132.0', '84.00', '64.00', '74.00',
       '116.0', '82.00'], dtype=object)
```

In [14]:
```
# replace '?' with np.NaN
dataSet['horsepower'] = dataSet['horsepower'].replace('?', np.nan)
```

In [15]:
```
imputer_horserpower = Imputer(missing_values='NaN', strategy='mean', axis=0)

dataSet['horsepower'] = imputer_horserpower.fit_transform(dataSet['horsepower'
].values.reshape(-1,1),dataSet['horsepower'])
```

In [21]:
```
# create new df with continuous variables for the dataset.
contus_df = pd.DataFrame([dataSet.mpg, dataSet.displacement, dataSet.horsepowe
r,dataSet.weight,dataSet.acceleration]).transpose()
```

In [40]:
```
# create a copy
dataSet_copy = pd.DataFrame(dataSet)
dataSet_copy = dataSet_copy.drop(['car_name'], axis=1)
```

In [42]:
```
# clustering
clustering = AgglomerativeClustering(n_clusters=3, affinity='euclidean', memor
y=None, connectivity=None,
                                    compute_full_tree =5, linkage ='average',
pooling_func = 'deprecated')
cluster  = clustering.fit(contus_df)
```

In [43]:
```
#Cluster labels
clustering.fit_predict(contus_df)
```

Out[43]:
```
array([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, 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, 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], dtype=int64)
```

In [47]:
```
dataSet_copy['labels'] = cluster.labels_
dataSet_copy.groupby('origin').mean()
```

Out[47]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | mode |
|---|---|---|---|---|---|---|---|
| **origin** | | | | | | | |
| **1** | 20.083534 | 6.248996 | 245.901606 | 118.814769 | 3361.931727 | 15.033735 | 75.61 |
| **2** | 27.891429 | 4.157143 | 109.142857 | 81.241983 | 2423.300000 | 16.787143 | 75.81 |
| **3** | 30.450633 | 4.101266 | 102.708861 | 79.835443 | 2221.227848 | 16.172152 | 77.44 |

In [48]:
```
dataSet_copy.groupby('origin').var()
```

Out[48]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | mc |
|---|---|---|---|---|---|---|---|
| **origin** | | | | | | | |
| **1** | 40.997026 | 2.760332 | 9702.612255 | 1569.532304 | 631695.128385 | 7.568615 | 13 |
| **2** | 45.211230 | 0.250311 | 509.950311 | 410.659789 | 240142.328986 | 9.276209 | 12 |
| **3** | 37.088685 | 0.348588 | 535.465433 | 317.523856 | 102718.485881 | 3.821779 | 13 |

In [49]:
```
dataSet_copy['Clusters'] = cluster.labels_
dataSet_copy.groupby('Clusters').mean()
```

Out[49]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration | mo |
|---|---|---|---|---|---|---|---|
| **Clusters** | | | | | | | |
| **0** | 27.365414 | 4.443609 | 131.934211 | 84.300061 | 2459.511278 | 16.298120 | 76 |
| **1** | 13.889062 | 8.000000 | 358.093750 | 167.046875 | 4398.593750 | 13.025000 | 73 |
| **2** | 17.510294 | 7.014706 | 278.985294 | 124.470588 | 3624.838235 | 15.105882 | 75 |

In [50]:
```
dataSet_copy.groupby('Clusters').var()
```

Out[50]:

| | mpg | cylinders | displacement | horsepower | weight | acceleration |
|---|---|---|---|---|---|---|
| **Clusters** | | | | | | |
| **0** | 41.976309 | 0.851525 | 2828.083391 | 369.143491 | 182632.099872 | 5.718298 |
| **1** | 3.359085 | 0.000000 | 2138.213294 | 756.521577 | 74312.340278 | 3.591429 |
| **2** | 8.829892 | 1.059482 | 2882.492318 | 713.088674 | 37775.809263 | 10.556980 |

Yes, we have a relationship between clusters assignment and class lables. We can see the mean values in both cases are similar. And we noticed that these are 3 classes in both lables as Origin and 3 clusters are created. We have a clear relationship.

# Problem 2

In [98]:
```
from sklearn import preprocessing, datasets
from sklearn.datasets import load_boston
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

In [99]:
```
boston = load_boston()
# scaling the data using preprocessing.scale
scaled_boston_data = pd.DataFrame(data=preprocessing.scale(boston.data), colum
ns= boston.feature_names)
```

