Intelligent Data Analysis
Homework #3
Due Date: Nov 30th, 2017
Name: Anshul Gautam

Consider the data file attached with this homework. It contains scores for some students in four different subjects (Physics, Maths, English, and Music). Perform the following tasks with this data set.

1. Perform k-means clustering with this dataset for values of k to be 3, 4, 5, 6, 7, and 8. For each case of k run the clustering algorithm with three different initial cluster centers and select the one with the lowest total SSE value of all clusters in the clustering. Report the following in the submitted work: (Use Matlab kmeans function or any other similar toolbox)

Python code:
Kmean(n_clusters=3, n_init=3) runs the kmeans algorithm 3 times before selecting the minimum total SSE. Also, since python does not provide SSE of individual clusters, KMeans was run on each cluster with a value 1 to find the centroid of individual clusters. Kmean(n_clusters=1) as seen in the code below.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm


import sklearn
from sklearn.cluster import KMeans
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import scale
import sklearn.metrics as sm
from sklearn import datasets
from sklearn.metrics import confusion_matrix, classification_report
import itertools
from sklearn.metrics import silhouette_samples, silhouette_score
from scipy.spatial.distance import cdist, pdist


```

```
address = 'D:/Meng Classes/Intelligent data analysis/Homework 3/HW3-StudentData3.xlsx'
studentsData = pd.read_excel(address)
studentsDf = pd.DataFrame(data=studentsData)
studentNormData = scale(studentsDf)
studentDfNorm = pd.DataFrame(data=studentNormData)
```

```
estimators = [('studentData_3_clusters', KMeans(n_clusters=3, n_init = 3)), ('studentData_4_clusters', KMeans(n_clusters=4, n
```

```
fignum = 1
```

```
titles = ['3 clusters', '4 clusters', '5 clusters', '6 clusters', '7 clusters', '8 clusters']
#%matplotlib inline
sse = []
centroids2 = []
D_k = []
```

```
fignum = 1
range_n_clusters = [3, 4, 5, 6, 7, 8]
for name, est in estimators:
    n_clusters = range_n_clusters[fignum - 1]
    #fig = plt.figure(fignum, figsize=(4, 3))

    est.fit(studentNormData)
    labels = est.labels_





    # Create a subplot with 1 row and 1 columns
    sfig, ax1 = plt.subplots(1, 1)
    sfig.set_size_inches(18, 5)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -0.4, 1
    ax1.set_xlim([-0.4, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(studentNormData) + (n_clusters + 1) * 10])

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(studentNormData, est.labels_)
    print("For n_clusters =", titles[fignum - 1], ", the average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(studentNormData, est.labels_)

    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[est.labels_ == i]

        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
```

```
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1])

    #ax2 = Axes3D(sfig, rect=[1, 0, 0.95, 1], elev=48, azim=130)

   # ax2.scatter(studentNormData[:, 0], studentNormData[:, 1], studentNormData[:, 2], c=labels.astype(np.float), edgecolor='

    centroids = est.cluster_centers_
    #centroids2.append = est.cluster_centers_
    #D_k.append = cdist(studentNormData, centroids, 'euclidean')
    #ax2.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], s=169, marker='x', linewidths=7, color='b')

    #ax2.w_xaxis.set_ticklabels([])
    #ax2.w_yaxis.set_ticklabels([])
    #ax2.w_zaxis.set_ticklabels([])
    #ax2.set_xlabel('Physics')
    #ax2.set_ylabel('Maths')
    #ax2.set_zlabel('English')
    print("Sum of square errors = ", est.inertia_)
    print("Cluster Centers ", est.cluster_centers_)
    #print("Centroids ", centroids)
    print(" \n")
    sse.append(est.inertia_)
    #ax2.set_title(titles[fignum - 1])
    #ax2.dist = 12

    ClusterAssignments = pd.DataFrame(data=labels)
    StudentClustNorm = pd.concat([studentDfNorm, ClusterAssignments], axis=1)
    studentId = pd.DataFrame(np.arange(1,70).reshape(69,1))
    StudentClustNorm = pd.concat([studentId, StudentClustNorm], axis=1)
    StudentClustNorm.columns = ['studentId', 'physics', 'maths', 'english', 'music', 'clusters']
```

```
    StudentClustNorm.columns = ['studentId', 'physics', 'maths', 'english', 'music', 'clusters']
    StudentGroup = StudentClustNorm.groupby(StudentClustNorm['clusters'])
    StudentGroup.studentId.groups
    StudentGroup.count()

    for numberClusters in range(n_clusters):
        ClusterIndex = StudentClustNorm[['physics', 'maths', 'english', 'music', 'clusters']].groupby(['clusters']).get_grou
        del ClusterIndex['clusters']
        KM = KMeans(n_clusters=1).fit(ClusterIndex)
        print("Cluster SSE", KM.inertia_)

    fignum = fignum + 1
```

a. Show the cluster centers, SSE values of the clusters, and the total SSE value for the clustering for each value of k.

Answer:

```
For n_clusters = 3 clusters , the average silhouette_score is : 0.36743313
7074
Sum of square errors =  129.190038934
Cluster Centers  [[-0.53877602 -0.05898597  0.43749553  0.81043823]
 [ 1.05384946  0.58880541  0.11110165 -0.54263997]
 [-0.92356848 -1.22012634 -1.53531816 -1.07503492]]
Cluster SSE 49.7266468002
```

```
Cluster SSE 49.726360182
Cluster SSE 29.7370319522

For n_clusters = 4 clusters , the average silhouette_score is : 0.30787614
0042
Sum of square errors =  109.11871002
Cluster Centers  [[-0.92356848 -1.22012634 -1.53531816 -1.07503492]
 [ 0.95419872  0.43426569 -0.02131525 -0.75088282]
 [ 0.13317498  0.53520999  0.29108882  0.8896329 ]
 [-1.13208976 -0.64164162  0.84421511  0.73259731]]
Cluster SSE 29.7370319522
Cluster SSE 39.3680683271
Cluster SSE 22.4077171976
Cluster SSE 17.6058925429


For n_clusters = 5 clusters , the average silhouette_score is : 0.34111176
5111
Sum of square errors =  91.0583496517
Cluster Centers  [[-1.1954119  -0.57507957  0.97922263  0.71951101]
 [-0.13739219  0.39736028  0.17521136  0.93265223]
 [ 0.23341942 -0.56384182  0.20284368 -1.15822932]
 [-0.81789413 -1.2228234  -1.78775274 -0.78577438]
 [ 1.24607389  1.00994244  0.04120949 -0.26797811]]
Cluster SSE 13.4190893566
Cluster SSE 10.3669588712
Cluster SSE 19.7629698552
Cluster SSE 26.1008508486
Cluster SSE 21.4084807199

For n_clusters = 6 clusters , the average silhouette_score is : 0.34527167
8932
Sum of square errors =  80.7338545059
Cluster Centers  [[ 0.14695859 -0.71847324  0.35954565 -0.98288112]
 [-0.13438922  0.34839895  0.25884238  0.93104188]
 [ 1.23828198  1.00849665 -0.03275818 -0.40712196]
 [-0.71353251 -1.74412651 -1.86002971 -0.97081336]
 [-1.13208976 -0.64164162  0.84421511  0.73259731]
 [-1.00014409  0.8835802  -1.77598672 -0.33055167]]
Cluster SSE 10.2271960849
Cluster SSE 7.48965027355
Cluster SSE 29.9225814449
Cluster SSE 9.96894916171
Cluster SSE 17.6058925429
Cluster SSE 5.51958499794


For n_clusters = 7 clusters , the average silhouette_score is : 0.36413119
0935
Sum of square errors =  65.8613044569
Cluster Centers  [[-1.03383736 -1.44038622 -0.78764116 -1.63051755]
 [-0.14975176  0.30063678  0.12515418  0.93038805]
 [-1.1954119  -0.57507957  0.97922263  0.71951101]
 [ 1.31243942  0.96186985  0.51838691  0.07422109]
```

```
 [ 0.47776524 -0.56889881  0.422788   -0.84385047]
 [ 0.55178833  1.00444843 -1.2138325  -1.17529484]
 [-0.54451919 -1.91012411 -2.21721242 -0.4302783 ]]
Cluster SSE 4.51273364621
Cluster SSE 14.4774714395
Cluster SSE 13.4190893566
Cluster SSE 7.47251610771
Cluster SSE 3.15071520263
Cluster SSE 18.1311108658
Cluster SSE 4.69766783845

