Assignment 6

Hierarchical Clustering

Manish Mondal

ID- 700713864

GitHub Link : https://github.com/manish-mondal/MachineLearning.git

Video Link : https://drive.google.com/drive/folders/1j3-osgYS-TFHPLAlO5XcVFBnD7GFqJHr?usp=share_link

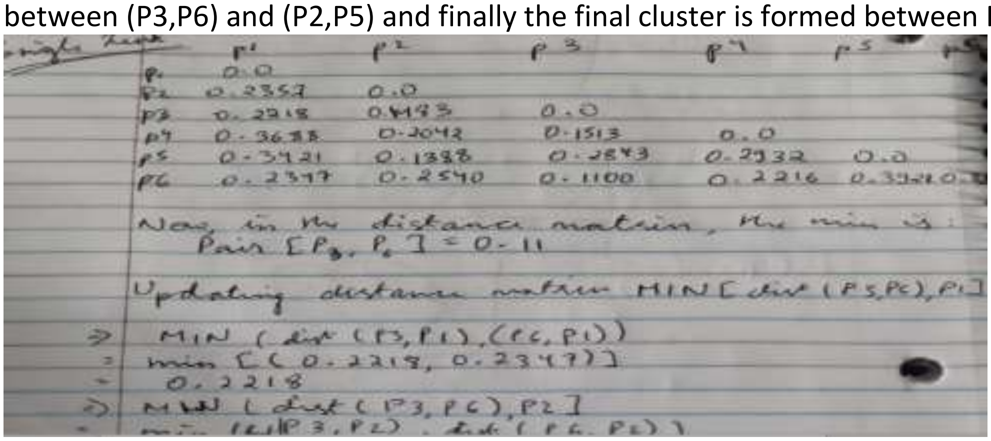
Question 1:

The x coordinate and y coordinate is given in the question. For our ease, the distance matrix is also provided which makes calculations less.

We are asked to calculate and the clustering representation for Hierarchical clustering and also draw dendograms for Single, Complete and Average links.

For the Single Link, We take the minimum of the points of the clusters that has to be merged and continue the process till only 1 cluster remains.

Here, 1st cluster is formed for pairs P3, P6, 2nd cluster is formed for P2, P5, 3rd cluster is formed between (P3,P6) and (P2,P5) and finally the final cluster is formed between P2,P3,P5,P5 and P4.



	P1	P2	P3	P4	P5	P6
P1	0.0					
P2	0.2357	0.0				
P3	0.2218	0.1483	0.0			
P4	0.3688	0.2042	0.1513	0.0		
P5	0.3421	0.1388	0.2843	0.2932	0.0	
P6	0.2347	0.2540	0.1100	0.2216	0.3921	0.0

Now, in the distance matrix, the min is :
Pair [P3, P6] = 0.11

Updating distance matrix MIN[dist (P3,P6), P1]

=> MIN (dist (P3, P1), (P6, P1))
= min [(0.2218, 0.2347)]
= 0.2218
=> MIN [dist (P3, P6), P2]
= min (dist(P3, P2), dist (P6, P2))

= min (0.1483 , 0.2340)
= 0.1483

=> MIN [dist (P3, P6), P4)]
= min [dist (P3, P4) , (P6, P4)]
= min (0.1513 , 0.2216)
= 0.1513

=> MIN [dist (P3, P6) , P5]
= min (dist (P5, P3) , (P6, P5)
= min (0.2843 , 0.3521)
= 0.2843

Updated distance matrix for P3, P6

	P1	P2	P3,P6	P4	P5
P1	0				
P2	0.2357	0			
P3,P6	0.2218	0.1483	0		
P4	0.3683	0.2042	0.1513	0	
P5	0.3421	0.1388	0.2843	0.2932	0

The min dist in distance matrix is
Pair [P2, P5] : 0.1388

=> MIN [dist (P2, P5), P1]
= min (dist (P2, P1) , (P5, P1))
= min (0.2357 , 0.3421)
= 0.2357