In [100]:
```
cluster_range = [2, 3, 4, 5, 6]

for n_clusters in cluster_range:
    km = KMeans(n_clusters = n_clusters, init='k-means++')
    cluster_labels = km.fit_predict(scaled_boston_data)
    silhouette_avg = silhouette_score(scaled_boston_data, cluster_labels)
    print(" For cluster= ", n_clusters, "Avg Silhouette is: ", silhouette_avg)
```

```
 For cluster=  2 Avg Silhouette is:  0.359977342374
 For cluster=  3 Avg Silhouette is:  0.257259122893
 For cluster=  4 Avg Silhouette is:  0.280818056241
 For cluster=  5 Avg Silhouette is:  0.276863962183
 For cluster=  6 Avg Silhouette is:  0.285929370948
```

In [101]:
```
# finding the mean for the optimum cluster = 2, greater
km = KMeans(n_clusters=2, init='k-means++')
cluster_labels = km.fit_predict(scaled_boston_data)
silhouette_avg = silhouette_score(scaled_boston_data, cluster_labels)
print(" For cluster= 2", "Avg Silhouette is: ", silhouette_avg)
scaled_boston_data['CLUSTER'] = cluster_labels
scaled_boston_data.groupby('CLUSTER').mean()
```

```
 For cluster= 2 Avg Silhouette is:  0.359977342374
```

Out[101]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DI |
|---|---|---|---|---|---|---|---|---|
| **CLUSTER** | | | | | | | | |
| **0** | -0.388039 | 0.262392 | -0.620368 | 0.002912 | -0.584675 | 0.243315 | -0.435108 | 0. |
| **1** | 0.721270 | -0.487722 | 1.153113 | -0.005412 | 1.086769 | -0.452263 | 0.808760 | -0 |

In [102]:
```
data_summary = scaled_boston_data.loc[scaled_boston_data['CLUSTER'] == 0]
data_summary.describe()
```

Out[102]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | A |
|---|---|---|---|---|---|---|---|
| **count** | 329.000000 | 329.000000 | 329.000000 | 329.000000 | 329.000000 | 329.000000 | 329.0000 |
| **mean** | -0.388039 | 0.262392 | -0.620368 | 0.002912 | -0.584675 | 0.243315 | -0.43510 |
| **std** | 0.045200 | 1.159853 | 0.622363 | 1.006458 | 0.534114 | 0.944020 | 0.95454 |
| **min** | -0.417713 | -0.487722 | -1.557842 | -0.272599 | -1.465882 | -1.868631 | -2.33543 |
| **25%** | -0.412070 | -0.487722 | -1.045700 | -0.272599 | -1.016689 | -0.445397 | -1.25795 |
| **50%** | -0.405146 | -0.487722 | -0.720322 | -0.272599 | -0.576134 | 0.043261 | -0.42939 |
| **75%** | -0.387200 | 0.585267 | -0.375976 | -0.272599 | -0.144217 | 0.734220 | 0.46318 |
| **max** | -0.111580 | 3.804234 | 2.117615 | 3.668398 | 0.797361 | 3.476688 | 1.11749 |

In [103]:
```
data_summary = scaled_boston_data.loc[scaled_boston_data['CLUSTER'] == 1]
data_summary.describe()
```

Out[103]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | |
|---|---|---|---|---|---|---|---|
| count | 177.000000 | 1.770000e+02 | 177.000000 | 177.000000 | 177.000000 | 177.000000 | 177.0( |
| mean | 0.721270 | -4.877224e-01 | 1.153113 | -0.005412 | 1.086769 | -0.452263 | 0.808 |
| std | 1.437545 | 1.447384e-15 | 0.310655 | 0.993564 | 0.718716 | 0.947526 | 0.406 |
| min | -0.407622 | -4.877224e-01 | 1.015999 | -0.272599 | -0.196047 | -3.880249 | -1.005 |
| 25% | -0.077985 | -4.877224e-01 | 1.015999 | -0.272599 | 0.512296 | -0.935480 | 0.690 |
| 50% | 0.342909 | -4.877224e-01 | 1.015999 | -0.272599 | 1.073787 | -0.290109 | 0.953 |
| 75% | 0.973109 | -4.877224e-01 | 1.231945 | -0.272599 | 1.367490 | 0.135863 | 1.074 |
| max | 9.941735 | -4.877224e-01 | 2.422565 | 3.668398 | 2.732346 | 3.555044 | 1.117 |

In [104]:
```
# scaling the data using preprocessing.normalize
scaled_boston_data_1 = pd.DataFrame(data=preprocessing.normalize(boston.data),
columns= boston.feature_names)
```