For n_clusters = 8 clusters , the average silhouette_score is : 0.38466144
4782
Sum of square errors =  55.4057707814
Cluster Centers  [[ 0.14695859 -0.71847324  0.35954565 -0.98288112]
 [-0.14249723  0.97423137 -0.41308957  0.74884237]
 [-1.40599484  0.71051887 -2.04072214 -0.97584159]
 [ 1.11115495  1.0884283  -0.97757831 -1.23228149]
 [-0.71353251 -1.74412651 -1.86002971 -0.97081336]
 [-0.21006394 -0.23913699  0.58067759  0.93932368]
 [-1.59551948 -0.37111443  0.84195242  0.73124356]
 [ 1.31243942  0.96186985  0.51838691  0.07422109]]
Cluster SSE 10.2271960849
Cluster SSE 2.34096382119
Cluster SSE 1.43269114202
Cluster SSE 4.75969448429
Cluster SSE 9.96894916171
Cluster SSE 12.4174212412
Cluster SSE 6.78633873837
Cluster SSE 7.47251610771
```

Cluster 4 cluster labels:

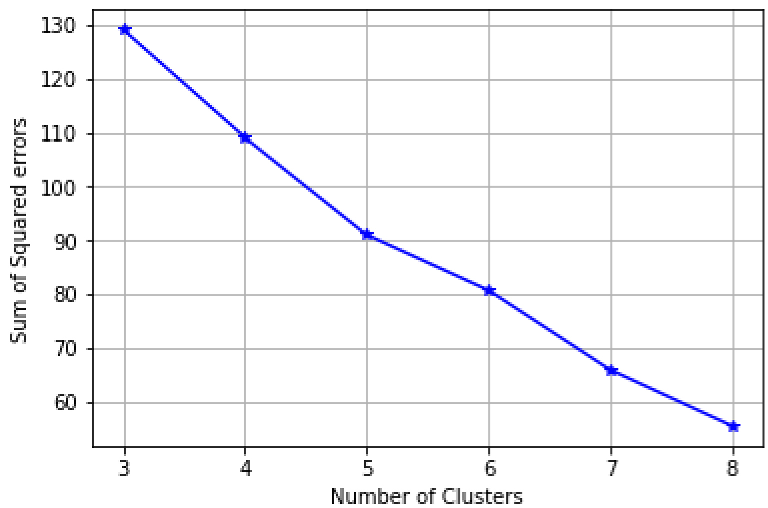
```
In [141]: name, est = estimators[1]

In [144]: est.labels_

Out[144]: array([1, 1, 3, 0, 2, 2, 1, 3, 3, 0, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1,
       3, 1, 1, 0, 3, 2, 0, 2, 2, 0, 0, 2, 0, 1, 0, 1, 0, 1, 1, 0, 0, 3, 2,
       2, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

b. Plot the total SSE value against the values of k.

Answer:

```
]:  plt.plot(range(3,9), sse, 'b*-')
    plt.grid(True)
    plt.xlabel('Number of Clusters')
    plt.ylabel('Sum of Squared errors')
```

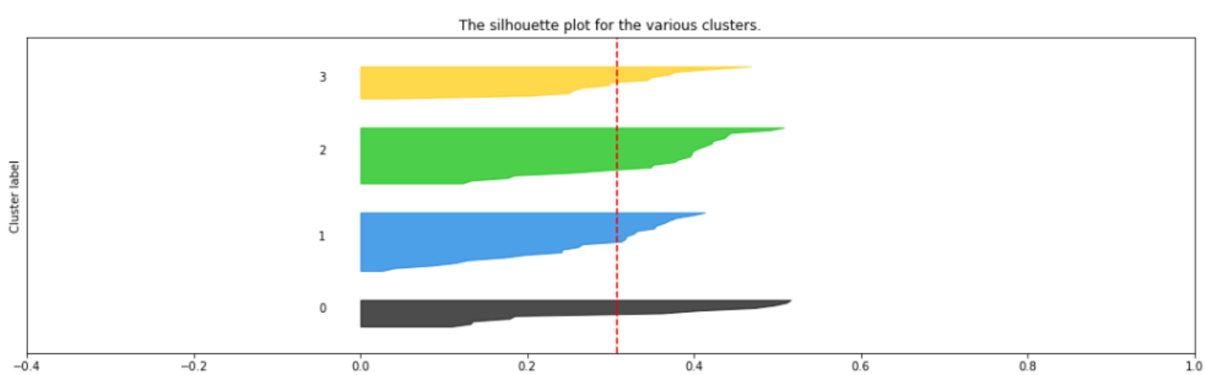
```
]: <matplotlib.text.Text at 0x23276ed90b8>
```



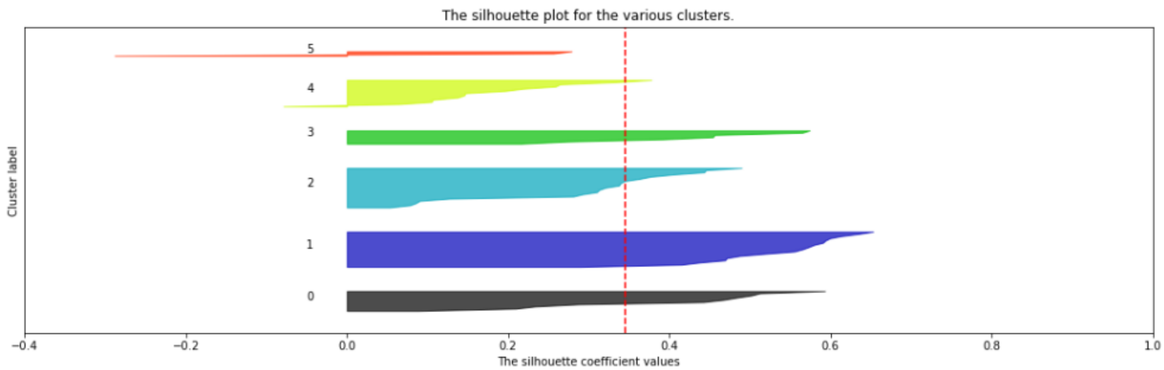
c. Show a plot of the silhouette coefficients for the data points in any two of the clusterings. (Each value of k results in one clustering)

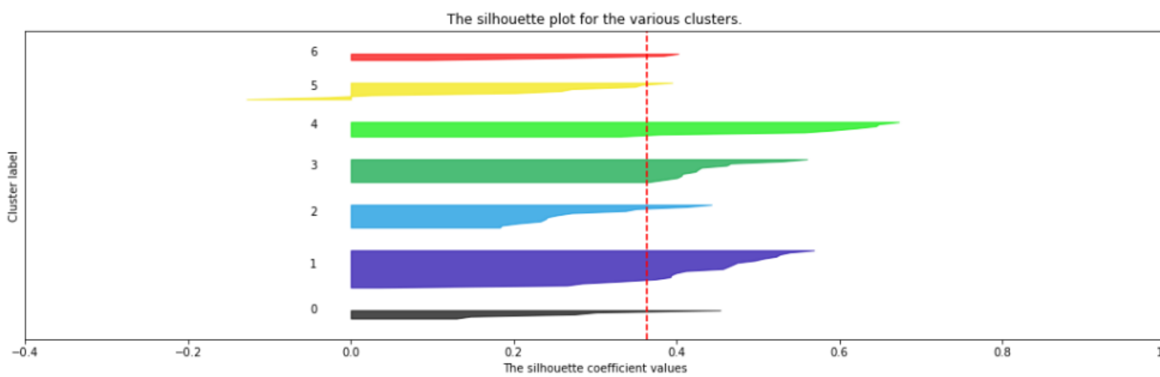
Answer:

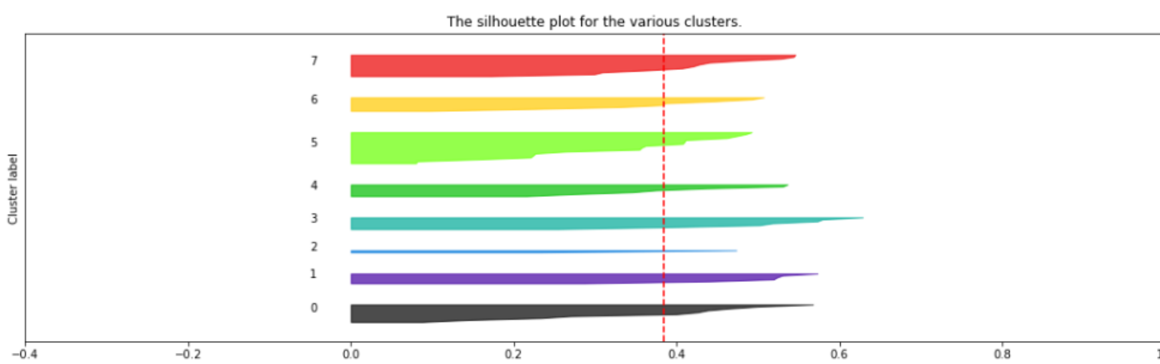
For silhouette code, refer to screenshots above.




The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4


The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5


The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6


The silhouette plot for the various clusters.
Cluster label
The silhouette coefficient values
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4
5
6
7

d. How many clusters would you form in this dataset? Justify your answer. For your choice of the best number of clusters, report the centroids of all the clusters and their SSE values (Call this as Clustering-1).

Answer:

Based on the silhouette coefficients plot for different values of k, the plots with value of 6, 7 and 8 are a bad pick for the given data as the silhouette scores are below average in most cases and even the thickness of silhouette plot has too many variations.

The silhouette plot with 3 clusters has high variation in thickness of the size of clusters.

For k=5, the silhouette plot has high variation in thickness.

When number of clusters is equal to 4, all plots are mostly of similar thickness. Hence k = 4 is a good value for clustering.

Clustering 1

```
For n_clusters = 4 clusters , the average silhouette_score is : 0.30787614
0042
Sum of square errors =  109.11871002
Cluster Centers  [[-0.92356848 -1.22012634 -1.53531816 -1.07503492]
 [ 0.95419872  0.43426569 -0.02131525 -0.75088282]
 [ 0.13317498  0.53520999  0.29108882  0.8896329 ]
 [-1.13208976 -0.64164162  0.84421511  0.73259731]]
Cluster SSE 29.7370319522
Cluster SSE 39.3680683271
Cluster SSE 22.4077171976
Cluster SSE 17.6058925429
```

e. Generate 100 random 4-dimensional random data points such that each attribute can take values between 0 and 100. With this dataset form the same number of clusters as selected by you in (d) above. Report the centroids and populations of the clusters. Compare the total SSE for this random dataset with the SSE for the clustering of the provided dataset. Compare and comment on the differences between the two total SSE values.