=> MIN [dist (P2, P5), (P3, P6)]
= min (dist (P2 (P3, P6), P5 (P3, P6)]

Updated distance matrix is

	P1	P2,P5,P3,P6	P4
P1	0		
P2,P5,P3,P6	0.2218	0	
P4	0.3683	0.1513	0

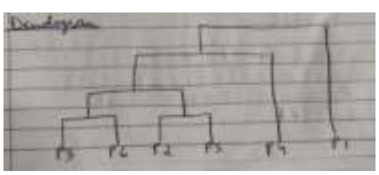
The min distance is for cluster
P min([P3, P5, P2, P6], P4] = 0.1513

~~[illegible]~~
min (0.2218 ,

MIN [dist (P2, P5, P3, P6), P1)]
= min [dist ((P2, P5, P3, P6), P1) , [illegible] (P1, P4)
= min (0.2218 , 0.3683)
= 0.2218

Final Distance matrix for cluster is :

	P1	P2, P5, P3, P6, P4
P1	0	
P2,P5,P3,P6,P4	0.2218	0



For the Complete Link, we have selected the minimum point to begin the clustering. We take the maximum of the two minimum data points distances and then start creating clusters depending on the total no. of clusters we need.

For this assignment, the clusters are formed as :
P3 and P6, then between P2 and P5, then, (P3,P6) and P4. Then between (P5,P2) and (P3,P4,P6).

Complete Link:

	P1	P2	P3	P4	P5	P6
P1	0					
P2	0.2357	0				
P3	0.2218	0.1483	0			
P4	0.3688	0.2042	0.1513	0		
P5	0.3421	0.1388	0.2843	0.2932	0	
P6	0.2347	0.2540	0.1100	0.2216	0.3921	0

The lowest is Pair (P3, P6) = 0.1100

MAX[dist (P3, P6), P1]]
max[(P3, P1), (P6, P1)]
max(0.2218 , 0.2347)
0.2347

MAX[dist (P3, P6), P2]
max(dist (P3, P2), (P6, P2))
max (0.1483 , 0.2540]
0.2540

MAX[dist (P3, P6], P4]
max(dist (P3, P4) , (P6, P4)]
max (0.1513 , 0.2216)
0.2216

MAX[dist (P3, P6), P5)]
max(dist (P3, P5) , (P6, P5))
max(0.2843, 0.3921)
0.3921

	P1	P2	P3, P6	P4	P5
P1	0				
P2	0.2357	0			
P3, P6	0.2347	0.2540	0		
P4	0.3688	0.2042	0.2216	0	
P5	0.3421	0.1388	0.3921	0.2932	0