In [105]:
```
cluster_range = [2, 3, 4, 5, 6]

for n_clusters in cluster_range:
    km = KMeans(n_clusters = n_clusters, init='k-means++')
    cluster_labels = km.fit_predict(scaled_boston_data_1)
    silhouette_avg = silhouette_score(scaled_boston_data_1, cluster_labels)
    print(" For cluster= ", n_clusters, "Avg Silhouette is: ", silhouette_avg)
    print(" Cluster Centroids= ", km.cluster_centers_)
```

```
 For cluster=  2 Avg Silhouette is:  0.625790153315
 Cluster Centroids=  [[  1.55650380e-02  -2.08166817e-17   2.58355757e-02
8.56414632e-05
    9.51202648e-04   8.46013220e-03   1.27125744e-01   2.92385753e-03
    2.96296276e-02   8.93089272e-01   2.78042327e-02   3.82724656e-01
    2.60228732e-02]
 [  6.48561493e-04   3.23663943e-02   1.58698312e-02   1.47169613e-04
    1.00436783e-03   1.28019832e-02   1.17982345e-01   9.07279657e-03
    8.80792040e-03   6.06836448e-01   3.56496023e-02   7.72519906e-01
    2.02319473e-02]]
 For cluster=  3 Avg Silhouette is:  0.575733193467
 Cluster Centroids=  [[  5.33766803e-04   3.61874259e-02   1.41978422e-02
1.49858483e-04
    9.91875118e-04   1.30845203e-02   1.13438117e-01   9.67327874e-03
    8.89716294e-03   5.88243515e-01   3.62365701e-02   7.89409879e-01
    1.96933254e-02]
 [  9.94657937e-03   1.47186113e-03   2.59539951e-02   1.03376878e-04
    9.67664212e-04   8.89467526e-03   1.30805856e-01   3.41404251e-03
    2.25771863e-02   8.28056708e-01   2.81504699e-02   5.30891636e-01
    2.42049719e-02]
 [  2.21562709e-02  -1.38777878e-17   2.75851760e-02   6.23949590e-05
    1.04851698e-03   9.38784631e-03   1.38957760e-01   3.09816617e-03
    3.35517426e-02   9.80050092e-01   3.06413005e-02   9.05811705e-02
    3.11692578e-02]]
 For cluster=  4 Avg Silhouette is:  0.492676766732
 Cluster Centroids=  [[  1.35513597e-02  -2.08166817e-17   2.52396898e-02
9.27640141e-05
    9.17225925e-04   8.13488591e-03   1.23238155e-01   2.85866780e-03
    2.83372148e-02   8.64014925e-01   2.68483071e-02   4.81089806e-01
    2.42978201e-02]
 [  4.48643313e-04   3.18358518e-02   1.44134947e-02   1.72013654e-04
    1.01896098e-03   1.35363885e-02   1.20366189e-01   9.54134476e-03
    8.83152187e-03   5.55872861e-01   3.70949081e-02   8.14295505e-01
    1.97399010e-02]
 [  2.16604966e-02  -1.38777878e-17   2.76393386e-02   6.40813092e-05
    1.05405111e-03   9.44466151e-03   1.38893581e-01   3.12118859e-03
    3.35417959e-02   9.81098106e-01   3.06978454e-02   8.49706911e-02
    3.12446556e-02]
 [  1.01061804e-03   3.33272194e-02   1.85072911e-02   1.02176466e-04
    9.77939291e-04   1.14719579e-02   1.13665149e-01   8.22424473e-03
    8.76517758e-03   6.99132708e-01   3.30321194e-02   6.96863311e-01
    2.11230548e-02]]