Answer:

Code for random points

In [20]:
```python
studentNormData = np.random.uniform(-1,1,size=(100,4))
#studentsDf = pd.DataFrame(data=studentsData)
#studentNormData = scale(studentsDf)
```

In [21]:
```python
)), ('studentData_7_clusters', KMeans(n_clusters=7, n_init = 3)), ('studentData_8_clusters', KMeans(n_clusters=8, n_init = 3))
```

In [22]:
```python
fignum = 1
```

In [23]:
```python
titles = ['3 clusters', '4 clusters', '5 clusters', '6 clusters', '7 clusters', '8 clusters']
%matplotlib inline
sse = []
centroids2 = []
D_k = []
```

In [24]:
```python
fignum = 1
range_n_clusters = [3, 4, 5, 6, 7, 8]
for name, est in estimators:
    n_clusters = range_n_clusters[fignum - 1]
    #fig = plt.figure(fignum, figsize=(4, 3))

    est.fit(studentNormData)
    labels = est.labels_


    # Create a subplot with 1 row and 1 columns
    sfig, ax1 = plt.subplots(1, 1)
    sfig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -0.4, 1
    ax1.set_xlim([-0.4, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(studentNormData) + (n_clusters + 1) * 10])

    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = silhouette_score(studentNormData, est.labels_)
    print("For n_clusters =", titles[fignum - 1], "The average silhouette_score is :", silhouette_avg)

    # Compute the silhouette scores for each sample
    sample_silhouette_values = silhouette_samples(studentNormData, est.labels_)
```

```python
    y_lower = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
        ith_cluster_silhouette_values = \
            sample_silhouette_values[est.labels_ == i]
        print("Population ", len(ith_cluster_silhouette_values))
        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7)

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

        # Compute the new y_lower for next plot
        y_lower = y_upper + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the various clusters.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1])
```

```python

    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1])

    #ax2 = Axes3D(sfig, rect=[1, 0, 0.95, 1], elev=48, azim=130)

   # ax2.scatter(studentNormData[:, 0], studentNormData[:, 1], studentNormData[:, 2], c=labels.astype(np.float), edgecolor='k

    centroids = est.cluster_centers_
    #centroids2.append = est.cluster_centers_
    #D_k.append = cdist(studentNormData, centroids, 'euclidean')
    #ax2.scatter(centroids[:, 0], centroids[:, 1], centroids[:, 2], s=169, marker='x', linewidths=7, color='b')

    #ax2.w_xaxis.set_ticklabels([])
    #ax2.w_yaxis.set_ticklabels([])
    #ax2.w_zaxis.set_ticklabels([])
    #ax2.set_xlabel('Physics')
    #ax2.set_ylabel('Maths')
    #ax2.set_zlabel('English')
    print("Sum of square errors = ", est.inertia_)
    print("Cluster Centers ", est.cluster_centers_)
    #print("Centroids ", centroids)
    print(" \n")
    sse.append(est.inertia_)
    #ax2.set_title(titles[fignum - 1])
    #ax2.dist = 12

    fignum = fignum + 1
```

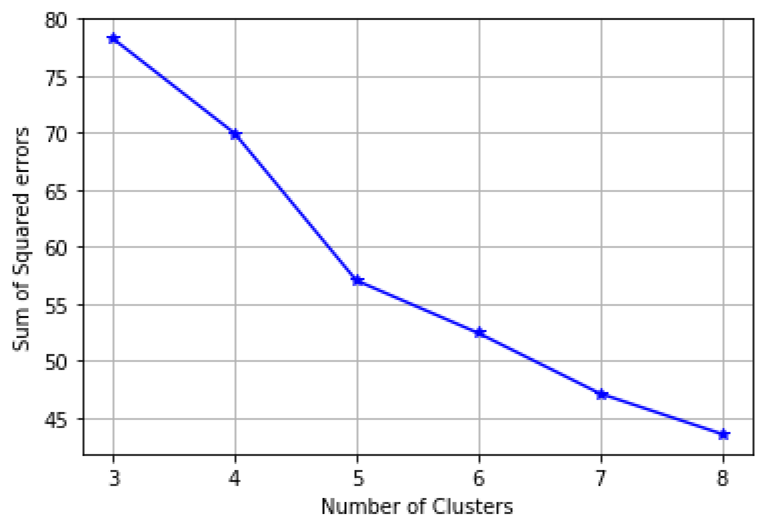
The population and cluster center for 100 random points with different clusters is present below. The silhouette plot has been added to provide more information. As seen from the silhouette plots, the random data almost has similar distribution for all values of k, and the thickness of each plot element is the same which indicates almost equal categorization, and no extreme groups with unique properties is detected.

The SSE value distribution of random points: As seen from the plot for different k values, the SSE values are between 80 and 45 for the random data, whereas the SSE values for students dataset range between 60 to 130. The higher SSE in case of students for different cluster sizes indicates that the inter cluster distance is higher, and therefore the clusters obtained using students dataset have unique properties as compared to the random dataset where inter cluster distance is very low. The silhouette plot confirms this observation.

5]:

```
plt.plot(range(3,9), sse, 'b*-')
plt.grid(True)
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared errors')
```

5]: <matplotlib.text.Text at 0x26d2e0da7f0>

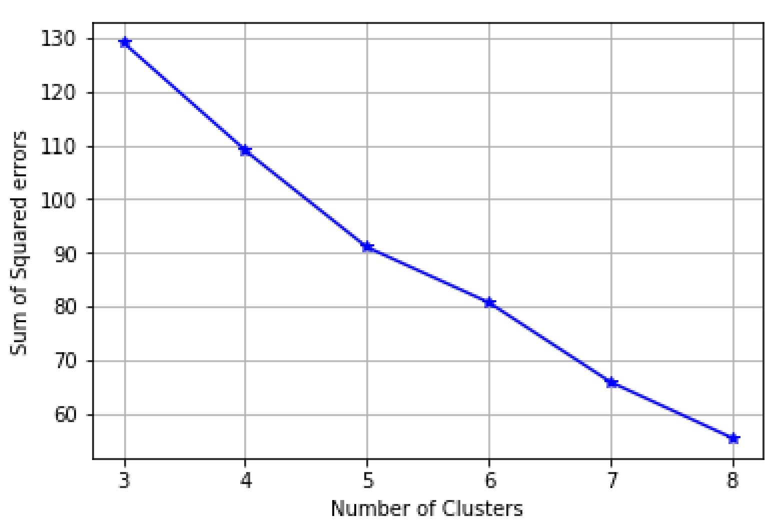


The SSE value distribution of the students dataset.

]:

```
plt.plot(range(3,9), sse, 'b*-')
plt.grid(True)
plt.xlabel('Number of Clusters')
plt.ylabel('Sum of Squared errors')
```

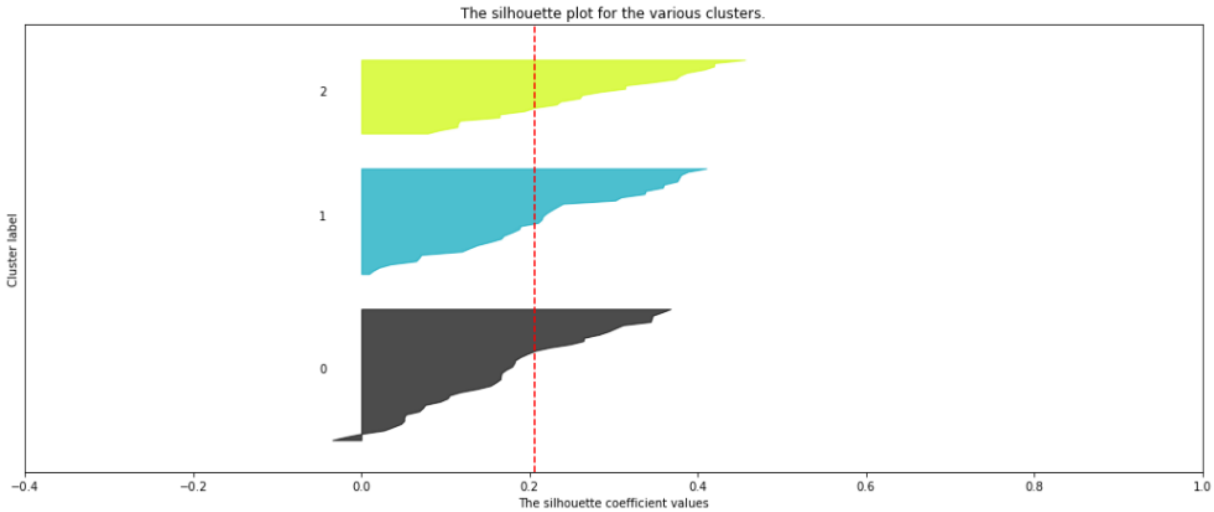
]: <matplotlib.text.Text at 0x23276ed90b8>