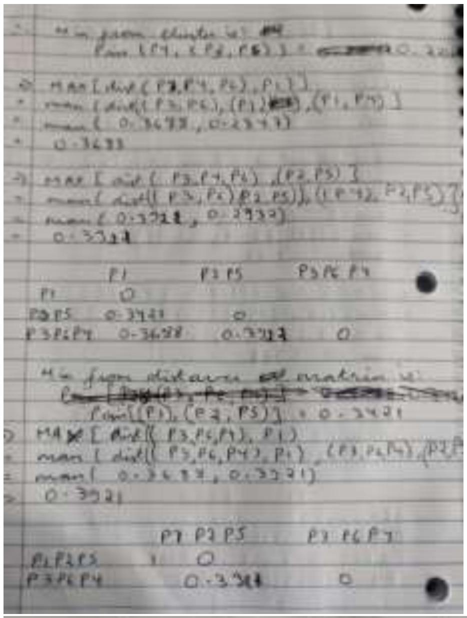
=> The lowest is Pair (P2, P5) = 0.1388

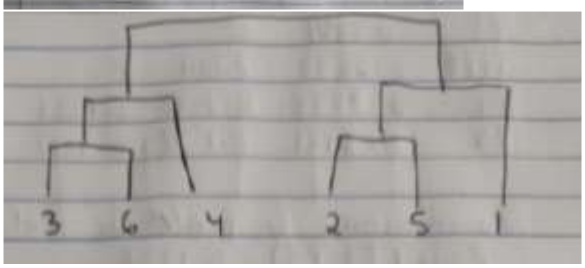
=> MAX[dist (P2, P5), P1)
= max(dist((P2, P1) , (P5, P1))
= max (0.2357, 0.3421)
= ~~max~~ 0.3421

=> MAX [dist (P2, P5), (P3, P6))
= max [dist(P2, (P3, P6)) , (P5, (P3, P6))]
= max (0.2540 , 3.3921)
= ~~max~~ 0.3921

=> MAX [dist (P2, P5), P4]
= max (dist (P2, P4) , (P5, P4))
= max (0.2042, 0.2932)
= 0.2932

	P1	P2 P5	P3 P6	P4
P1	0			
P2, P5	0.3421	0		
P3, P6	0.2347	0.3921	0	
P4	0.3688	0.2932	0.2216	0





For the Average Link, This link takes the average between the distances of two minimum data points to form clusters.

Here, the clusters formed for this assignment are P3 and P6, then P2,P5, then between P4 and (P3 and P5). Then between P1 and (P2,P5). Finally it ends with (P1,P2,P5) and (P3,P6,P4). The screenshots and dendrogram is attached below.

Average Link

Minimum distance matrix is for cluster
Pair [P3, P6] = 0.11

=> Avg [(P3, P6), P1]
= Avg (0.2218, 0.2347)
= 0.2282

=> Avg [(P3, P6), P2]
= Avg (0.1483, 0.2540)
= 0.2011

=> Avg [(P3, P6), P4]
= Avg (0.1513, 0.2216)
= 0.1864

=> Avg [(P3, P6), P5]
= Avg (0.2843, 0.3921)
= 0.3382

	P1	P2	P3P6	P4	P5
P1	0				
P2	0.2357	0			
P3P6	0.2282	0.2011	0		
P4	0.3688	0.2042	0.1864	0	
P5	0.3421	0.1388	0.3382	0.2932	0

The minimum in matrix distance is:
Pair (P2, P5) = 0.1388

=> Avg [dist (P2, P5) P1]
= Avg [dist (P2, P1), (P5, P1)]
= avg (0.2357, 0.3421)
= 0.2889

=> Avg [dist (P2, P5), (P3, P6)]
= avg [dist [(P3, P6), P2], ((P3, P6), P5)]
= avg (0.2011, 0.3382)
= 0.2946

=> Avg [dist (P2, P5), P4)]
= avg (dist (P2, P4), (P5, P4)]
avg (0.2042, 0.2932)
0.2487

	P1	P5P2	P3P6	P4
P1	0			
P2P5	0.2889	0		
P3P6	0.2282	0.2946	0	
P4	0.3688	0.2487	0.1864	0

The minimum dist matrix is for cluster:
Pair [(P3, P6), P4] = 0.1364

=> Avg [dist (P3, P4, P6), (P1)]
= avg [dist((P3, P6), P1), (P4, P1)]
= avg (0.2282, 0.3688)
= 0.2985

=> Avg [dist (P3, P4, P6), (P2, P5)]
= avg [dist ((P3, P6), P2, P5), ((P4), P2, P5)]
= avg (0.2976, 0.2487)
= 0.2591

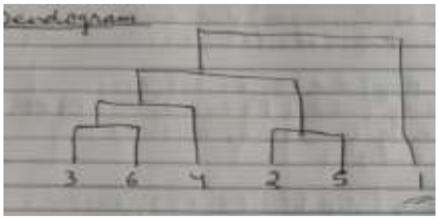
	P1	P2 P5	P3 P4 P6
P1	0		
P2 P5	0.2889	0	
P3 P4 P6	0.2985	0.2591	0