 For cluster=  5 Avg Silhouette is:  0.481830347736
 Cluster Centroids=  [[  5.17499536e-04   1.26579884e-02   1.55039476e-02
1.83002829e-04
    1.05364986e-03   1.34640067e-02   1.37595825e-01   8.60926443e-03
    8.99760599e-03   5.51984876e-01   3.74438281e-02   8.16481356e-01
    2.12790745e-02]
 [  2.16604966e-02  -1.38777878e-17   2.76393386e-02   6.40813092e-05
    1.05405111e-03   9.44466151e-03   1.38893581e-01   3.12118859e-03
    3.35417959e-02   9.81098106e-01   3.06978454e-02   8.49706911e-02
    3.12446556e-02]
 [  1.38289801e-02  -2.08166817e-17   2.50398447e-02   9.53171521e-05
    9.08185679e-04   8.00103924e-03   1.21291414e-01   2.76321427e-03
    2.88441434e-02   8.66149000e-01   2.65703088e-02   4.78314411e-01
    2.38635394e-02]
 [  1.38339654e-03   9.40306870e-03   2.28354236e-02   1.35170950e-04
```

```
    1.04623160e-03   1.11212287e-02   1.37447650e-01   6.02373023e-03
    8.84232300e-03   7.17563118e-01   3.30830741e-02   6.76803054e-01
    2.45032866e-02]
 [  1.86221976e-04   1.00735088e-01   9.21225002e-03   7.75858853e-05
    8.57199069e-04   1.33093828e-02   5.54432266e-02   1.35110102e-02
    8.39463041e-03   6.04876152e-01   3.47523706e-02   7.80471992e-01
    1.37253884e-02]]
 For cluster=  6 Avg Silhouette is:  0.493266259425
 Cluster Centroids=  [[  1.80084755e-04   9.06705111e-02   8.78666946e-03
1.35525272e-19
    8.24908036e-04   1.27117756e-02   4.90516916e-02   1.36392169e-02
    8.33882033e-03   6.44462556e-01   3.34048030e-02   7.51958680e-01
    1.37781803e-02]
 [  2.16604966e-02  -1.38777878e-17   2.76393386e-02   6.40813092e-05
    1.05405111e-03   9.44466151e-03   1.38893581e-01   3.12118859e-03
    3.35417959e-02   9.81098106e-01   3.06978454e-02   8.49706911e-02
    3.12446556e-02]
 [  1.83430627e-04   4.84832093e-02   1.43677352e-02   1.95490333e-04
    1.02272793e-03   1.43545955e-02   1.10161603e-01   1.05306871e-02
    7.96166416e-03   4.89884326e-01   3.92300179e-02   8.56570597e-01
    1.86722971e-02]
 [  1.38289801e-02  -2.08166817e-17   2.50398447e-02   9.53171521e-05
    9.08185679e-04   8.00103924e-03   1.21291414e-01   2.76321427e-03
    2.88441434e-02   8.66149000e-01   2.65703088e-02   4.78314411e-01
    2.38635394e-02]
 [  1.47320787e-03   1.49002359e-03   2.51285445e-02   1.34400862e-04
    1.07852427e-03   1.09716164e-02   1.46093110e-01   5.10650984e-03
    8.67461087e-03   7.23263447e-01   3.30444396e-02   6.70124146e-01
    2.58147366e-02]
 [  7.14125095e-04   1.23567912e-02   1.50149355e-02   1.95591377e-04
    1.04236448e-03   1.30515954e-02   1.41022577e-01   8.31673780e-03
    9.63087845e-03   5.90149609e-01   3.62703464e-02   7.91211546e-01
    2.13795741e-02]]
```