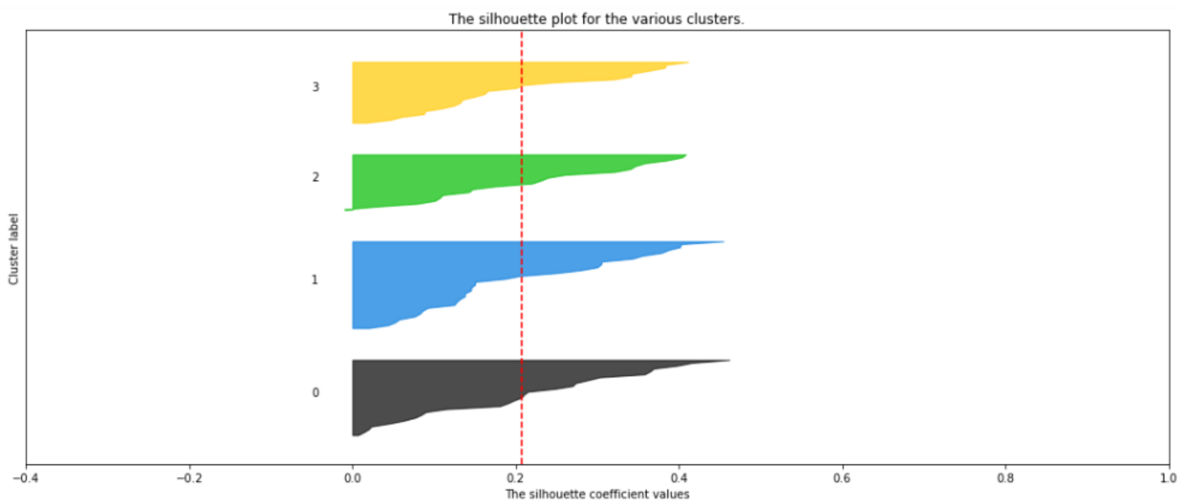


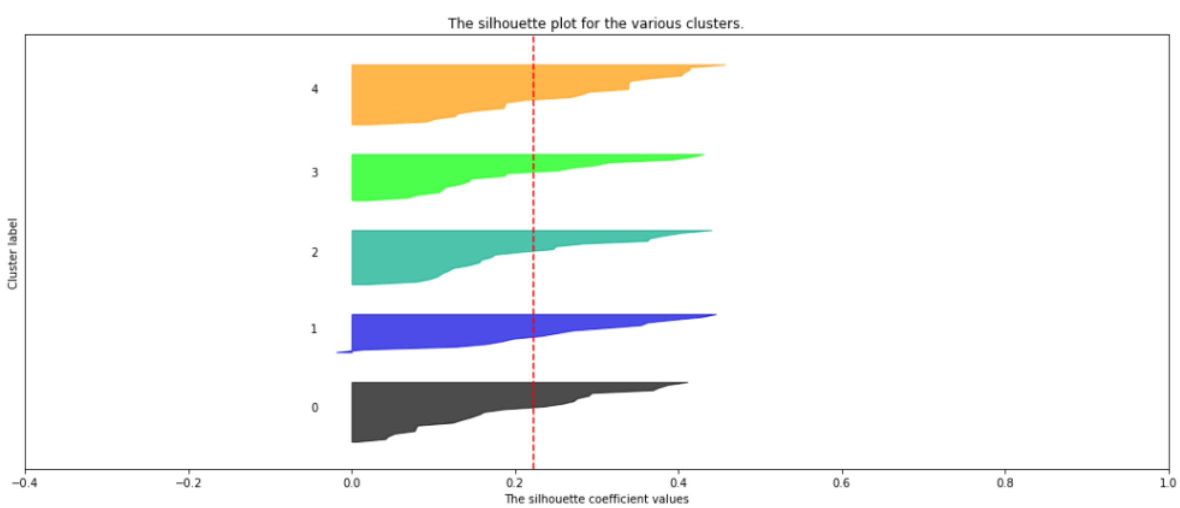

```
For n_clusters = 3 clusters The average silhouette_score is : 0.2061050436
62
```

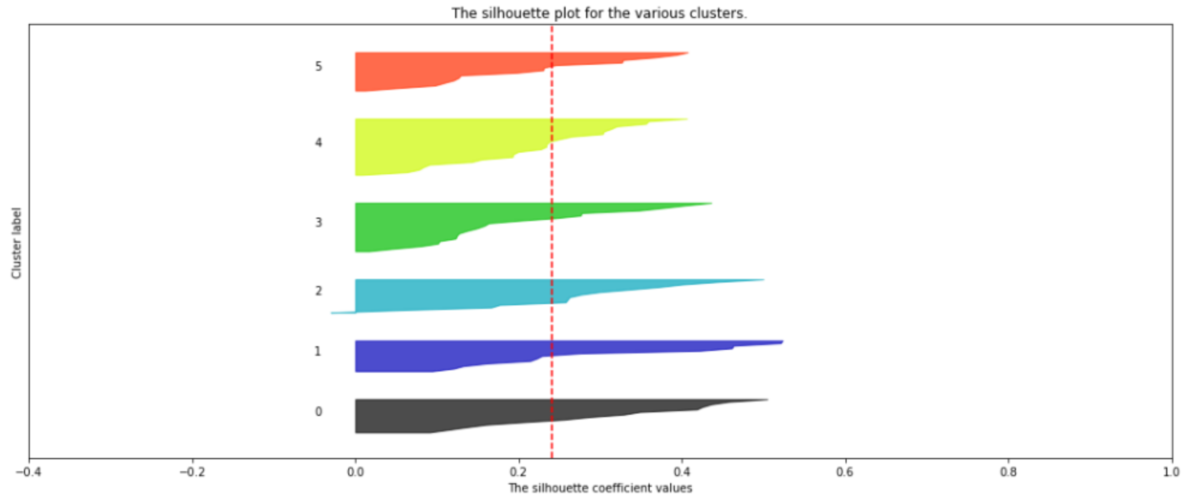
```
Population  42
Population  34
Population  24
Sum of square errors =  80.2481958313
Cluster Centers  [[-0.07150196 -0.57254668  0.30254164  0.1027617 ]
 [ 0.5167953   0.2702406  -0.1278396   0.30442136]
 [-0.62641003  0.43031071 -0.14526429 -0.10049306]]


For n_clusters = 4 clusters The average silhouette_score is : 0.2068601679
43
Population  27
Population  31
Population  20
Population  22
Sum of square errors =  70.3977196748
Cluster Centers  [[-0.38309042 -0.46511708  0.54873936  0.13117643]
 [ 0.47073048 -0.41325626 -0.26237604  0.17491911]
 [-0.13753185  0.46386752 -0.05170031 -0.5071111 ]
 [-0.08929315  0.52547418 -0.03520181  0.61056592]]


For n_clusters = 5 clusters The average silhouette_score is : 0.2229354870
14
Population  23
Population  15
Population  21
Population  18
Population  23
Sum of square errors =  59.5854574602
Cluster Centers  [[ 0.50570352  0.35523295 -0.21692529  0.53220388]
 [-0.76840974  0.09170349 -0.13162794  0.41651265]
 [-0.24326558 -0.43348712  0.76602996  0.16813782]
 [-0.21609795  0.506934   -0.12195346 -0.51777949]
 [ 0.36640968 -0.61299197 -0.08930031 -0.01933731]]


For n_clusters = 6 clusters The average silhouette_score is : 0.2407466862
45
Population  14
Population  13
Population  14
Population  20
Population  23
Population  16
Sum of square errors =  50.7528531476
Cluster Centers  [[-0.51600276  0.37704934 -0.27691193 -0.589733  ]
 [ 0.48786026  0.17467248 -0.67570637  0.56277321]
 [-0.77507356  0.1504569  -0.05202516  0.55803044]
 [-0.22489554 -0.44238434  0.77116274  0.12927091]
 [ 0.32631582 -0.60959467 -0.07457764 -0.02897508]
 [ 0.5162281   0.54257572  0.28469952  0.21645481]]
```