= Min dist matrix is for cluster
Pair [(P5, P2), (P3, P4, P6)] = 0.2591

=> Avg [dist (P2, P3, P4, P5, P6), P1)]
= avg (dist ((P5, P2), P1)), ((P3, P4, P6), P1)]
= avg (0.2889, 0.2985)
= 0.2937

	P1	P2 P3 P4 P5 P6
P1	0	
P2 P3 P4 P5 P6	0.2937	0

Dendrogram



Question 2

we are supposed to analyze the dataset CC General and then perform basic data preprocessing, then normalize and scale the data and the apply agglomerative clustering with different no. of clusters and finally calculation the silhouette score to analyze the accuracy of the mode.

a) Firstly, I have done the preprocessing by removing null values and replacing it by mean of the column and then dropping the categorical column

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

```
In [105]: df.shape         #Getting the no. of attributes in the dataset

Out[105]: (8950, 18)

In [106]: df.isnull().sum()     #Checking for null values

Out[106]: CUST_ID                               0
          BALANCE                               0
          BALANCE_FREQUENCY                     0
          PURCHASES                             0
          ONEOFF_PURCHASES                      0
          INSTALLMENTS_PURCHASES                0
          CASH_ADVANCE                          0
          PURCHASES_FREQUENCY                   0
          ONEOFF_PURCHASES_FREQUENCY            0
          PURCHASES_INSTALLMENTS_FREQUENCY      0
          CASH_ADVANCE_FREQUENCY                0
          CASH_ADVANCE_TRX                      0
          PURCHASES_TRX                         0
          CREDIT_LIMIT                          1
          PAYMENTS                              0
          MINIMUM_PAYMENTS                    313
          PRC_FULL_PAYMENT                      0
          TENURE                                0
          dtype: int64

In [107]: df = df.drop('CUST_ID', axis = 1)        #Dropping the categorical column

In [108]: df = df.fillna(df.mean())          #Filling the missing values

In [109]: df.head()          #Checking again after dropping customer ID
```

b) Then, I have scaled the data using Standard Scaler and then normalized the raw data points using the normalize() function

```
In [111]: scaler = StandardScaler()
          X_Scale = scaler.fit_transform(df)         # Scaling the dataset
          print(X_Scale)

          [[-0.73198937 -0.24943448 -0.42489974 ... -0.31096755 -0.52555097
             0.36067954]
           [ 0.78696085  0.13432467 -0.46955188 ...  0.08931021  0.2342269
             0.36067954]
           [ 0.44713513  0.51808382 -0.10766823 ... -0.10166318 -0.52555097
             0.36067954]
           ...
           [-0.7403981  -0.18547673 -0.40196519 ... -0.33546549  0.32919999
            -4.12276757]
           [-0.74517423 -0.18547673 -0.46955188 ... -0.34690648  0.32919999
            -4.12276757]
           [-0.57257511 -0.88903307  0.04214581 ... -0.33294642 -0.52555097
            -4.12276757]]

In [112]: normal = preprocessing.Normalizer().fit(X_Scale)       #Normalizing the raw input data
          X_Norm = normal.transform(X_Scale)
          print(X_Norm)

          [[-0.31193826 -0.10629684 -0.1810716  ... -0.13251924 -0.22396426
             0.15370408]
           [ 0.21992533  0.03753859 -0.13122171 ...  0.02495877  0.06545742
             0.10079608]
           [ 0.12668203  0.14678317 -0.03050449 ... -0.02880315 -0.14889876
             0.10218749]
           ...
           [-0.1569743  -0.03932355 -0.085222   ... -0.07112317  0.0697948
            -0.87408185]
           [-0.15431961 -0.03841074 -0.09724043 ... -0.07184155  0.06817468
```

```python
In [113]: X_normalized = pd.DataFrame(X_Norm)          #Converting from array to panda
          X_normalized.head()
```