In [106]:

```python
# finding the mean for the optimum cluster = 2, greater
km = KMeans(n_clusters=2, init='k-means++')
cluster_labels = km.fit_predict(scaled_boston_data_1)
silhouette_avg = silhouette_score(scaled_boston_data_1, cluster_labels)
print(" For cluster= 2", "Avg Silhouette is: ", silhouette_avg)
scaled_boston_data_1['CLUSTER'] = cluster_labels
scaled_boston_data_1.groupby('CLUSTER').mean()
```

```
 For cluster= 2 Avg Silhouette is:  0.625790153315
```

Out[106]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS |
|---|---|---|---|---|---|---|---|---|
| **CLUSTER** | | | | | | | | |
| **0** | 0.000649 | 0.032366 | 0.015870 | 0.000147 | 0.001004 | 0.012802 | 0.117982 | 0.00907 |
| **1** | 0.015565 | 0.000000 | 0.025836 | 0.000086 | 0.000951 | 0.008460 | 0.127126 | 0.00292 |

In [107]:
```
data_summary = scaled_boston_data_1.loc[scaled_boston_data_1['CLUSTER'] == 0]
data_summary.describe()
```

Out[107]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | A |
|---|---|---|---|---|---|---|---|
| **count** | 357.000000 | 357.000000 | 357.000000 | 357.000000 | 357.000000 | 357.000000 | 357.000( |
| **mean** | 0.000649 | 0.032366 | 0.015870 | 0.000147 | 0.001004 | 0.012802 | 0.117982 |
| **std** | 0.001011 | 0.052953 | 0.011056 | 0.000527 | 0.000170 | 0.001825 | 0.054527 |
| **min** | 0.000013 | 0.000000 | 0.000963 | 0.000000 | 0.000660 | 0.008357 | 0.00643 |
| **25%** | 0.000119 | 0.000000 | 0.008207 | 0.000000 | 0.000888 | 0.011632 | 0.07095 |
| **50%** | 0.000243 | 0.000000 | 0.012721 | 0.000000 | 0.001003 | 0.012712 | 0.12713 |
| **75%** | 0.000663 | 0.044981 | 0.019558 | 0.000000 | 0.001070 | 0.013769 | 0.16410( |
| **max** | 0.007119 | 0.207644 | 0.061249 | 0.002282 | 0.001659 | 0.018156 | 0.23762( |

In [108]:
```
data_summary = scaled_boston_data_1.loc[scaled_boston_data_1['CLUSTER'] == 1]
data_summary.describe()
```

Out[108]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | |
|---|---|---|---|---|---|---|---|---|
| **count** | 149.000000 | 149.0 | 149.000000 | 149.000000 | 149.000000 | 149.000000 | 149.000000 | 1 |
| **mean** | 0.015565 | 0.0 | 0.025836 | 0.000086 | 0.000951 | 0.008460 | 0.127126 | 0 |
| **std** | 0.017421 | 0.0 | 0.004999 | 0.000349 | 0.000224 | 0.001765 | 0.028194 | 0 |
| **min** | 0.000129 | 0.0 | 0.020582 | 0.000000 | 0.000681 | 0.004682 | 0.051982 | 0 |
| **25%** | 0.006232 | 0.0 | 0.023208 | 0.000000 | 0.000855 | 0.007464 | 0.116167 | 0 |
| **50%** | 0.010680 | 0.0 | 0.023743 | 0.000000 | 0.000897 | 0.008157 | 0.125861 | 0 |
| **75%** | 0.018505 | 0.0 | 0.026696 | 0.000000 | 0.000996 | 0.009070 | 0.131503 | 0 |
| **max** | 0.113081 | 0.0 | 0.046424 | 0.002371 | 0.002065 | 0.015990 | 0.222496 | 0 |

Silhouette score teel us the similarity an object is to its own cluster compared with another cluster. Here k=2, the silhouette score is the highest a thus optimal. The mean values for all features in each cluster for the optimal clustering is the same as that of the centroid co-ordinates.

# Problem 3

In [125]:
```
from sklearn.datasets import load_wine
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import homogeneity_score, completeness_score
```

In [126]:
```
data_wine = load_wine()
# scaling
scaled_wine_data = pd.DataFrame(data = preprocessing.scale(data_wine.data), co
lumns=data_wine.feature_names)
```

In [127]:
```
# for cluster = 3
cluster_model = KMeans(n_clusters=3, init = 'k-means++')
cluster_predict = cluster_model.fit_predict(scaled_wine_data)

target_values = data_wine.target
```

In [128]:
```
#silhouette score
silhouette_avg = silhouette_score(scaled_wine_data, cluster_predict)
print(" For cluster= 3", "Avg Silhouette is: ", silhouette_avg)
```

```
 For cluster= 3 Avg Silhouette is:  0.28485891919
```

In [129]:
```
# cal Homogeneity/Completeness

h_score = homogeneity_score(target_values, cluster_predict)
print(h_score)
c_score = completeness_score(target_values, cluster_predict)
print(c_score)
```

```
0.878843200366
0.872963601608
```

In [130]:
```
# normalize
norm_wine_data = pd.DataFrame(data = preprocessing.normalize(data_wine.data),
columns=data_wine.feature_names)
```

In [131]:
```
# for cluster = 3
cluster_model = KMeans(n_clusters=3, init = 'k-means++')
cluster_predict = cluster_model.fit_predict(norm_wine_data)

target_values = data_wine.target
```

In [132]:
```
#silhouette score
silhouette_avg = silhouette_score(norm_wine_data, cluster_predict)
print(" For cluster= 3", "Avg Silhouette is: ", silhouette_avg)
```

```
 For cluster= 3 Avg Silhouette is:  0.523346128229
```

In [133]:
```
# cal Homogeneity/Completeness

h_score = homogeneity_score(target_values, cluster_predict)
print(h_score)
c_score = completeness_score(target_values, cluster_predict)
print(c_score)
```

```
0.376177457958
0.38858465671
```

if (Homogeneity == 1), then each cluster contains only members of a single class.

if (Completeness == 1), then all the members of a given class are assigned to the same cluster.

Homogeneity and completeness both score are used to measure the quality of the cluster.