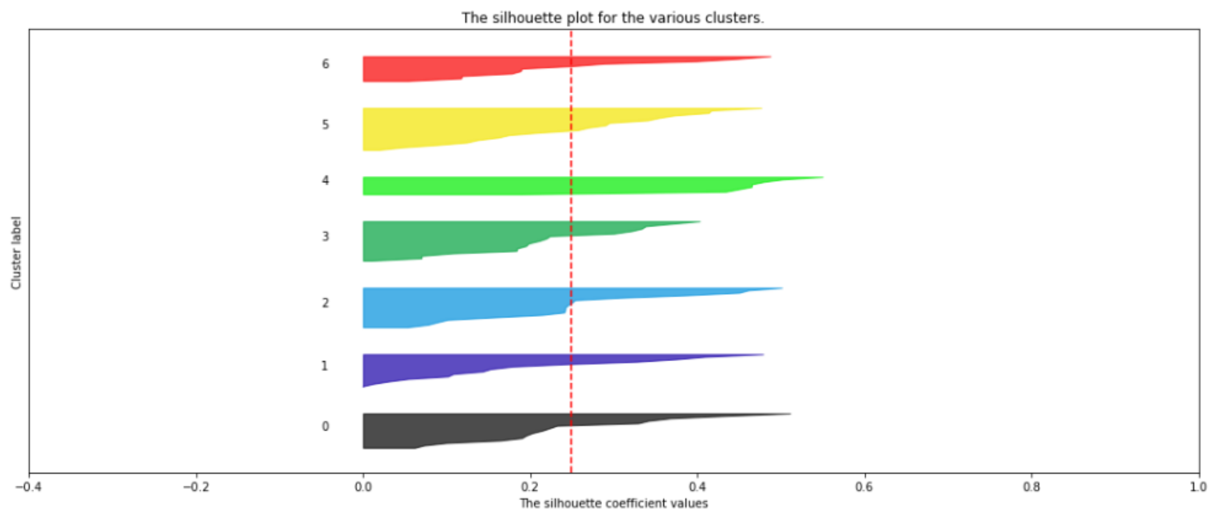
```
For n_clusters = 7 clusters The average silhouette_score is : 0.2497100499
02
Population  15
Population  14
Population  17
Population  17
Population  8
Population  18
Population  11
Sum of square errors =  45.8572222599
Cluster Centers  [[-0.76928396  0.14275456  0.11780858  0.6145646 ]
 [ 0.31421125  0.45392195  0.27164946 -0.31322465]
 [ 0.35835935 -0.46330955  0.60390074  0.31650632]
 [ 0.49381578  0.35210232 -0.43615769  0.63264304]
 [-0.62707611  0.5676274  -0.52349512 -0.60245309]
 [ 0.16993558 -0.6244952  -0.33431291  0.00767608]
 [-0.53225359 -0.40337556  0.60523633 -0.3666294 ]]


For n_clusters = 8 clusters The average silhouette_score is : 0.2321309628
01
Population  11
Population  18
Population  13
Population  12
Population  10
Population  13
Population  12
Population  11
Sum of square errors =  42.7281925651
Cluster Centers  [[-0.46121027  0.66468176 -0.04287307  0.63622002]
 [ 0.36605821 -0.59780073 -0.09838415 -0.16266541]
 [ 0.54188536 -0.01084256  0.58631074  0.48965624]
 [-0.35546893 -0.49220016  0.75750471 -0.19055648]
 [-0.66661388  0.40074561 -0.41411774 -0.60073994]
 [-0.60842197 -0.45178694  0.0523983   0.60153486]
 [ 0.58560505  0.13063704 -0.66244488  0.52018084]
 [ 0.25344216  0.47846145  0.16487451 -0.35904873]]
```

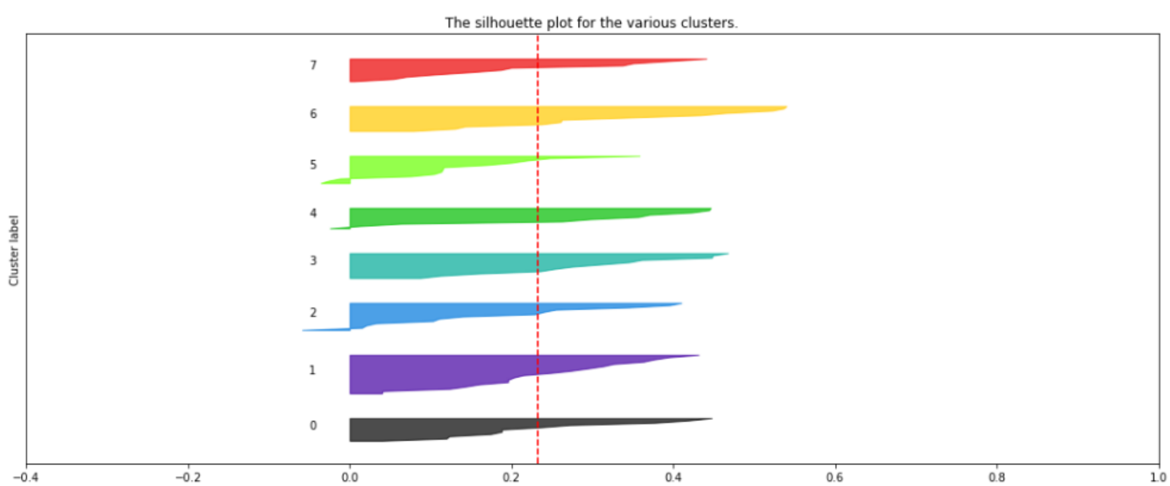

The silhouette plot for the various clusters.
Cluster label
2
1
0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values


The silhouette plot for the various clusters.
Cluster label
3
2
1
0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values


The silhouette plot for the various clusters.
Cluster label
4
3
2
1
0
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
The silhouette coefficient values







2. Perform hierarchical clustering for the students' scores dataset. Generate and show dendrograms for the cases (i) Single-Linkage clustering (Clustering-2), and (ii) Complete-Linkage

clustering (Clustering-3). Use Euclidean distance for computing distance between data points. Report the following in the submitted work: (Use Matlab functions pdist and linkage, or any other similar toolbox.) Make sure the dendrogram shows all points at its lowest level.

Code for hierarchical clustering

In [103]:
```python

# needed imports
from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram, linkage
import numpy as np
import pandas as pd
import sklearn
from sklearn.preprocessing import scale
from scipy.spatial.distance import cdist, pdist
from scipy.cluster.hierarchy import fcluster
from sklearn.metrics import confusion_matrix
import seaborn as sns; sns.set()  # for plot styling


from sklearn.cluster import KMeans
from sklearn.metrics.cluster import adjusted_rand_score

```

In [104]:
```python
%matplotlib inline
np.set_printoptions(precision=5, suppress=True)  # suppress scientific float notation
```

In [105]:
```python
address = 'D:/Meng Classes/Intelligent data analysis/Homework 3/HW3-StudentData3.xlsx'
studentsData = pd.read_excel(address)
studentsDf = pd.DataFrame(data=studentsData)
studentNormData = scale(studentsDf)
studentPDDist = pdist(studentNormData, 'euclidean')
```

In [106]:
```python
print(studentNormData.shape)
```

```
(69, 4)
```

In [107]:
```python
Z = linkage(studentNormData, 'single', 'euclidean')
```

In [108]:
```python
# calculate full dendrogram
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    Z,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8.,  # font size for the x axis labels
)
plt.show()
```

In [109]:
```python
A = linkage(studentNormData, 'complete', 'euclidean')
# calculate full dendrogram
plt.figure(figsize=(25, 10))
plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('sample index')
plt.ylabel('distance')
dendrogram(
    A,
    leaf_rotation=90.,  # rotates the x axis labels
    leaf_font_size=8.,  # font size for the x axis labels
)
plt.show()
```

Hierarchical Clustering Dendrogram

In [69]:
```python
completeLink = fcluster(linkage(studentNormData, 'complete', 'euclidean'), 4, 'maxclust')
completeLink
```

Out[69]:
```
array([4, 3, 4, 4, 3, 3, 3, 4, 4, 4, 2, 3, 3, 4, 2, 3, 4, 3, 3, 2, 2, 4, 4,
       4, 3, 4, 4, 1, 3, 1, 3, 3, 4, 4, 3, 1, 2, 1, 2, 2, 2, 4, 2, 1, 4, 3,
       3, 4, 3, 3, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3], dtype=int32)
```

In [70]:
```python
singleLink = fcluster(linkage(studentNormData, 'single', 'euclidean'), 4, 'maxclust')
singleLink
```

Out[70]:
```
array([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, 4, 2, 1, 2, 2, 1, 2, 3, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], dtype=int32)
```

In [75]:
```
singleLinkClusterAssignments = pd.DataFrame(data=singleLink)
completeLinkClusterAssignments = pd.DataFrame(data=completeLink)
```

In [125]:
```
singleLinkStudentClustNorm = pd.concat([studentDfNorm, singleLinkClusterAssignments], axis=1)
completeLinkStudentClustNorm = pd.concat([studentDfNorm, completeLinkClusterAssignments], axis=1)
singleLinkStudentClustNorm
```

Out[125]:

	0	1	2	3	0
0	0.546683	-1.390940	0.694877	-0.729458	2
1	1.098028	1.031918	-0.011084	-0.518272	2
2	-0.372224	-0.797587	1.283178	0.608052	2
3	-1.337077	-1.885401	0.341897	-1.714992	2
4	-0.372224	1.180257	-0.717045	1.241609	2
5	-0.188443	1.229703	-1.246516	0.960028	2
6	1.006137	0.784688	0.224236	-0.307087	2

In [78]:
```
studentId= range(1,70)
```

In [79]:
```
singleLinkStudentClustNorm.columns = ['physics', 'maths', 'english', 'music', 'clusters']
completeLinkStudentClustNorm.columns = ['physics', 'maths', 'english', 'music', 'clusters']
```

In [80]:
```
studentClustNorm.head()
```

In [82]:
```
studentId = pd.DataFrame(np.arange(1,70).reshape(69,1))

```

In [83]:
```
studentId.head()
```

Out[83]:

	0
0	1
1	2
2	3
3	4
4	5

In [84]:
```
singleLinkStudentClustNorm = pd.concat([studentId, singleLinkStudentClustNorm], axis=1)
completeLinkStudentClustNorm = pd.concat([studentId, completeLinkStudentClustNorm], axis=1)
```

In [85]:
```
singleLinkStudentClustNorm.head()
```

Out[85]:

	0	physics	maths	english	music	clusters
0	1	0.546683	-1.390940	0.694877	-0.729458	2
1	2	1.098028	1.031918	-0.011084	-0.518272	2
2	3	-0.372224	-0.797587	1.283178	0.608052	2
3	4	-1.337077	-1.885401	0.341897	-1.714992	2
4	5	-0.372224	1.180257	-0.717045	1.241609	2

In [86]:
```
singleLinkStudentClustNorm.columns = ['studentId', 'physics', 'maths', 'english', 'music', 'clusters']
completeLinkStudentClustNorm.columns = ['studentId', 'physics', 'maths', 'english', 'music', 'clusters']
```