Out[113]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
0	-0.311938	-0.106297	-0.181072	-0.162108	-0.148760	-0.198921	-0.343687	-0.289212	-0.301422	-0.287801	-0.202878	-0.217905	-0.409290	-0.226425	-1.32
1	0.219925	0.037539	-0.131222	-0.099749	-0.127037	0.728166	-0.341434	-0.189660	-0.256265	0.160401	0.030761	-0.165384	0.192448	0.228779	2.49
2	0.126682	0.146783	-0.030504	0.030850	-0.128790	-0.132249	0.358771	0.757440	-0.259802	-0.191339	-0.134880	-0.030888	0.234039	-0.108739	-2.88
3	0.020589	-0.426439	0.097309	0.229034	-0.190618	-0.154587	-0.425253	-0.167447	-0.384524	-0.108570	-0.138184	-0.231288	0.346393	-0.251048	-1.841
4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	-1.12

c) Now, I have used PCA and reduced the attributes to 2 dimensions.

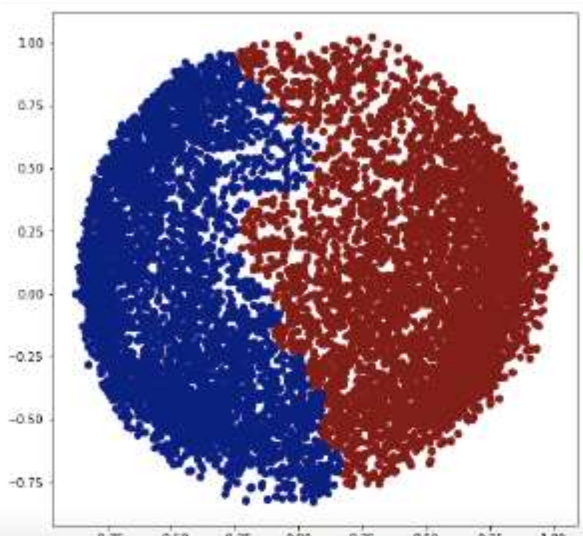
```python
In [114]: pca2 = PCA(n_components=2)
          principalComponents = pca2.fit_transform(X_Norm)          #Applying PCA and reducing the no. features to 2

          principalDf = pd.DataFrame(data = principalComponents, columns = ['principal component 1', 'principal component 2'])

          principalDf.head()
```

Out[114]:

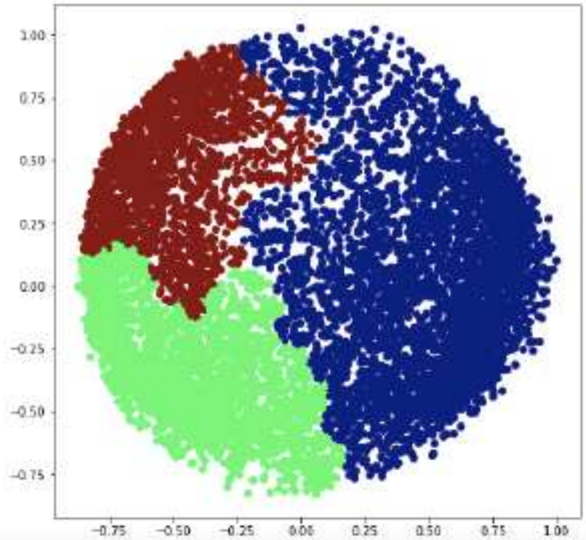
	principal component 1	principal component 2
0	-0.489826	-0.679678
1	-0.518792	0.545011
2	0.330865	0.268976
3	-0.482374	-0.092111
4	-0.563269	-0.481915

d) After applying PCA, I have now used scatter plot to visualize the agglomerative clustering using by taking k(no. of clusters) as 2,3,4,5