In [87]:
```
completeLinkStudentClustNorm.head()
```

Out[87]:

	studentId	physics	maths	english	music	clusters
0	1	0.546683	-1.390940	0.694877	-0.729458	4
1	2	1.098028	1.031918	-0.011084	-0.518272	3

In [86]:
```
singleLinkStudentClustNorm.columns = ['studentId', 'physics', 'maths', 'english', 'music', 'clusters']
completeLinkStudentClustNorm.columns = ['studentId', 'physics', 'maths', 'english', 'music', 'clusters']
```

In [87]:
```
completeLinkStudentClustNorm.head()
```

Out[87]:

	studentId	physics	maths	english	music	clusters
0	1	0.546683	-1.390940	0.694877	-0.729458	4
1	2	1.098028	1.031918	-0.011084	-0.518272	3
2	3	-0.372224	-0.797587	1.283178	0.608052	4
3	4	-1.337077	-1.885401	0.341897	-1.714992	4
4	5	-0.372224	1.180257	-0.717045	1.241609	3

In [88]:
```
singleLinkStudentGroup = singleLinkStudentClustNorm.groupby(singleLinkStudentClustNorm['clusters'])
completeLinkStudentGroup = completeLinkStudentClustNorm.groupby(completeLinkStudentClustNorm['clusters'])
```

In [89]:
```
singleLinkStudentGroup.studentId.groups

```

Out[89]:
```
{1: Int64Index([39, 42], dtype='int64'),
 2: Int64Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
            17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
            34, 35, 36, 38, 40, 41, 43, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54,
            55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68],
           dtype='int64'),
 3: Int64Index([44], dtype='int64'),
 4: Int64Index([37], dtype='int64')}
```

In [90]:
```
completeLinkStudentGroup.studentId.groups
```

Out[90]:
```
{1: Int64Index([27, 29, 35, 37, 43], dtype='int64'),
 2: Int64Index([10, 14, 19, 20, 36, 38, 39, 40, 42], dtype='int64'),
 3: Int64Index([ 1,  4,  5,  6, 11, 12, 15, 17, 18, 24, 28, 30, 31, 34, 45, 46, 48,
            49, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68],
           dtype='int64'),
 4: Int64Index([ 0,  2,  3,  7,  8,  9, 13, 16, 21, 22, 23, 25, 26, 32, 33, 41, 44,
            47, 50, 51, 52, 53, 54, 55, 56],
           dtype='int64')}
```

In [ ]:
```

```

In [96]:
```
sCluster4 = singleLinkStudentClustNorm[['physics', 'maths', 'english', 'music', 'clusters']].groupby(['clusters']).get_group(
sCluster3 = singleLinkStudentClustNorm[['physics', 'maths', 'english', 'music', 'clusters']].groupby(['clusters']).get_group(
sCluster2 = singleLinkStudentClustNorm[['physics', 'maths', 'english', 'music', 'clusters']].groupby(['clusters']).get_group(
sCluster1 = singleLinkStudentClustNorm[['physics', 'maths', 'english', 'music', 'clusters']].groupby(['clusters']).get_group(

del sCluster4['clusters']
del sCluster3['clusters']
del sCluster2['clusters']
del sCluster1['clusters']

slKM4 = KMeans(n_clusters=1).fit(sCluster4)
slKM3 = KMeans(n_clusters=1).fit(sCluster3)
slKM2 = KMeans(n_clusters=1).fit(sCluster2)
slKM1 = KMeans(n_clusters=1).fit(sCluster1)

print("Single Link Cluster Center 4", slKM4.cluster_centers_)
print("Single Link Cluster Center 3", slKM3.cluster_centers_)
print("Single Link Cluster Center 2", slKM2.cluster_centers_)
print("Single Link Cluster Center 1", slKM1.cluster_centers_)
```

```
Single Link Cluster Center 4 [[ 1.09803 -1.98429 -2.48195 -0.51827]]
Single Link Cluster Center 3 [[-1.70464 -1.63817  1.69499 -0.65906]]
Single Link Cluster Center 2 [[ 0.05259  0.03387  0.0749   0.04814]]
Single Link Cluster Center 1 [[-1.40599  0.71052 -2.04072 -0.97584]]
```

```
NameError: name 'KM' is not defined
```

In [99]:

```
cCluster4 = completeLinkStudentClustNorm[['physics', 'maths', 'english', 'music', 'clusters']].groupby(['clusters']).get_grou
cCluster3 = completeLinkStudentClustNorm[['physics', 'maths', 'english', 'music', 'clusters']].groupby(['clusters']).get_grou
cCluster2 = completeLinkStudentClustNorm[['physics', 'maths', 'english', 'music', 'clusters']].groupby(['clusters']).get_grou
cCluster1 = completeLinkStudentClustNorm[['physics', 'maths', 'english', 'music', 'clusters']].groupby(['clusters']).get_grou

del cCluster4['clusters']
del cCluster3['clusters']
del cCluster2['clusters']
del cCluster1['clusters']

KM4 = KMeans(n_clusters=1).fit(cCluster4)
KM3 = KMeans(n_clusters=1).fit(cCluster3)
KM2 = KMeans(n_clusters=1).fit(cCluster2)
KM1 = KMeans(n_clusters=1).fit(cCluster1)

print("Complete Link Cluster Center 4", KM4.cluster_centers_)
print("Complete Link Cluster Center 3", KM3.cluster_centers_)
print("Complete Link Cluster Center 2", KM2.cluster_centers_)
print("Complete Link Cluster Center 1", KM1.cluster_centers_)
```

3]:

```
#if 1 & 2 both points are in same cluster, then add 1 to a. If 1 and 2 were in same clusters and now in different clusters,
# then add 1 to b. If 1 and 2 were in different clusters and now in same cluster, then add 1 to c. If both are in different
#clusters the add 1 to d.
a = 0
b = 0
c = 0
d = 0

print("single link length = ", range(len(singleLink)))

for i in range(len(singleLink)):
    print("abcde =", i)
    j = i + 1;
    for j in range(j, len(singleLink)):
        print("j =",j)
        # if 1 and 2 belong to different group and now to the same group then
        print("singlelink ", singleLink[j] )
        print("completeLink ", completeLink[j] )
        if singleLink[i] == singleLink[j] and completeLink[i] == completeLink[j]:
            a = a + 1
            print ("a =", a)
        if singleLink[i] != singleLink[j] and completeLink[i] != completeLink[j]:
            d = d + 1
            print ("d =", d)
        if singleLink[i] == singleLink[j] and completeLink[i] != completeLink[j]:
            c = c + 1
            print ("c =", c)
        if singleLink[i] != singleLink[j] and completeLink[i] == completeLink[j]:
            b = b + 1
            print ("b =", b)

print ("a =", a)
print ("b =", b)
print ("c =", c)
print ("d =", d)

randIndex = (a + d) / (a + b + c + d)
print("randIndex = ", randIndex)
```

In [188]:

```
kmeansClusters = np.ndarray((69,), buffer=np.array([1, 1, 3, 0, 2, 2, 1, 3, 3, 0, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1, 3, 1
...            offset=0,
...            dtype=int)
```

In [189]:

```
singleLink
```

Out[189]:

```
array([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, 4, 2, 1, 2, 2, 1, 2, 3, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], dtype=int32)
```

In [190]:

```
kmeansClusters
```

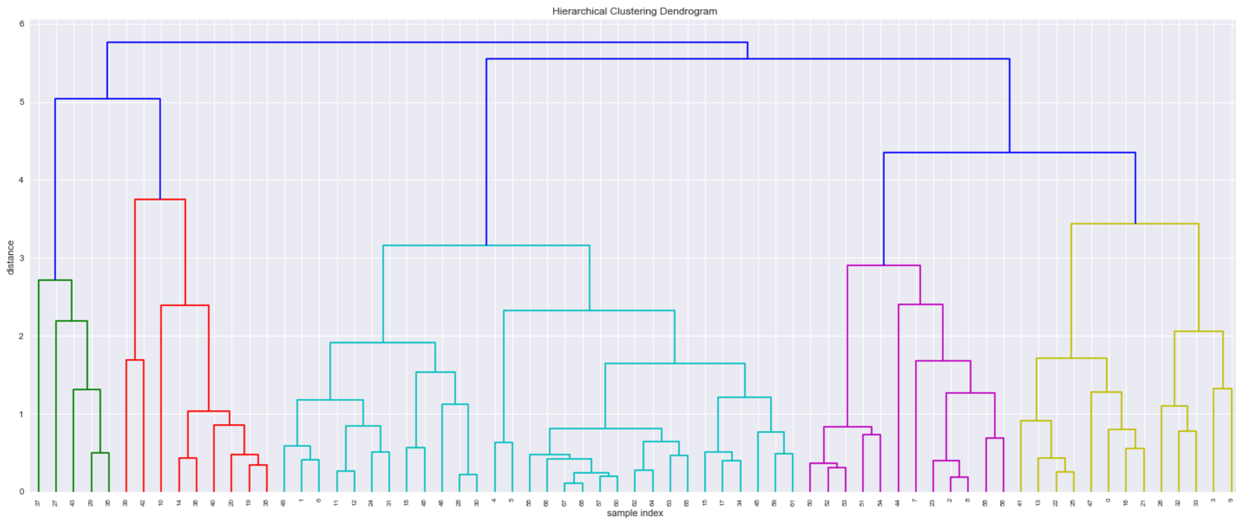
Out[190]:

```
array([1, 1, 3, 0, 2, 2, 1, 3, 3, 0, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1,
       3, 1, 1, 0, 3, 2, 0, 2, 2, 0, 0, 2, 0, 1, 0, 1, 0, 1, 1, 0, 0, 3, 2,
       2, 1, 1, 1, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

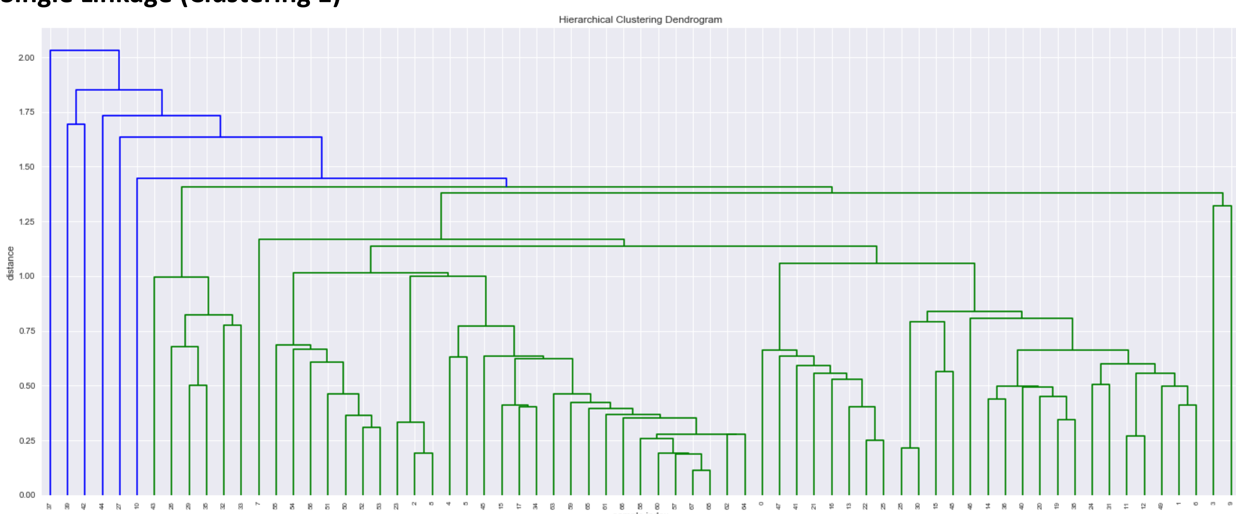
a. Dendrograms for the two clusterings (Clustering-2 and Clustering-3)

Answer:

Complete Linkage (Clustering 2)



Single Linkage (Clustering 1)



b. Cluster compositions for each case when we need only four clusters. Write ALL the data points included in each cluster and compute their centroids.

Answer:

Complete Linkage Cluster Composition and Centroids

Python code

4	5	-0.372224	1.180257	-0.717045	1.241609	3

```
In [88]:
1 singleLinkStudentGroup = singleLinkStudentClustNorm.groupby(singleLinkStudentClustNorm['clusters'])
2 completeLinkStudentGroup = completeLinkStudentClustNorm.groupby(completeLinkStudentClustNorm['clusters'])
```

```
In [90]:
1 completeLinkStudentGroup.studentId.groups
```

Output: Different groups are displayed below using Python index from 0 to 68.

```
{1: Int64Index([27, 29, 35, 37, 43], dtype='int64'),
```

```
 2: Int64Index([10, 14, 19, 20, 36, 38, 39, 40, 42], dtype='int64'),

 3: Int64Index([ 1,  4,  5,  6, 11, 12, 15, 17, 18, 24, 28, 30, 31, 34, 45
, 46, 48,
              49, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68],
             dtype='int64'),

 4: Int64Index([ 0,  2,  3,  7,  8,  9, 13, 16, 21, 22, 23, 25, 26, 32, 33
, 41, 44,
              47, 50, 51, 52, 53, 54, 55, 56],
             dtype='int64')}
```

Population of each cluster

```
dtype='int64')}
```

In [121]:

```
#For each cluster, update centroids
#cdist(studentGroup, cent, 'euclidean'.groups()
completeLinkStudentGroup.count()
```

Out[121]:

clusters	studentId	physics	maths	english	music	clusters
1	5	5	5	5	5	5
2	9	9	9	9	9	9
3	30	30	30	30	30	30
4	25	25	25	25	25	25

```
Complete Link Cluster Center 4 [[-0.62768 -0.74616  0.44779 -0.25077]]
Complete Link Cluster Center 3 [[ 0.44254  0.62316  0.31248  0.58928]]
Complete Link Cluster Center 2 [[ 0.55179  1.00445 -1.21383 -1.17529]]
Complete Link Cluster Center 1 [[-0.51006 -1.81618 -1.92894 -0.1663 ]]
```

Single Linkage Cluster Composition and Centroids

Population of each cluster

```
In [122]: singleLinkStudentGroup.count()
```

Out[122]:

clusters	studentId	physics	maths	english	music	clusters
1	2	2	2	2	2	2
2	65	65	65	65	65	65
3	1	1	1	1	1	1
4	1	1	1	1	1	1

Output: Different groups are displayed below using Python index from 0 to 68.

```
{1: Int64Index([39, 42], dtype='int64'),
 2: Int64Index([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14
, 15, 16,
             17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 3
2, 33,
             34, 35, 36, 38, 40, 41, 43, 45, 46, 47, 48, 49, 50, 51, 52, 5
3, 54,
             55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68],
            dtype='int64'),
 3: Int64Index([44], dtype='int64'),
 4: Int64Index([37], dtype='int64')}
```

Centroids for each cluster:

```
Single Link Cluster Center 4 [[ 1.09803 -1.98429 -2.48195 -0.51827]]
Single Link Cluster Center 3 [[-1.70464 -1.63817  1.69499 -0.65906]]
Single Link Cluster Center 2 [[ 0.05259  0.03387  0.0749   0.04814]]
Single Link Cluster Center 1 [[-1.40599  0.71052 -2.04072 -0.97584]]
```

c. Comment on any differences in the cluster centers and cluster compositions for the two different clusterings as performed in (b) above.

Answer:

Single link cluster use Minimum Euclidean distance to group different points together whereas complete link uses maximum Euclidean distance to group different points together. Hence, in single link clustering, many points are a part of cluster 2, whereas in complete link clustering, the points are more evenly distributed between cluster 1 and 2, and cluster 3 and 4. Single link identifies cluster 2 as the dominant cluster as the pairwise distance of points in this cluster is minimum as compared to other clusters in single link.

d. Compute Rand Index for the comparison of Clustering-2 and Clustering-3 and show the counts a, b, c, and d as determined for computing the Rand index. Explain the meaning of each count and why such counts have been obtained for this dataset and their clusterings.

Answer:

Rand Index computation Python code for a, b, c, and d

```python
#if 1 & 2 both points are in same cluster, then add 1 to a. If 1 and 2 were in same clusters and now in different clusters,
# then add 1 to b. If 1 and 2 were in different clusters and now in same cluster, then add 1 to c. If both are in different
#clusters the add 1 to d.
a = 0
b = 0
c = 0
d = 0

print("single link length = ", range(len(singleLink)))

for i in range(len(singleLink)):
    print("abcde =", i)
    j = i + 1;
    for j in range(j, len(singleLink)):
        print("j =",j)
        # if 1 and 2 belong to different group and now to the same group then
        print("singlelink ", singleLink[j] )
        print("completeLink ", completeLink[j] )
        if singleLink[i] == singleLink[j] and completeLink[i] == completeLink[j]:
            a = a + 1
            print ("a =", a)
        if singleLink[i] != singleLink[j] and completeLink[i] != completeLink[j]:
            d = d + 1
            print ("d =", d)
        if singleLink[i] == singleLink[j] and completeLink[i] != completeLink[j]:
            c = c + 1
            print ("c =", c)
        if singleLink[i] != singleLink[j] and completeLink[i] == completeLink[j]:
            b = b + 1
            print ("b =", b)

print ("a =", a)
print ("b =", b)
print ("c =", c)
print ("d =", d)

randIndex = (a + d) / (a + b + c + d)
print("randIndex = ", randIndex)
```

```
a = 739
b = 42
c = 1342
d = 223
randIndex =  0.4100596760443308
```

Difference explained: Single link cluster use Minimum Euclidean distance to group different points together whereas complete link use maximum Euclidean distance to group different points together. The rand index shows that

- 739 pair of points for both single linkage and complete link belong to the same cluster in both single link and complete link clustering.
- 42 pair of points that were in the different groups in single link are found in the same group in complete link clustering.
- 1342 pair of points that were in same group in single link and are found in different group in complete link.
- 223 pair of points were in different group in single link and are still in different group in complete linkage.

The randIndex of 0.41 shows that there is less similarity between clusters obtained from single link and complete link clustering. A rand index of 1 implies exactly same. As we approach 0, the similarity decreas ed. 0 indicates that none of the clusters obtained from the clustering techniques are completely dissimilar.

3. Compute Rand Index for the comparison of Clustering-1 and Clustering-2 and show the counts a, b, c, and d as determined for computing the Rand index. Explain the meaning of each count and why such counts have been obtained for this dataset and these clusterings in this comparison.

Answer:

Rand Index for kmeans Clustering 1 and single Link Clustering 2

```
In [189]: singleLink
Out[189]: array([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, 4, 2, 1, 2, 2, 1, 2, 3, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2], dtype=int32)