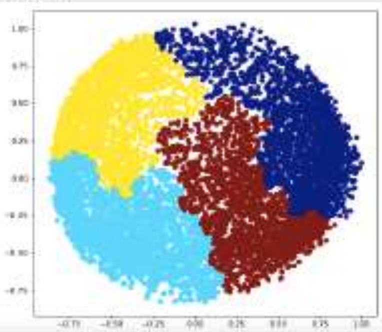
```python
In [115]: # Using scatter plot to visualize for k =2
          Agglo_Cluster2 = AgglomerativeClustering(n_clusters = 2)
          plt.figure(figsize =(8,8))
          plt.scatter(principalDf['principal component 1'], principalDf['principal component 2'],
          c = Agglo_Cluster2.fit_predict(principalDf), cmap ='jet')
          plt.show()
```



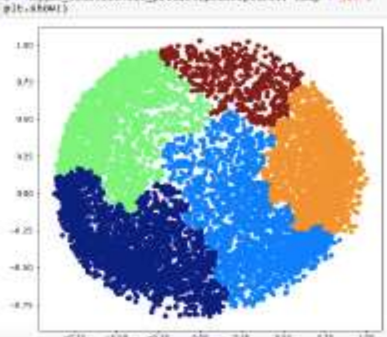
```
In [116]: # Using scatter plot to visualize for k =3
Agglo_Cluster3 = AgglomerativeClustering(n_clusters = 3)
plt.figure(figsize =(8,8))
plt.scatter(principalDf['principal component 1'], principalDf['principal component 2'],
c = Agglo_Cluster3.fit_predict(principalDf), cmap ='jet')
plt.show()
```




```
In [119]: # Using scatter plot to visualize for k = 4
Agglo_Cluster4 = AgglomerativeClustering(n_clusters = 4)
plt.figure(figsize =(8,8))
plt.scatter(principalDf['principal component 1'], principalDf['principal component 2'],
c = Agglo_Cluster4.fit_predict(principalDf), cmap ='jet')
plt.show()
```



```
In [118]: # Using scatter plot to visualize for k = 5
Agglo_Cluster5 = AgglomerativeClustering(n_clusters = 5)
plt.figure(figsize =(8,8))
plt.scatter(principalDf['principal component 1'], principalDf['principal component 2'],
c = Agglo_Cluster5.fit_predict(principalDf), cmap ='jet')
plt.show()
```

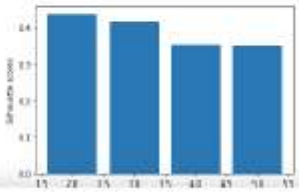


e) Finally, I have calculated the silhouette score for each clusters and them compared them using a bar plot.

```
print("Silhouetter score for 2 clusters is =", Silhouette[0])
Silhouette.append(silhouette_score(principalDf, Agglo_Cluster3.fit_predict(principalDf)))
print("Silhouetter score for 3 clusters is =", Silhouette[1])
Silhouette.append(silhouette_score(principalDf, Agglo_Cluster4.fit_predict(principalDf)))
print("Silhouetter score for 4 clusters is =", Silhouette[2])
Silhouette.append(silhouette_score(principalDf, Agglo_Cluster5.fit_predict(principalDf)))
print("Silhouetter score for 5 clusters is =", Silhouette[3])
```

```
Silhouetter score for 2 clusters is = 0.4373242834738189
Silhouetter score for 3 clusters is = 0.4167329587664966
Silhouetter score for 4 clusters is = 0.3531061921312974
Silhouetter score for 5 clusters is = 0.3505508240817038
```

```
In [21]: # Plotting a bar graph to compare the results
plt.bar(k, Silhouette)
plt.xlabel('Number of clusters', fontsize = 10)
plt.ylabel('Silhouette scores', fontsize = 10)
plt.show()
```



I can infer from this that using 2 cluster results in the best silhouette score, which is 43.7324%.