In [190]: kmeansClusters
Out[190]: array([1, 1, 3, 0, 2, 2, 1, 3, 3, 0, 1, 1, 1, 1, 1, 2, 1, 2, 1, 1, 1, 1, 1,
       3, 1, 1, 0, 3, 2, 0, 2, 2, 0, 0, 2, 0, 1, 0, 1, 0, 1, 1, 0, 0, 3, 2,
       2, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])

In [191]:
#if 1 & 2 both points are in same cluster, then add 1 to a. If 1 and 2 were in same clusters and now in different clusters,
# then add 1 to b. If 1 and 2 were in different clusters and now in same cluster, then add 1 to c. If both are in different
#clusters the add 1 to d.
a = 0
b = 0
c = 0
d = 0

for i in range(len(singleLink)):
    print("abcde =", i)
    j = i + 1;
    for j in range(j, len(singleLink)):
        print("j =",j)
        # if 1 and 2 belong to different group and now to the same group then
        print("singlelink ", singleLink[j] )
        print("kmeansCluster ", kmeansClusters[j] )
        if singleLink[i] == singleLink[j] and kmeansClusters[i] == kmeansClusters[j]:
            a = a + 1
            print ("a =", a)
        if singleLink[i] != singleLink[j] and kmeansClusters[i] != kmeansClusters[j]:
            d = d + 1
            print ("d =", d)
        if singleLink[i] == singleLink[j] and kmeansClusters[i] != kmeansClusters[j]:
            c = c + 1
            print ("c =", c)
        if singleLink[i] != singleLink[j] and kmeansClusters[i] == kmeansClusters[j]:
            b = b + 1
            print ("b =", b)

print ("a =", a)
print ("b =", b)
print ("c =", c)
print ("d =", d)

randIndex = (a + d) / (a + b + c + d)
print("randIndex = ", randIndex)
```

```
a = 579
b = 38
c = 1502
d = 227
randIndex =  0.3435635123614663
```

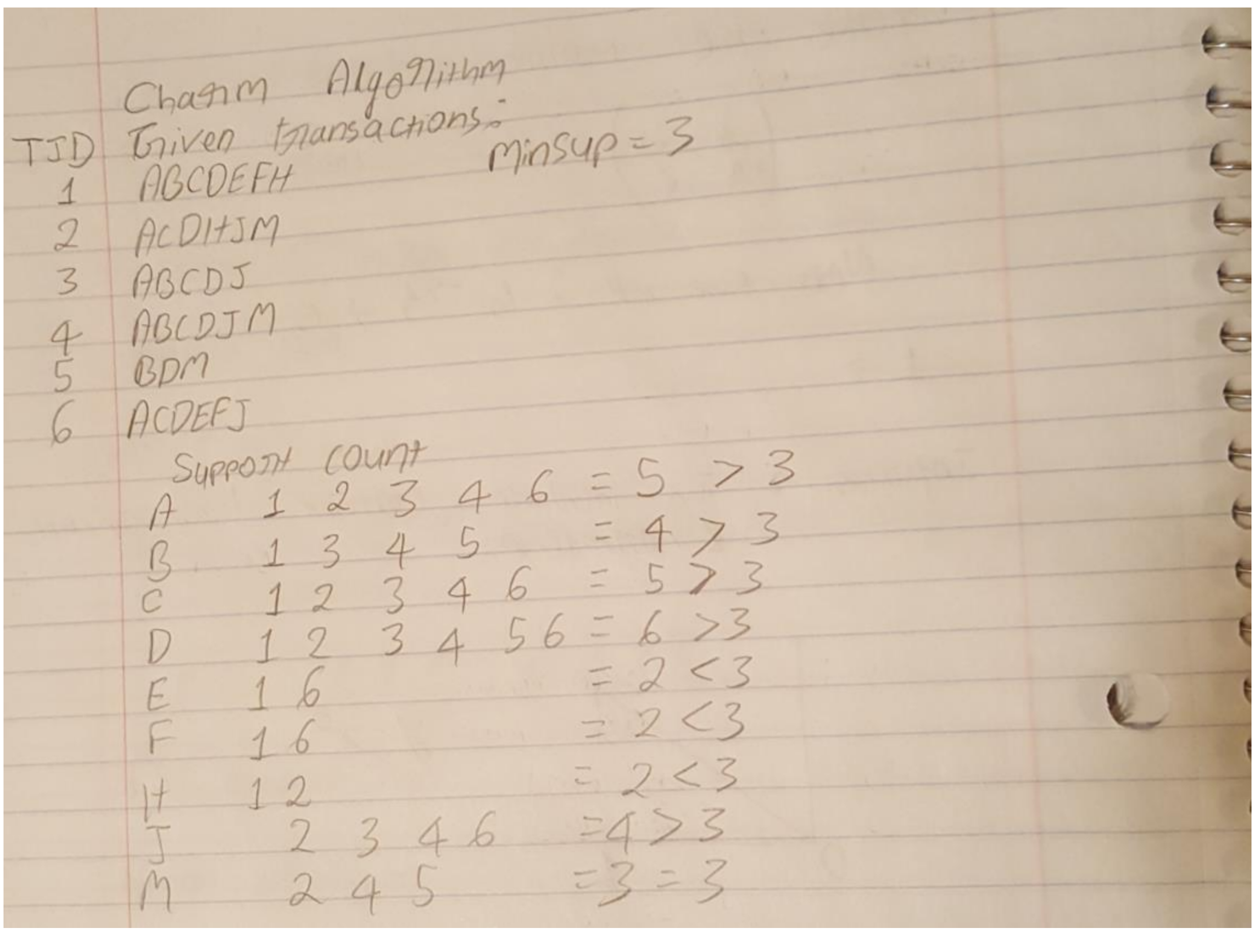
Difference explained: Single link cluster use Minimum Euclidean distance to group different points together whereas complete link use maximum Euclidean distance to group different points together. The rand index shows that

- 579 pair of points for both single linkage and kmeans clustering belong to the same cluster.
- 38 pair of points that were in the different groups in single link are found in the same group in k means clustering.
- 1502 pair of points that were in same group in single link and are found in different group in kmeans clustering.
- 227 pair of points were in different group in single link and are still in different group in kmeans.

The randIndex of 0.34 shows that there is less similarity between clusters obtained from single link and k means clustering. As seen above there are 1502 points which were in the same cluster in single link, but in different clusters in complete link. This value is high as compared to other values, and indicates that t he two clusters are not similar. A rand index of 1 implies exactly same. As we approach 0, the similarity d ecreased. 0 indicates that none of the clusters obtained from the clustering techniques are completely d issimilar.

4. Show the execution tree for the CHARM algorithm for finding all the closed itemsets for the dataset containing the following transactions: ABCDEFH, ACDHJM, ABCDJ, ABCDJM, BDM, ACDEFJ.

Answer:
MD, D, BD, JACD, BACD, ACD



Charm Algorithm

Given transactions: minsup = 3

TID	
1	ABCDEFH
2	ACDHJM
3	ABCDJ
4	ABCDJM
5	BDM
6	ACDEFJ

Support count

A	1 2 3 4 6	= 5 > 3
B	1 3 4 5	= 4 > 3
C	1 2 3 4 6	= 5 > 3
D	1 2 3 4 5 6	= 6 > 3
E	1 6	= 2 < 3
F	1 6	= 2 < 3
H	1 2	= 2 < 3
J	2 3 4 6	= 4 > 3
M	2 4 5	= 3 = 3

We will sort the items based on their support count in ascending order

M B J A C D
245 1345 2346 12346 12346 123456

MD 245 | BA 134 | BC 134 | BD 1345 | JA 2346 | JC 2346 | JD 2346 | AC 12346 | AD 12346

BAC 134 | BAD 134 | JACD 2346 | ACD 12346

BACD 134

As per property 2, we cannot prune a subset that contains different transaction id. Hence, we cannot prune BD, with a different transaction id, and a subset of BACD.

Our closed itemsets are: MD, D, BD, JACD, BACD, ACD

5. For the same data as in #4 above, show execution of the algorithm for finding all the maximal itemsets.

Answer:

Maximal frequent itemset = ABCD, ACDJ, DM

TID Given Transactions

TID	Items
1	ABCDEFH
2	ACDHJM
3	ABCDJ
4	ABCDJM
5	BDM
6	ACDEFJ

minsup = 3

Support count

A	1 2 3 4 6	= 5 > 3
B	1 3 4 5	= 4 > 3
C	1 2 3 4 6	= 5 > 3
D	1 2 3 4 5 6	= 6 > 3
E	1 6	= 2 < 3
F	1 6	= 2 < 3
H	1 2	= 2 < 3
J	2 3 4 6	= 4 > 3
M	2 4 5	= 3 = 3

A	B	C	D	J	M
12346	1345	12346	123456	2346	245

AB	AC	AD	AJ	AM	BC	BD	BJ	BM	CD	CJ	CM	DJ	DM	JM
134	12346	12346	2346	24	134	1345	34	45	12346	2346	24	2346	245	24

ABC	ABD	ABJ	ACD	ACJ	ADJ	BCD	CDJ	DJM
134	134	34	12346	2346	2346	134	2346	24

ABCD	ACDJ
134	2346

ACDJ is a max frequent itemset. As ADJ is a subset of ACDJ, we prune ADJ. Similarly, CDJ is a subset of ACDJ, hence we prune CDJ.
BCD is a subset of ABCD, hence we prune BCD also.

∴ Maximal frequent itemset = {ABCD, ACDJ, DM}